36th International Conference on Principles of Diagnosis and Resilient Systems

DX 2025, September 22–24, 2025, Nashville, TN, USA

Edited by

Marcos Quinones-Grueiro

Gautam Biswas

Ingo Pill

Editors

Marcos Quinones-Grueiro
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA
marcos.quinones.grueiro@Vanderbilt.Edu

Gautam Biswas
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA
gautam.biswas@Vanderbilt.Edu

Ingo Pill
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria
ingo.pill@gmail.com

ACM Classification 2012
Computing methodologies → Causal reasoning and diagnostics

ISBN 978-3-95977-394-2

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-394-2.

Publication date
November, 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists all publications of this volume in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/OASIcs.DX.2025.0

ISBN 978-3-95977-394-2 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Regular Papers

36th International Conference on Principles of Diagnosis and Resilient Systems (DX 2025).
Editors: Marcos Quinones-Grueiro, Gautam Biswas, and Ingo Pill
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Short Paper

DX Competitions

PhD Panel

Preface

In 2025, the 36th International Conference on Principles of Diagnosis and Resilient Systems (DX'25) was held from September 22nd to 24th in Nashville, United States, hosted by Vanderbilt University. The DX conference is a leading forum to present and discuss the latest research, experience reports, and emerging ideas related to diagnosis and resilient systems. DX is application-agnostic, meaning the research presented covers a wide range of systems, from physical to computational, represented either in abstract or detailed forms. Starting with the previous DX'24 edition and continuing onward, we expanded our focus from diagnosis to include resilience, which refers to a system's inherent ability to maintain its essential operations when faced with both expected and unforeseen challenges. DX welcomes papers that address resilient design as well as approaches for operational resilience.

Topics covered in the DX conference include, but are not limited to:

- Monitoring, detection, diagnosis, and mitigation of faults, unexpected issues, and anomalies
- Formal theories and symbolic, sub-symbolic, as well as hybrid approaches for diagnosis
- Data-driven and learning-enabled methods for monitoring and diagnosis
- Connections between diagnosis and other techniques such as decision-making, (re-)planning, (re-)configuration, control, formal verification, and testing.
- Concept papers on the theory of design and operational resilience
- Designing, developing, and operationalizing resilient systems
- Hardware and software instrumentation, as well as dependable data acquisition and probing
- Development, learning, abstraction, transformation, analysis, optimization, and transfer of diagnosis models
- Diagnosis in resilient, intelligent, and autonomous systems
- Diagnosis in a distributed, hierarchical, system-of-systems, or multi-agent context

For DX'25, we received 24 submissions in total, despite the feedback about the current economic difficulties in science and the reluctance to travel that we have been receiving from the DX community. The 24 submissions are split into 20 submissions by the regular deadline, and four submissions that were rejected but significantly revised and then resubmitted for the fast-track option. Out of the 20 original submissions, 1 (out of 1) was accepted for the PhD panel, and 2 (out of 2) were accepted for the DX competition track. Out of the remaining 17 submissions, 11 (64.7%) were accepted to be published with our DX'25 proceedings as regular papers. We received four resubmissions of rejected papers for the fast track and accepted one paper (25%) as regular paper and one (25%) as short paper. The overall acceptance rate is 66.7% (16 out of 24 papers).

The papers published in these proceedings demonstrate the broad and innovative scope of current research on diagnosis, prognosis, and resilient systems. Contributions include advancements in methodology – such as temporal reasoning, functional event calculus, hybrid system identification, multimodal and zero-shot learning, spectral receptive fields, and particle filtering – as well as the incorporation of large language models for developing diagnostic models, controller design, and fault detection. The application areas are just as varied, covering HVAC systems, radiotherapy machines, automotive die-casting, drones, and heat pumps. Several studies highlight benchmarking and evaluation frameworks, like the revived DX Competition and systematic analyses of diagnostic algorithms. Overall, these papers

36th International Conference on Principles of Diagnosis and Resilient Systems (DX 2025).
Editors: Marcos Quinones-Grueiro, Gautam Biswas, and Ingo Pill
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

highlight both the theoretical depth and practical applications of diagnosis research, pointing toward future integration of AI, multimodal reasoning, and resilience-by-design in complex engineered systems.

Organizing a conference is always a team effort, so we would like to thank a few people, starting with Gerilynn Pearce and Mary Margaret Sprinkle from our local support team at the Institute for Software Integrated Systems at Vanderbilt University. Special thanks to the Artificial Intelligence Journal for once again sponsoring DX'25 and to the program committee members for their reviews, which helped us select the right papers to create a high-quality technical program for the 36th International Conference on Principles of Diagnosis and Resilient Systems. We would like to specifically mention also our keynote speaker Sankaran Mahadevan, our two invited speakers Daniel Jung and Chetan Kukarni, the DX competition committee (see below), as well as the DX steering committee (see below) and the award committee chairs Louise Travé-Massuyès and Meir Kalech. We also appreciate Dagstuhl, our publisher, for collaborating with us to make the technical program available through open-access proceedings.

Finally, we would like to thank all the authors for contributing to the success of this conference. Without their work and their choice to present it at DX, this event would not have been possible.

The DX'25 chairs: Marcos Quinones-Grueiro, Gautam Biswas, Ingo Pill

Conference Organization

DX Steering Committee (in a.o.):

- Gautam Biswas (chair), Vanderbilt university, USA
- Johan de Kleer, c-infinity, USA
- Meir Kalech, Ben Gurion University of the Negev, Israel
- Oliver Niggemann, Helmut-Schmidt-Universität Hamburg, Germany
- Ingo Pill (chair), Graz University of Technology, Austria
- Louise Travé-Massuyès, LAAS-CNRS, France
- Franz Wotawa, Graz University of Technology, Austria
- Marina Zanella, Università di Brescia, Italy

DX Competition Committee (in a.o.):

- Johan de Kleer (Co-Chair)
- Jan Deeken
- Kai Dresia
- Erik Frisk
- Daniel Jung (Chair LiU-ICE Benchmark)
- Mattias Krysander
- Eldin Kurudzija (Chair LUMEN Bechmark)
- Ingo Pill (Chair)
- Michal Syfert
- Anna Sztyber-Betley (Chair SLIDe Benchmark)
- Tobias Traudt
- Günther Waxenegger-Wilfling

36th International Conference on Principles of Diagnosis and Resilient Systems (DX 2025).
Editors: Marcos Quinones-Grueiro, Gautam Biswas, and Ingo Pill
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Carlos Alonso-González (6)
University of Valladolid, Spain

Marlon J. Ares-Milian (10)
School of Computer Science, University College Cork, Ireland

Gautam Biswas (8, 11, 15)
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Elodie Chanthery (2)
LAAS-CNRS, INSA, University of Toulouse, France

Austin Coursey (8, 11, 15)
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Miguel Cubero (6)
University of Valladolid, Spain

Ankita Das (4)
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Johan de Kleer (5, 14)
c-infinity, Mountain View, CA, USA

Abel Diaz-Gonzalez (11, 15)
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Alexander Diedrich (2)
Helmut-Schmidt-University, Hamburg, Germany

Kai Dresia (14)
Institute of Space Propulsion, German Aerospace Center (DLR), Lampoldhausen, Germany

Goerschwin Fey (7)
Hamburg University of Technology, Germany

Maxence Glotin (2)
LAAS-CNRS, INSA, University of Toulouse, France

Chao Hu (8)
School of Mechanical, Aerospace, and Manufacturing Engineering, University of Connecticut, Storrs, CT, USA

Luis Ignacio Jiménez (6)
University of Valladolid, Spain

Daniel Jung (14)
Department of Electrical Engineering, Linköping University, Sweden

Roxane Koitz-Hristov (1, 4)
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Karol Kukla (2)
Warsaw, Poland

Chetan Kulkarni (13)
KBR Inc, NASA Ames Research Center, Moffett Field, CA, USA

Eldin Kurudzija (14)
Institute of Space Propulsion, German Aerospace Center (DLR), Köln, Germany

Daniel López (6)
HORSE Powertrain, Valladolid, Spain

Silke Merkelbach (2)
Fraunhofer IEM, Paderborn, Germany

Herbert Muehlburger (12)
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Stan Muñoz Gutiérrez (9)
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Oliver Niggemann (2)
Helmut-Schmidt-University, Hamburg, Germany

Ingo Pill (5, 14)
Institute of Software Engineering and Artificial Intelligence, TU Graz, Austria

Jérémy Pirard (16)
Airbus, Toulouse, France

Swantje Plambeck (7)
Hamburg University of Technology, Germany

Kélian Poujade (16)
Université de Toulouse, Oncopole Claudius Regaud, Institut Universitaire du Cancer de Toulouse (IUCT), France; Université de Toulouse, CNRS, INSERM, Centre de Recherches en Cancérologie de Toulouse (CRCT), France

36th International Conference on Principles of Diagnosis and Resilient Systems (DX 2025).
Editors: Marcos Quinones-Grueiro, Gautam Biswas, and Ingo Pill
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Liliana Marie Prikler (1)
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Gregory Provan (10)
School of Computer Science, University College Cork, Ireland

Belarmino Pulido (6)
University of Valladolid, Spain

Marcos Quinones-Grueiro (8, 10, 11, 15)
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Jonas Schleich (3)
Graz University of Technology, Austria

Maximilian Schmidt (7)
Hamburg University of Technology, Germany

Johann Schumann (13)
KBR Inc, NASA Ames Research Center, Moffett Field, CA, USA

Michał Syfert (14)
Warsaw University of Technology, Poland

Anna Sztyber-Betley (2, 14)
Warsaw University of Technology, Poland

Abenezer Taye (8)
Mechanical and Aerospace Engineering Department, School of Engineering and Applied Science, George Washington University, Washington DC, USA

Louise Travé-Massuyès (2, 7, 16)
LAAS-CNRS, University of Toulouse, France

Laure Vieillevigne (16)
Université de Toulouse, Oncopole Claudius Regaud, Institut Universitaire du Cancer de Toulouse (IUCT), France; Université de Toulouse, CNRS, INSERM, Centre de Recherches en Cancérologie de Toulouse (CRCT), France

Günther Waxenegger-Wilfing (14)
Institute of Space Propulsion, German Aerospace Center (DLR), Hardthausen am Kocher, Germany; Institute of Computer Science, University of Würzburg, Germany

Peng Wei (8)
Mechanical and Aerospace Engineering Department, School of Engineering and Applied Science, George Washington University, Washington DC, USA

Franz Wotawa (1, 3, 4, 9, 12)
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Beyond Static Diagnosis: A Temporal ASP Framework for HVAC Fault Detection

Roxane Koitz-Hristov[1]
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Liliana Marie Prikler
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Franz Wotawa
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Abstract

Improving sustainability in the building sector requires more efficient operation of energy-intensive systems such as Heating, Ventilation, and Air Conditioning (HVAC). We present a novel diagnostic framework for HVAC systems that integrates Answer Set Programming (ASP) with Functional Event Calculus (FEC). Our approach exploits the declarative nature of ASP for modeling and incorporates FEC to capture temporal system dynamics.

We demonstrate the feasibility of our approach through a case study on a real-world heating system, where we model key components and system constraints. Our evaluation on nominal and faulty traces shows that exploiting ASP in combination with FEC can identify plausible diagnoses. Moreover, we explore the difference between static and rolling-window strategies and provide insights into runtime versus soundness on those variants. Our work provides a step toward the practical application of ASP-based temporal reasoning in building diagnostics.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics; Computing methodologies → Logic programming and answer set programming

Keywords and phrases Model-based diagnosis, Answer set programming, HVAC, Modeling for diagnosis, Experimental evaluation

Digital Object Identifier 10.4230/OASIcs.DX.2025.1

Funding The work presented in this paper has been supported by the FFG project Artificial Intelligence for Smart Diagnosis in Building Automation (ALFA) under grant FO999914932.

Acknowledgements We would like to express our gratitude to our industrial partner, DiLT Analytics FlexCo.

1 Introduction

It is estimated the building sector accounts for 40% of the energy consumption in the European Union [9] and world-wide [30]. Heating, Ventilation, and Air Conditioning (HVAC) systems in buildings cause a large portion of a building's energy expenditure. Approximately 40% of the final energy consumption of heat and 70% of cooling energy consumption is caused by non-residential buildings [17] and 75% of greenhouse gas emissions are emitted during building operation [10]. Hence, reducing the energy consumption in the building sector is a highly relevant contribution to achieving climate targets.

Fault Detection and Diagnosis (FDD) for HVAC systems in buildings primarily relies on massive human effort with manual data evaluation triggered by user complaints, random checks, or non-specific alarms from the building automation systems [39]. Around 15–30% of a building's energy consumption is due to faults or inefficient operations of HVAC systems [21, 8],

[1] Authors are listed in alphabetical order.

affecting existing and new buildings [19]. Timely fault detection and resolution are essential for preserving the performance and reliability of HVAC systems, ultimately extending their lifespan [45]. However, their inherent complexity, dynamic nature of operation, and the diverse range of potential faults, makes HVAC FDD a challenging task.

Over the past decades, numerous FDD techniques have been developed for HVAC systems in buildings, including *data-driven* and *model-based* methods. *Data-driven approaches* (e.g., machine learning and statistical methods) have become increasingly popular due to their ability to analyze large datasets and detect relevant patterns [8]. However, these techniques require large amounts of data that not only capture nominal but also faulty system behavior, which is often unavailable in real-world HVAC systems. Additionally, many data-driven methods lack interpretability, making it challenging to derive explainable diagnoses [24].

In contrast, *model-based techniques* rely on an explicit system representation to detect deviations from expected behavior. Both the Fault Detection and Isolation (FDI) and Model-Based Diagnosis (MBD) community have contributed to HVAC system diagnosis. FDI methods, rooted in control theory, utilize analytical models based on system equations [46]. In the FDI domain, for instance, Thumati et al. [44] employ a real-time FDI approach for HVAC systems using residual generation and state estimation. Papadopoulos et al. [32] extend model-based FDI techniques to multi-zone HVAC systems, introducing a distributed diagnosis method that enhances scalability and reliability to detect and isolate actuator and sensor faults. However, creating a precise analytical model for HVAC systems is challenging due to their complexity [36]. To overcome these challenges, MBD provides an alternative that does not depend on detailed mathematical formulations but instead represents the system's behavior using logical relationships and constraints [38, 11]. For example, Struss et al. [41] present an approach that exploits numerical Modelica models to generate a qualitative diagnosis model in the domain of Air Handling Units (AHUs). The resulting diagnostic model captures the qualitative deviations of variables from their nominal values. When a discrepancy has been detected between the engineering model and the sensor data, the diagnostic model and observations are supplied to a consistency-based diagnosis engine to identify and isolate faults. Provan [35] propose an approach that involves simplifying detailed HVAC system models to focus on critical components and interactions. The resulting reference model captures the essential system dynamics while minimizing computational complexity. From this reference model, a diagnostic model is generated that retains the necessary behavioral characteristics to effectively isolate faults.

Recently, Answer Set Programming (ASP) [12] has shown promise in the context of model-based reasoning due to its declarative nature and efficient inference. While ASP has been successfully applied in a diagnostic context in various subsets of technical systems [22], digital and analog circuits [47, 48], automated feedback generation for programming assignments [4], and autonomous robots [20], its use in HVAC fault diagnosis remains largely unexplored to the best of our knowledge. In addition, many classical MBD approaches lack explicit temporal reasoning [7], which is essential for tracking dynamic system behavior which is highly relevant in HVAC systems.

In this paper, we introduce an ASP based-diagnosis framework, that integrates Functional Event Calculus (FEC) [23, 29, 26] to model events and fluents (state properties) over time. Our framework thus exploits ASP for constraint reasoning and FEC for modeling temporal dependencies. In addition, we use solver heuristics to compute minimal diagnoses. We further present an initial case study, demonstrating the feasibility of our approach on an HVAC system with limited real-world data.

The remainder of this paper is organized as follows: Section 2 reviews prior work on ASP-based diagnosis, including approaches related to temporal reasoning in MBD and ASP. Subsequently, in Section 3, we provide essential preliminaries on MBD, ASP and FEC. In Section 4, we describe our approach of combining FEC with ASP for diagnosis. Afterward, in Section 5 we present a case study where we applied our approach to a HVAC system and discuss our initial empirical results. Finally, Section 6 concludes the paper and outlines future directions for research.

2 Related work

Various approaches to MBD have been proposed over the last decades, ranging from procedural to declarative paradigms. While most research emphasizes on algorithmic techniques, we focus in this section on declarative techniques, particularly ASP-based methods. In addition, we discuss related approaches to temporal diagnosis.

2.1 ASP-based diagnosis

One early example of declarative MBD is the work by Friedrich and Nejdl [14]. The authors introduce a direct diagnosis approach using hyperresolution in Prolog [5]. Unlike traditional MBD methods that first generate conflicts and then apply a hitting set algorithm to derive diagnoses, their approach obtains candidates directly as a subset of the consistent logical model encoded in Prolog.

Similarly, declarative MBD approaches using ASP define the system model directly in the formalism of the reasoning engine, avoiding an intermediate transformation into a separate logic or constraint representation. Wotawa [47] investigates ASP as an alternative reasoning mechanism for diagnosis of the well-known ISCAS85 circuits. In a consistency-based fashion, the ASP encoding describes the system as a composition of interconnected components, each defined by its nominal behavior. Observations are incorporated in the model as constraints, ensuring that any valid diagnosis must be consistent with the observed behavior. The empirical results indicate that while a traditional dedicated diagnosis algorithm is often faster, ASP is a stable and reliable approach. In an extension of this research, Wotawa and Kaufmann [48] present their direct IDIAG algorithm that iteratively generates all diagnoses with a cardinality of 0 to n based on a theorem prover. Their evaluation revealed that ASP-based diagnosis performance is comparable to specialized algorithms for digital circuits. However, for analog circuits, the required grounding process led to a significant increase in computation time.

Bayerkuhnlein and Wolter [4] propose an alternative ASP encoding for diagnosing faults in programming assignments within an intelligent tutoring system, incorporating intermediate values to enhance fault analysis. In addition, to their "traditional" ASP-based framework, they propose the use of Constraint Answer Set Programming, which enables reasoning over non-groundable domains, to deal with large or infinite domains in their application.

Prikler and Wotawa [34] introduce multi-shot solving, heuristics, and preference-based optimizations in ASP diagnosis and show that heuristic-based methods generally yield faster solutions. By incorporating incremental solving techniques, the authors improve the IDIAG diagnosis algorithm, making it more competitive with specialized diagnosis approaches. Experiments on the ISCAS85 circuits revealed that ASP-based diagnosis can match or outperform hitting-set-based methods when properly optimized.

2.2 Temporal diagnosis

The ability to reason about changing environments is fundamental to many AI applications, particularly in explaining observed phenomena and diagnosing unexpected system behaviors [7]. For instance, Raiman et al. [37] showed that even a single, weak temporal axiom – the non-intermittent-fault assumption – in combination with multiple snapshots allows to disregard explanations that would otherwise be candidates. McIlraith [28] applied Situation Calculus [27] to MBD to address temporal phenomena. By using Situation Calculus as the logical framework for diagnosing faults in dynamic systems, the approach can address fundamental challenges such as the ramification problem. In parallel, Thielscher [42] extended Fluent Calculus [43] to dynamic domains, enabling reasoning about system evolution and diagnosis using a constraint-based representation of healthy system behavior. Building on these foundations, Baral, McIlraith, and Son [3] use an action languages to provide a more compact and computationally efficient representation of diagnosis in dynamic systems. In the domain of robotics, Gspandl et al. [18] present a belief management system that embeds history-based diagnosis into the IndiGolog Situation-Calculus interpreter. Their method keeps a set of possible action histories and updates them after every step. It checks each history against the newest sensor readings, explains any inconsistencies by selecting the most plausible fault hypothesis, and retains only the best-supported histories to guide the robot's next action.

Further advancing the field, Balduccini and Gelfond [2] exploit ASP in combination with the action language $\mathcal{AL}$ to integrate diagnosis into an intelligent agent architecture. In their approach, $\mathcal{AL}$ is used to formalize a high-level description of the system behavior, specifying actions, fluents, and causal laws. A-Prolog, logic programming under answer set semantics, serves as the computational framework for diagnosing faults. Their approach enables intelligent agents to perform diagnosis, testing, and repair within a unified framework of A-Prolog.

Temporal logics are another formalism that has been used for diagnostic purposes, e.g., for behavioral diagnosis of Linear-Temporal Logic (LTL) specifications [33]. More recently, Feldman et al. [13] proposed Finite-Trace Next Logic (FTNL) and a SAT-based algorithm to diagnose synchronous sequential circuits. FTNL is a simpler temporal logic as it considers only finite traces and restricts temporal expressiveness to a single *next* operator. The FTNL formula is unrolled for a finite horizon before passing it to a SAT solver with a novel encoding that reduces the number of variables and clauses and has been shown to be efficient on the ISCAS89 circuits.

Our work is a natural extension of previous research as it combines declarative diagnostic modeling via ASP with FEC to capture behavior over time. Thus, we enable temporal reasoning that accounts for the progression of system behavior with the goal of efficient and effective FDD in the building heating domain.

3 Background

To describe our ASP-based approach, we start by introducing the necessary preliminaries. We begin with MBD, recalling the classical consistency-based definition of diagnosis, before explaining ASP and FEC, with some slight adaptations of the formalism.

3.1 Model-based diagnosis

In MBD [11, 38], we describe a system as a set of components, whose nominal behaviour can be formulated using some system of logics, e.g., first-order logic. A diagnosis problem occurs once we have gathered observations.

▶ **Definition 1** (Diagnosis problem). *A diagnosis problem is a tuple* $(SD, COMP, Obs)$*, where* SD *is the (logic) description of some system,* $COMP$ *the set of components of said system, and* Obs *a set of observations gathered from the system.*

The diagnosis task, given a system description, components and observations as above, is to determine which components, if any, are faulty. The approach taken by MBD is based on consistency between the system description and observations: if the observations match the nominal behaviour of the system, no component needs to be marked as faulty. However, the system description may also formulate constraints based on the assumption that some $c \in \mathsf{COMP}$ is healthy, typically written $\neg\mathsf{ab}(c)$. If one assumes to the contrary $\mathsf{ab}(c)$, said constraints are no longer enforced, allowing consistency once again if they were not to hold otherwise.

▶ **Definition 2** (Diagnosis). *Let* $(SD, COMP, Obs)$ *be a diagnosis problem. A set* $\Delta \subset COMP$ *is a diagnosis if and only if* $SD \cup Obs \cup \{ab(c) | c \in \Delta\} \cup \{\neg ab(c) | c \in COMP \setminus \Delta\}$ *is consistent.*

A diagnosis problem is ill-defined if $\Delta = \mathsf{COMP}$ is not a diagnosis. Loosely speaking, when all components of a system are found to be abnormal, there is no nominal behaviour to expect. On the other extreme, we expect $\Delta = \varnothing$ to be a diagnosis when the system behaves as intended.

More generally, we are interested in diagnoses that are in some sense *minimal*, that is diagnoses that do not mark components as faulty without reason.

▶ **Definition 3** (Minimality of a diagnosis). *Let* Δ *be a diagnosis.* Δ *is*

- *subset-minimal, or parsimonious, if there exists no diagnosis* Δ' *s.t.* $\Delta' \subset \Delta$*, and*
- *cardinality-minimal, if there exists no diagnosis* Δ' *s.t.* $|\Delta'| < |\Delta|$*.*

3.2 Answer set programming

ASP [6, 25] is a logic programming paradigm that allows for non-monotonic reasoning. An answer set program consists of rules of the shape

$$a \leftarrow b_1 \wedge \cdots \wedge b_m \wedge \textbf{not } b_{m+1} \wedge \cdots \wedge \textbf{not } b_n,$$

where a and b_i are atoms and **not** denotes *default* negation – that is **not** x is assumed to hold unless x is known to hold. The head of a rule r is $h(r) = \{a\}$ and the body consists of positive atoms $b^+(r) = \{b_1, \ldots, b_m\}$ and negative atoms $b^-(r) = \{b_{m+1}, \ldots, b_n\}$. Intuitively $h(r)$ follows if all atoms in $b^+(r)$ holds and no atom in $b^-(r)$ holds. A rule is called a fact, if $m = n = 0$.

We call an answer set program *ground* if no variable occurs in any of its rules. The Gelfond-Lifschitz transformation [16] of a ground program P with respect to a set of atoms (or model) τ is

$$P^\tau = \{h(r) \leftarrow b^+(r) | r \in P, b^-(r) \cap \tau = \varnothing\}.$$

By construction P^τ is a Horn formula and thus admits one unique minimal model. If this model is τ, then τ is a stable model.

▶ **Definition 4** (Stable model). *Let P be a grounded answer set program and τ a set of atoms. Let P^τ be the Gelfond-Lifschitz transformation of P w.r.t. τ. τ is a stable model (or answer set) of P if and only if τ is the smallest satisfying model of P^τ.*

The rest of this section deals with non-ground programs, particularly Clingo extensions that we make use of. Non-ground answer set programs keep the basic shape of answer set programs, but use a wider set of literals than pure atoms and their default negations.

The first natural extension to ASP is the use of (implicitly quantified) variables. Consider for example the program

$$\mathsf{flies}(B) \leftarrow \mathsf{bird}(B) \wedge \mathbf{not}\ \mathsf{flightless}(B)$$
$$\mathsf{bird}(B) \leftarrow \mathsf{penguin}(B)$$
$$\mathsf{flightless}(B) \leftarrow \mathsf{penguin}(B)$$
[... and so on for other bird species ...]

which is one way of formulating the "Tweety problem". Given bird(tweety), we want to derive flies(tweety). Given penguin(tweety), we want to derive bird(tweety), but not flies(tweety), and so on.

To solve answer set programs with variables, we need to ensure that all replacements of variables with concrete (that is variable-free) terms are properly accounted for. To this end, we construct the Herbrand base of the program and then ground the program with respect to it.

▶ **Definition 5** (Herbrand universe and base). *Let P be a logic program. The Herbrand universe of P, denoted $\mathcal{U}(P)$ is the set of all terms which can be formed from constants and function symbols in P.*

The Herbrand base $\mathcal{B}(P)$ is the set of all variable-free atoms, which can be constructed from predicates in P and terms in $\mathcal{U}(P)$.

▶ **Definition 6** (Grounding). *Let P be an answer set program, and $r \in P$. The grounding of r, written $\mathcal{G}(r)$ is a set of ground rules, s.t. $r \Leftrightarrow \mathcal{G}(r)$ w.r.t $\mathcal{B}(P)$. Furthermore, $\mathcal{G}(P) = \bigcup_{r \in P} \mathcal{G}(r)$.*

Our definition of grounding is notably looser than the ones we find in literature, for reasons that will become apparent once we discuss extensions other than variables. However, even when dealing with variables, the traditional approach of replacing a variable with every possible term in $\mathcal{U}(P)$ one at a time is a wasteful over-approximation that modern solvers tend to avoid [31].

In some places, where a domain is explicitly known, one can also use a *pool* as a notational shorthand rather than a variable. Pools are collections of terms separated by semicolons, i.e., written as $t_1; \ldots; t_n$ with t_i terms for $1 \leq i \leq n$. Similar to variables, rules are instantiated with each possible replacement of the pool with one of its terms. As an illustrative example, the pooled "fact"

bird(tweety; roy; silo).

is a shorthand for the facts

bird(tweety).
bird(roy).
bird(silo).

Similar to variables, we might want to perform calculations on numbers, both concrete numbers and again variables. Let $\prec \in \{<, \leq, =, \neq, \geq, >\}$ and t_1, t_2 terms, then $t_1 \prec t_2$ is an arithmetic literal that the solver must evaluate numerically after having replaced all variables with concrete numbers.

We continue with conditional literals. Written $b : c_1, \ldots, c_n$, conditional literals are grounded by replacing the symbolic or arithmetic literal b with all replacements $b^{x=t}$, where $c_1^{x=t}, \ldots, c_n^{x=t}$ hold, where $l^{x=t}$ denotes the replacement of variables x with concrete terms t similar to how variables are handled in grounding.

Aggregate literals are written as $l\ \{b_1; \ldots; b_n\}\ u$, where l, u are integers and $b_1, \ldots, b_n$ are literals[2], enforce a lower and upper bound respectively on the number of $b_1, \ldots, b_n$ that may hold. Both l and u may be omitted; in this case no bound is enforced. As a special case, $\{a\}$ for a single literal a is known as a *choice expression* and states that a may or may not follow as the result of applying some rule.

For a more complete description of the syntax and semantics of Clingo, we refer the interested reader to [15].

3.3 Functional event calculus

The event calculus (EC) [23, 29] is a logical framework for representing and reasoning about actions (events) and their effects. FEC [26] extends this framework from the boolean domain to arbitrary domains.

The FEC has a sort $\mathcal{F}$ for fluents (variables $f, f', f_1, f_2, \ldots$), a sort $\mathcal{A}$ for actions (variables $a, a', a_1, a_2, \ldots$), a sort $\mathcal{V}$ for values (variables $v, v', v_1, v_2, \ldots$), and a sort $\mathcal{T}$ for discrete points in time (variables $t, t', t_1, t_2, \ldots$). For the scope of this paper, it suffices that $\mathcal{T} \subset \mathbb{N}$ and hence time is totally ordered under the comparison $\leq$. Key predicates are $\mathsf{happens} \subset \mathcal{A} \times \mathcal{T}$ and $\mathsf{causes\_value} \subset \mathcal{A} \times \mathcal{F} \times \mathcal{V} \times \mathcal{T}$. The function $\mathsf{domain} : \mathcal{F} \to 2^{|\mathcal{V}|}$ assigns a domain to each fluent, while $\mathsf{value\_of} : \mathcal{F} \times \mathcal{T} \to \mathcal{V}$ assigns a value to each fluent at each time step.

To describe the relationships between these predicates and functions, we first define the auxiliary predicates $\mathsf{value\_caused} \subset \mathcal{F} \times \mathcal{V} \times \mathcal{T}$, $\mathsf{value\_caused\_between} \subset \mathcal{F} \times \mathcal{V} \times \mathcal{T} \times \mathcal{T}$ and $\mathsf{other\_value\_caused\_between} \subset \mathcal{F} \times \mathcal{V} \times \mathcal{T} \times \mathcal{T}$.

$$\mathsf{value\_caused}(f, v, t) \overset{\text{def}}{\equiv} \exists a[\mathsf{happens}(a, t) \wedge \mathsf{causes\_value}(a, f, v, t).] \tag{FEC1}$$

$$\mathsf{value\_caused\_between}(f, v, t_1, t_2) \overset{\text{def}}{\equiv} \exists t[\mathsf{value\_caused}(f, v, t) \wedge t_1 \leq t \wedge t < t_2] \tag{FEC2a}$$

$$\mathsf{other\_value\_caused\_between}(f, v, t_1, t_2) \overset{\text{def}}{\equiv} \exists t, v'[\mathsf{value\_caused\_between}(f, v', t_1, t_2) \wedge v \neq v'] \tag{FEC2b}$$

As can be seen from their definitions, value_caused posits that some action happens at the specified time, which gives cause for some fluent to take a certain value, whereas value_caused_between posits that such an action takes place within the interval $[t_1, t_2)$. Finally, other_value_caused_between posits that an action within the interval gives cause to some other value. In all of these, *gives cause* needs to be interpreted in a non-deterministic manner: a non-deterministic action, such as a dice roll, gives cause for any side of the dice to show up, but only one side will show up. Likewise, if two people try to respectively open and close a door, the door will not be open and close at the same time[3].

[2] We only deal with a particular type of head aggregates here. Clingo also supports other aggregates, that are not needed within the scope of this paper.

[3] It may however be "half open" if such a state is modeled.

With these auxiliary definitions, we can define notions of cause, effect and inertia. Axiom (FEC3) deals with inertia: a fluent which takes on a certain value at some time and has no action changing that value, will still take on that value in the future. Axiom (FEC4) deals with cause and effect in a roundabout way: a fluent can not take on a value without cause if another value has been caused in some interval.[4] Calling back to our earlier examples of non-deterministic outcomes, the consequent discards any value that has been given cause within the interval, leaving only values that were given cause to be potential effects.

$$\mathsf{value\_of}(f, t_2) = v \leftarrow [\mathsf{value\_of}(f, t_1) \wedge t_1 < t_2 \wedge \neg \mathsf{other\_value\_caused\_between}(f, v, t_1, t_2)] \tag{FEC3}$$

$$\mathsf{value\_of}(f, t_2) \neq v \leftarrow [\mathsf{other\_value\_caused\_between}(f, v, t_1, t_2) \wedge \neg \mathsf{value\_caused\_between}(f, v, t_1, t_2)] \tag{FEC4}$$

Finally, we constrain fluents to only take values from their assigned domains.

$$\neg(\mathsf{value\_of}(f, t) = v \wedge v \notin \mathsf{domain}(f)) \tag{FEC5}$$

The axioms (FEC1)–(FEC5) define the functional event calculus.

4 ASP diagnosis with functional event calculus

In this section, we describe our model-based diagnosis system built on top of the functional event calculus. We first discuss how we describe the health state as fluents. Next, we describe how we can model individual components and the connections between them. Finally, we extend the functional event calculus with immediate changes.

4.1 Fluent health states

We assume, that the health state of a component may vary over time and thus model it as a fluent, i.e., $\mathcal{F} \supset \{\mathsf{health}(c) | c \in \mathsf{COMP}\}$. The health state of a component is composed of the good state ok and at least one abnormal state. The domain of this health state can be modeled as $\mathsf{domain}(\mathsf{health}(c)) = \{\mathsf{ok}, \mathsf{ab}\}$ in a weak fault model, whereas in a strong fault model we have $\mathsf{domain}(\mathsf{health}(c)) = \{\mathsf{ok}, \mathsf{ab}_1, \ldots \mathsf{ab}_n\}$ for some n.

In order to allow for the health state of a component to switch from ok to ab (or ab_i for some i), we allow unknown actions to occur. Likewise, to model repair, we add a dedicated repair action for each component. Formally, we have $\mathcal{A} \supset \{\mathsf{unknown}\} \cup \{\mathsf{repair}(c) | c \in \mathsf{COMP}\}$. The singular $\mathsf{unknown}$ action acts as a stand-in for any number of actions that are all unknown.

The effects of unknown and repair actions are captured by $\mathsf{causes\_value} \supset U \cup R$ with $U \subset \{(\mathsf{unknown}, \mathsf{health}(c), v, t) | c \in \mathsf{COMP}, v \in \mathsf{domain}(c) \setminus \{\mathsf{ok}\}, t \in \mathcal{T}\}$ modeling the unknown action and $R = \{(\mathsf{repair}(c), \mathsf{health}(c), \mathsf{ok}, t) | c \in \mathsf{COMP}, t \in \mathcal{T}\}$ modeling repair. Semantically, the repair actions always give reason for a component to become healthy (but they do not always happen), whereas unknown actions are free to give cause for some, but not all health states to change.

[4] The original formalization [26] uses a slightly different variant of (FEC3) that also takes into account values caused at t_1. This is redundant as per (FEC4). Note that our (FEC4) is functionally identically to the original formalization, but makes use of the additional auxiliary axiom (FEC2a).

In ASP, we implement unknown actions using Listing 1. This implementation adds two further constraints: first $\mathsf{happens}(\mathsf{unknown}, t) \leftarrow \exists f, v[\mathsf{causes\_value}(\mathsf{unknown}, f, v, t)]$ ensures that the unknown action happens (and thus takes effect) when it is supposed to cause a value. Second, the unknown action only causes a healthy state to become abnormal, i.e., $\mathsf{causes\_value}(\mathsf{unknown}, f, v, t) \rightarrow \mathsf{value\_of}(f, t) = \mathsf{ok}$. By not allowing the unknown action to transition between faulty state, we prevent the health state from alternating between to values that have (near-)identical symptoms attached to them.

Listing 1 ASP implementation of the unknown action.

```
action(unknown).
% Allow solver to guess affected health states.
{ causes_value(unknown, health(C), ab, T) } :-
  time(T), comp(C), value_of((health(C), T), ok).
{ causes_value(unknown, health(C), ab(X), T) } :-
  time(T), comp(C), value_of((health(C), T), ok), fault(C, X).
happens(unknown, T) :- time(T), causes_value(unknown, _, _, T).
```

4.2 Modeling component behavior

We imagine idealized components to have three kinds of fluents attached to them: *inputs*, *outputs* and *internal controls*. Most components will have at least one input and an output, as well as a nominal relation between them.

In FEC, we need to represent this nominal relation using actions. For example, an analog adder has two numeric inputs i_1, i_2, one numeric output o and the nominal relation $o = i_1 + i_2$. This nominal relation is upheld by the adder *adding* its inputs; hence adding is an action (performed by the adder).

In ASP, we may describe adders as a generalized component as in Listing 2. The first line communicates that numbers are defined outside of the current scope – typically in the place where an adder is used. The second line states that adders are a component. A fluent for its health state is automatically created by code that implements Section 4.1. The third and fourth line define the fluents related to an adder and their domains respectively. Finally, the rule for inferring causes_value establishes the semantics of adding.

Listing 2 FEC-based description of analog adders.

```
#defined number/1. % elsewhere
comp(C) :- type(C, adder).
fluent(in1(C);in2(C);out(C)) :- type(C, adder).
domain((in1(C);in2(C);out(C)), N) :- type(C, adder), number(N).

causes_value(adding(C), out(C), N+M, T) :-
  type(C, adder),
  value_of((health(C), T), ok),
  number(N+M), % otherwise addition is undefined
  value_of((in1(C), T), N),
  value_of((in2(C), T), M).
```

In addition to nominal behavior, abnormal behavior must also be described via actions. While all behavior is at the end implemented using mere logic rules, it makes sense to conceptually distinguish between

- abnormal behavior that causes no change in any fluent,
- abnormal behavior that modifies existing actions, and
- abnormal behavior that causes new actions to occur.

The first kind of abnormal behavior is already implemented without any additional rules, as axioms (FEC3) and (FEC4) uphold that nothing happens without an action.

For abnormal behaviour that modifies existing actions, alternative rules with causes_value in their head need to be written. Just like the rules for the nominal behavior, these depend on the value of the health state. A typical assumption in a weak fault model would be "anything goes", i.e., the action may give cause for affected fluents (when in doubt: all output fluents) to take any value from its domain. A strong fault model could instead assume that the outcome differs only slightly from the one that is expected.

For abnormal behavior that causes new actions to occur, we also need to write rules with happens in their head, which again depend on the health state of a component.

As an alternative to the above, a generic rule could encode all possible behaviour, with constraints restricting the possible actions and caused values in the nominal state. An example of this is shown in Listing 3.

Listing 3 Alternative description of analog adders using constraints.

```
#defined number/1. % elsewhere
comp(C) :- type(C, adder).
fluent(in1(C);in2(C);out(C)) :- type(C, adder).
domain((in1(C);in2(C);out(C)), N) :- type(C, adder), number(N).

{ causes_value(adding(C), out(C), N, T) : number(N) } 1 :-
  type(C, adder), time(T).
happens(adding(C), T) :- causes_value(adding(C), _, _, T).

:- value_of((in1(C), T), N),
   value_of((in2(C), T), M),
   value_of((out(C), T), V), V != N+M,
   value_of((health(C), T), ok),
   not causes_value(adding(C), out(C), _, T).

:- causes_value(adding(C), out(C), V, T),
   value_of((health(C), T), ok),
   value_of((in1(C), T), N),
   value_of((in2(C), T), M),
   V != N+M.
```

Having modeled individual components, it is now time to model the connections between them. As in traditional model-based diagnosis, we assume that the values at either end of a connection is always the same as the other – if this is not the case, we can always add components that introduce the necessary delays.

Similar to the approach used by Wotawa and Kaufmann [48], we can establish connections with an auxiliary predicate bound $\subset \mathcal{F} \times \mathcal{F}$ and the rule

$$\mathsf{value\_of}(f_2, t) = v \leftarrow \mathsf{bound}(f_1, f_2) \wedge \mathsf{value\_of}(f_1, t) = v.$$

To make the relation symmetrical, one can write $\mathsf{bound}(f_1, f_2) \wedge \mathsf{bound}(f_2, f_1)$. Note that this rule does not yet propagate values caused from one fluent to another. To this end, the rule

$$\mathsf{causes\_value}(a, f_2, v, t) = v \leftarrow \mathsf{bound}(f_1, f_2) \wedge \mathsf{value\_of}(a, f_1, v, t) = v$$

can be used.

Apart from fluents that are unconditionally bound, our system also supports conditionally bound fluents. We write these as

$$\mathsf{value\_of}(f_2, t) = v \leftarrow \mathsf{bound}_c(f_1, f_2) \wedge \mathsf{value\_of}(\mathsf{health}(c), t) = \mathsf{ok} \wedge \mathsf{value\_of}(f_1, t) = v,$$

where f_2 typically refers to a fluent that may have its value changed by an action of the component c.

Alternatively, one could model connections by using the same fluent as the input/output of the affected components, i.e., using shared fluents. This incurs a level of indirection, as a mapping of fluents to component inputs/outputs needs to be established and consistently used.

4.3 Split seconds and immediate changes

The axioms (FEC3) and (FEC4) make it so that one unit of time passes between cause and effect of a change and thus assume a passage of time between them. Modeling time this closely requires sampling at the speed of change, which is wasteful, as often small changes accumulate over time to make large changes. Instead, we may wish to establish effects that become visible "immediately", that is in the same time step as they occur.

To this end, we introduce the predicates $\mathsf{causes\_value\_immediately} \subset \mathsf{causes\_value}$ and $\mathsf{value\_caused\_immediately} \subset \mathsf{value\_caused}$. The semantics of these "immediate" predicates is similar to their non-immediate counterparts, except that they cause a value change to be observed in the same time step rather than the next.

$$\mathsf{value\_caused\_immediately}(f, v, t) \overset{\text{def}}{\equiv} \exists a[\mathsf{happens}(a, t) \wedge \mathsf{causes\_value\_immediately}(a, f, v, t)]. \quad \text{(FECi1)}$$

To make axioms (FEC3) and (FEC4) respect immediate changes, we replace (FEC2a) with (FECi2). Intuitively, this version of $\mathsf{value\_caused\_between}$ allows "between" to mean at the end of the interval, or $t = t_2$, when a value is caused immediately, while preserving $t_1 \neq t_2$.

$$\mathsf{value\_caused\_between}(f, v, t_1, t_2) \overset{\text{def}}{\equiv} \exists t[\mathsf{value\_caused}(f, v, t) \wedge t_1 \leq t \wedge t < t_2] \vee [\mathsf{value\_caused\_immediately}(f, v, t_2) \wedge t_1 < t_2]. \quad \text{(FECi2)}$$

Lastly, a value caused immediately must have its change actually happen in the same time step. This is enforced by (FECi3). Without this axiom, two "immediate" changes happening at times t and $t + 1$ would trigger non-determinism in (FEC4), allowing either value to be taken.

$$\neg(\mathsf{value\_caused\_immediately}(f, v, t) \wedge \mathsf{value\_of}(f, t) \neq v). \quad \text{(FECi3)}$$

This set of axioms allows us to model sensors that are polled at regular intervals as well as components that merely propagate changes (e.g., idealized non-sequential components of sequential circuits).

5 A case study: Heat distribution line

In this section, we first describe the HVAC system, which is the base of our case study. The heat distribution line that we consider for our modeling is part of the heating system in a commercial building. The system includes a pressurized distributor linked to a district heating transfer station, which supplies heat to multiple distribution lines serving different zones in the building. Our case study focuses on one of these heat distribution lines.

Afterward, we report on how we modeled the system, as we simplify the original schematics while ensuring that our models maintains the essential behavioral aspects of the heat distribution line. Lastly, we present some initial empirical results on said use case.

5.1 System description

The heat distribution line is hydraulically designed as an injection circuit with a globe valve. Figure 1 depicts its schematics. When the heating system is switched on, the central flow pipe transports hot water to the flow pipe (FP) of the heat distribution line. The central return pipe is connected to the return pipe (RP) of the heat distribution line and transports the cooled water back to the transfer station. The heat output in the zone is regulated by controlling the temperature of the water in the flow pipe. This control is based on the outdoor temperature and is adjusted through a motorized valve. The room temperature is regulated directly at the radiators within the zone.

The relationship between the outdoor temperature and the room temperature is represented by the heating curve, which determines the setpoint for the flow pipe temperature. The lower the outdoor temperature, the higher the supply temperature must be to ensure the desired room temperature.

The heat distribution line consists of the following components:

- **Heat Exchanger** (H): H transfers heat from the hot water in the flow pipe to the indoor air within the zone.
- **Temperature Sensors** (T_1, T_2): T_1 measures the temperature of the water in the flow pipe and serves as a reference for control, while T_2 monitors the return water temperature.
- **Pumps** (P_1, P_2): The pumps circulate water within the heating circuit and maintain constant pressure. P_1 and P_2 operate alternately or remain simultaneously switched off. As soon as one pump is running, there is a flow.
- **Valve** (V): The motorized globe valve regulates the mixing ratio of flow and return water by modifying its position. It serves as the control element, adjusting the proportion of hot flow water entering the heating zone based on its position (0–100%). At 0%, no hot heating water is injected, meaning the flow temperature equals the return temperature. At 100%, the maximum amount is injected, and the flow temperature corresponds to the district heating supply temperature.
- **Check valve** (CV): CV prevents direct bypassing of hot water from the flow pipe to the return pipe, ensuring proper circulation within the heating system.
- **Outdoor Temperature Sensor**: Although not depicted in Figure 1, this sensor provides an essential input for the control system.

Additionally, we have preconfigured value ranges to verify the plausibility of the measured values, along with supplemental constraints that further define the system:

- The measured flow temperature must always be higher than the measured return temperature.

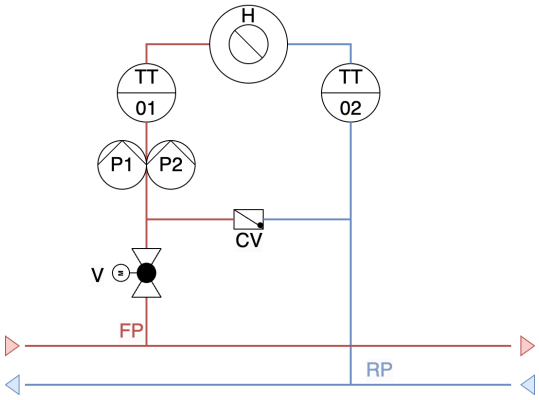


Figure 1 Heat distribution line schematics.

- The measured flow temperature may be up to a maximum of 2 K lower than the setpoint temperature.
- The measured flow temperature may exceed the setpoint temperature by a maximum of 1 K.

5.2 Heat distribution line model

We simplify our model as outlined in Figure 2:

- **Mixing valve** (MV): The check valve CV and motorized globe valve V from Figure 1 are simplified into a singular mixing valve MV.
- **Mixing valve** (IV): A second mixing valve IV is added. This mixing valve is controlled by the answer set program to always use the active pump.
- **Temperature Sensors** (T_1, T_2): The temperature sensors T_1 and T_2 are "loosely" attached to the system – they only measure the temperature, but do not influence it.
- **Main Pump** (MP): A "main pump" MP abstracts the link to the district heating transfer station.

Each component type has a set of fluents associated with its instances. For example, pumps have the fluents $\{⏻, F^{\leftarrow}, F^{\rightarrow}, T^{\leftarrow}, T^{\rightarrow}\}$ indicating their power status ("on" or "of"), inflow, outflow, inflow temperature and outflow temperature respectively. In ASP, we model these fluents and their domains as in Listing 4.

Listing 4 Fluents relating to pumps and their domains.

```
fluent(powered(C);flow((in;out), C);temp((in;out), C)) :-
  type(C, pump).
domain(powered(C), (on;off)) :- type(C, pump).
domain(flow(DIR, C), (flow;no_flow)) :- type(C, pump), DIR=(in;out).
domain(temp(DIR, C), T) :- type(C, pump), DIR=(in;out), temp(T).
```

We model pumps so that they allow for temperature to change in the direction of their input – that is, if the input is warmer than the current output, the next output is allowed to be warmer, and likewise for colder inputs. To model this behavior, we introduce two actions – `heating` and `cooling` respectively. The heating action is shown in Listing 5.

Mixing valves can either immediately forward temperatures and flows from one of their inputs or mix them, allowing for any temperature between the warmer and the colder side to be achieved at their output. Finally, the heat exchanger allows for any output strictly below its input.

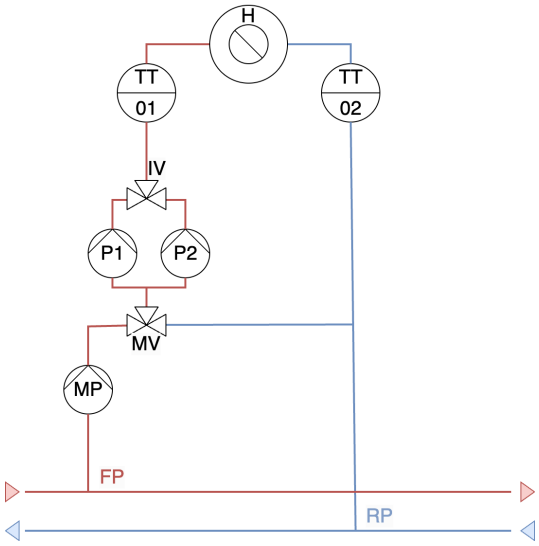


Figure 2 Model schematics.

Listing 5 The `heating` action in ASP.

```
happens(heating(C), T) :-
  type(C, pump),
  causes_value(heating(C), _, _, T).
{ causes_value(heating(C), temp(out, C), X, T) } :-
  time(T), time(T+1), temp(X),
  value_of((temp(in, C), T), W), value_of((temp(out, C), T), V),
  V < X, X <= W,
  value_of((flow(in, C), T), flow),
  value_of((flow(out, C), T), flow).
```

The preconfigured value ranges form the domains of fluents within our model. In particular, we discretize them, so that any integer temperature value within the known range is a temperature, power and flow are binary ("on" or "off" and "flowing" or "not flowing" respectively), and the mixing valves have positions `1`, `2`, and `mix`. The constraints regarding the flow temperature are implemented by additional rules for components and groups of components. The adjustment of setpoints and valve positions is unchecked, as is the condition that both pumps must operate alternately.

5.3 Results

We assess our model on three real-world traces, one containing data from operation under nominal conditions, and two where the valve was leaking and stuck respectively. We prepare those traces so that initial values for all fluents of interest – particularly, the states of the pumps and valve as well as all measured temperatures – are available, but encode later changes of the state only via actions. This means that in order to arrive at the correct diagnoses, the solver can not simply rely on the input providing the state directly, but has to actually reason about the actions and retain information.[5]

[5] It should be noted that in continuous monitoring scenarios, residual-based and data-driven approaches also require updates of internal variables when they start diverging.

Table 1 Diagnosis results. Diagnoses corresponding to the ground truth are highlighted in bold.

dataset	n	diagnoses
nominal	8	**∅**
leaking	8	$\{\mathbf{MV}\}, \{T_1, T_2\}, \{P_1, T_2\}$
stuck	8	$\varnothing$
nominal	16	**∅**
leaking	16	$\{\mathbf{MV}\}, \{T_1, T_2\}, \{P_1, T_2\}$
stuck	16	$\{\mathbf{MV}\}, \{MP\}, \{P_1\}, \{T_1\}$
nominal*)	64	$\{H, T_1\}$,†) $\{H, T_2\}$,†) $\{H, MV\}, \{T_1\}, \{T_2\}$
leaking*)	64	$\{\mathbf{MV}\}, \{T_1, T_2\}$
stuck*)	64	$\{\mathbf{MV}\}, \{T_1, T_2\}, \{P_1, T_2\}$

*) using a rolling window; preserving fluents related to power status

†) non-minimal in final step

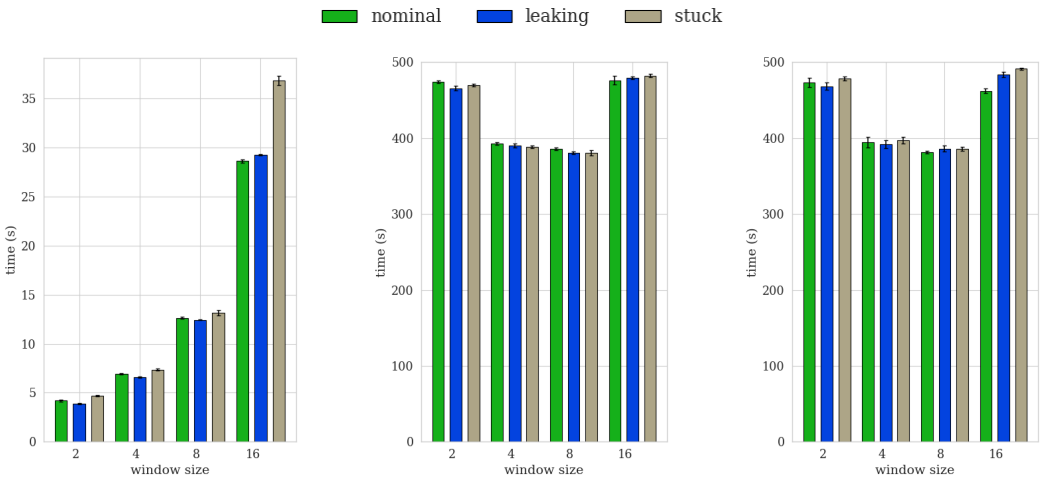
We consider two scenarios for our evaluation: first, we apply a naïve approach using a single solver call to diagnose n time steps at once. As we will see, this approach is quite effective for small n, but does not scale well as the number of time steps and thus the solution space increases. Next, we integrate multiple solver calls over a rolling window to achieve more efficient diagnosis over a larger number of time steps. In both scenarios, we use a domain-specific heuristic to ensure trace-minimal diagnoses. A diagnosis Δ is trace-minimal if $\nexists \Delta' \forall t : \Delta'_t \subseteq \Delta_t$, that is there is no other diagnosis that reports a subset of components as broken in each time step t. A stricter heuristic would consider subset-minimal diagnoses in the final time step.

Table 1 shows the diagnoses eventually yielded after 8, 16, and 64 time steps. For the nominal trace, there is a false positive at a certain time step, where in our discretization T_1 and T_2 report the same temperature despite T_1 still being larger. This issue could be solved by increasing the precision, e.g., by premultiplying with 10, 100, or 1000, which however also leads to larger domains for the fluents, and thus a significantly larger solution space.

For the faulty traces, the correct diagnosis is always among the diagnoses in the leaking case, but curiously absent in the stuck case when only a smaller part of the input is considered. In fact, with just one more time step, that is $n = 9$, we get the same diagnoses as for $n = 16$. More generally, without repair actions, which are absent from our inputs, diagnoses only grow larger. For most traces, this means that eventually a fixed point will be reached, where no more information is gained by considering future time steps – especially if $\Delta = COMP$. Practical applications should however consider repair sooner.

We also see an interesting behaviour with $n = 64$ by choice of our heuristic, as we compute the minimal diagnoses w.r.t. *all* time steps, but only report the last. In this data set, we encounter multiple steps where components supposedly break. The first yields the diagnoses $\{\{H\}, \{T_1\}, \{T_2\}\}$ – the second component, i.e., T_1, T_2 or MV, only breaks later and thus there are different explanations. If instead one were to compute the minimal diagnosis for the *last* time step only, these gratuitous diagnoses would vanish.

In terms of performance, we find that a naïve approach of using ASP to diagnose n time steps does not scale for increasingly large n. The performance limitations can be seen in Figure 3: while the total diagnosis time for 8 time steps is just three times that of 2 time steps – and thus scales sublinearly up to this point – diagnosis times for 16 time steps are



(a) Single window.

(b) 256 time steps.

(c) 256 time steps with preserved fluents.

Figure 3 Diagnosis times.

Table 2 Inconsistent results between different configurations in the rolling window approach.

dataset	params	diagnoses
stuck	$w = 8, n = 16, K = \varnothing$	$\varnothing$
stuck	$w = 16, n = 16, K = \varnothing$	$\{\mathbf{MV}\}, \{MP\}, \{P_1\}, \{T_1\}$
stuck	$w = 8, n = 16, K = \{⏻_{P_1}, ⏻_{P_2}, ⏻_{MP}\}$	$\{\mathbf{MV}\}, \{MP\}, \{P_1\}, \{T_1\}$
stuck	$w = 8, n = 256, K = \varnothing$	$\{\mathbf{MV}\}, \{T_1\}$
stuck	$w = 8, n = 256, K = \{⏻_{P_1}, ⏻_{P_2}, ⏻_{MP}\}$	$\{\mathbf{MV}\}, \{T_1, T_2\}, \{P_1, T_2\}$

more than twice as large. For even larger window sizes, e.g., $w = 32$ or $w = 64$, performance is worse still, if such window sizes are even feasible – in our experiments, we found $w = 64$ to be infeasible and thus used a rolling window for Table 1. A window size of $4 \leq w \leq 8$ appears optimal from our observations. The overhead of preserving fluents seems negligible.

The rolling window, however, introduces soundness problems to the diagnosis, as information is lost between subsequent invocations of the solver. It is both possible to be too strict and not strict enough about the fluent values to remember. Table 2 shows a few examples of different configurations – varying the window size w, the number n of time steps considered, and the set K of fluents, whose value is carried over to the next window (not including the `ab` health states, i.e., the previous diagnoses, which are always kept) – leading to different results.

It is well possible, that in the general case, this inconsistency can – or must be – resolved by means of another answer set program and solver call. Particularly, cautious consequences (cf., e.g., [1]) would indicate which fluents must take a certain value in any answer set that assumes a particular diagnosis, but may not be enough to also model which fluents may not ever take a certain value while assuming a particular diagnosis. We leave this gap to future work.

6 Conclusion and Future Work

We demonstrated a diagnosis system based on ASP and FEC and showed how this system can be used to model the temporal behavior of an HVAC system. Our approach enables an explicit modeling of time-dependent phenomena in ASP in the context of MBD. Interestingly, our approach is not only suitable for FDD but also for retrospective analysis. By evaluating a time series after the fact, the system can help identify precisely when a fault began to manifest, offering valuable insights into system dynamics and behavior over time.

We conducted an initial case study on a heating system that demonstrated that the ASP diagnosis model can be used for a practical diagnosis task. While the experiments indicate that a static window approach becomes computationally infeasible very quickly with increasing window sizes, the rolling window variant offers a better trade-off between runtime and window size. However, the rolling window approach comes at the cost of soundness due to information loss between iterations. We showed that maintaining select fluents across solver runs can mitigate this issue to some extent.

In future work, we plan on improving the incremental reasoning and formalizing a more robust preservation strategy for fluents between runs in the case of the rolling window approach. One possibility would be to consider cautious reasoning to infer missing information. Furthermore, we intend to extend our work by considering more case studies from the building domain and other potential application areas, such as diagnosis in the automotive domain.

Another practical challenge worth investigating lies in managing the number of diagnoses returned. Even when only subset minimal diagnoses are considered, we see a rather large number of possible results in the small system studied. Thus, finding strategies for reducing and prioritizing diagnoses is essential in a practical context. For instance, Stern et al. [40] highlight that presenting the full set of diagnoses may be less effective than aggregating them into a health state, which is a compact representation that estimates the likelihood of each component being faulty. It remains to be seen, whether their approach can be meaningfully integrated with ours.

References

1 Mario Alviano, Carmine Dodaro, Salvatore Fiorentino, Alessandro Previti, and Francesco Ricca. ASP and subset minimality: Enumeration, cautious reasoning and MUSes. *Artificial Intelligence*, 320, 2023. doi:10.1016/j.artint.2023.103931.

2 Marcello Balduccini and Michael Gelfond. Diagnostic reasoning with A-prolog. *Theory and Practice of Logic Programming*, 3(4-5):425–461, 2003. doi:10.1017/S1471068403001807.

3 Chitta Baral, Sheila McIlraith, and Tran Cao Son. Formulating diagnostic problem solving using an action language with narratives and sensing. In *KR*, pages 311–322. Citeseer, 2000.

4 Moritz Bayerkuhnlein and Diedrich Wolter. Model-based diagnosis with ASP for non-groundable domains. In *International Symposium on Foundations of Information and Knowledge Systems*, pages 363–380. Springer, 2024. doi:10.1007/978-3-031-56940-1_20.

5 Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2001.

6 Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, December 2011. doi:10.1145/2043174.2043195.

7 Vittorio Brusoni, Luca Console, Paolo Terenziani, and Daniele Theseider Dupré. A spectrum of definitions for temporal model-based diagnosis. *Artificial Intelligence*, 102(1):39–79, 1998. doi:10.1016/S0004-3702(98)00044-7.

8 Zhelun Chen, Zheng O'Neill, Jin Wen, Ojas Pradhan, Tao Yang, Xing Lu, Guanjing Lin, Shohei Miyata, Seungjae Lee, Chou Shen, et al. A review of data-driven fault detection and diagnostics for building HVAC systems. *Applied Energy*, 339:121030, 2023. doi:10.1016/j.apenergy.2023.121030.

9 European Commission. Press release: Commission welcomes political agreement on new rules to boost energy performance of buildings across the EU, December 2023. URL: https://ec.europa.eu/commission/presscorner/detail/de/ip_23_6423.

10 German Sustainable Building Council. *Wegweiser klimapositiver Gebäudebestand 2022 [Guide to Climate-positive Building Stock 2022]*. German Sustainable Building Council, 2022.

11 Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987. doi:10.1016/0004-3702(87)90063-4.

12 Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer Set Programming: A Primer*. Springer, 2009. doi:10.1007/978-3-642-03754-2_2.

13 Alexander Feldman, Ingo Pill, Franz Wotawa, Ion Matei, and Johan de Kleer. Efficient model-based diagnosis of sequential circuits. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 2814–2821. AAAI Press, 2020. doi:10.1609/AAAI.V34I03.5670.

14 Gerhard Friedrich and Wolfgang Nejdl. Momo-model-based diagnosis for everybody. In *Sixth Conference on Artificial Intelligence for Applications*, pages 206–213. IEEE, 1990. doi:10.1109/CAIA.1990.89191.

15 Martin Gebser, Amelia Harrison, Roland Kaminski, Vladimir Lifschitz, and Torsten Schaub. Abstract Gringo. *Theory and Practice of Logic Programming*, 15(4-5):449–463, 2015. doi:10.1017/S1471068415000150.

16 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

17 A Gevorgian, S Pezzutto, S Zambotti, S Croce, U Filippi Oberegger, R Lollini, L Kranzl, and A Muller. European building stock analysis. *Eurac Research: Bolzano, Italy*, 2021.

18 Stephan Gspandl, Ingo Pill, Michael Reip, Gerald Steinbauer, and Alexander Ferrein. Belief management for high-level robot programs. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 900–905. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-156.

19 Christoph Hutter, Michael Kappert, Ralph Krause, Patrick Müller, André König, Adrian Gebhardt, Falko Ziller, Peter Giesler, and Frank Zeidler. MFGeb - Methoden zur Fehlerkennung im Gebäudebetrieb [MFGeb - methods for fault detection in building operation], 2022. Final Report. URL: https://ibit.fh-erfurt.de/fileadmin/Dokumente/Projekte/IBIT/mfgeb/Abschlussbericht-MFGEB_2022.pdf.

20 Ledio Jahaj, Stalin Munoz Gutierrez, Thomas Walter Rosmarin, Franz Wotawa, and Gerald Steinbauer-Wagner. A model-based diagnosis integrated architecture for dependable autonomous robots. In *34th International Workshop on Principles of Diagnosis*, 2023.

21 Srinivas Katipamula and Michael R Brambley. Methods for fault detection, diagnostics, and prognostics for building systems—a review, part ii. *HVAC&R Research*, 11(2):169–187, 2005. doi:10.1080/10789669.2005.10391123.

22 Roxane Koitz-Hristov and Franz Wotawa. Faster horn diagnosis-a performance comparison of abductive reasoning algorithms. *Applied Intelligence*, 50(5):1558–1572, 2020. doi:10.1007/S10489-019-01575-5.

23 Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, March 1986. doi:10.1007/BF03037383.

24 Gerda Langer, Thomas Hirsch, Roman Kern, Theresa Kohl, and Gerald Schweiger. Large language models for fault detection in buildings' HVAC systems. In *Energy Informatics Academy Conference*, pages 49–60. Springer, 2024. doi:10.1007/978-3-031-74741-0_4.

25 Vladimir Lifschitz. *Answer Set Programming*. Springer, 2019. doi:10.1007/978-3-030-24658-7.

26 Jiefei Ma, Rob Miller, Leora Morgenstern, and Theodore Patkos. An epistemic event calculus for ASP-based reasoning about knowledge of the past, present and future. In Ken Mcmillan, Aart Middeldorp, Geoff Sutcliffe, and Andrei Voronkov, editors, *LPAR-19. 19th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 26 of *EPiC Series in Computing*, pages 75–87, Stellenbosch, South Africa, 2013. EasyChair. doi:10.29007/zswj.

27 John McCarthy and Patrick J Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4, 1969. doi:10.1016/B978-0-934613-03-3.50033-7.

28 Sheila A McIlraith. Representing actions and state constraints in model-based diagnosis. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, pages 43–49, 1997. URL: http://www.aaai.org/Library/AAAI/1997/aaai97-007.php.

29 Rob Miller and Murray Shanahan. Some alternative formulations of the event calculus. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, volume 2408 of *Lecture Notes in Computer Science*, pages 452–490. Springer, 2002. doi:10.1007/3-540-45632-5_17.

30 Payam Nejat, Fatemeh Jomehzadeh, Mohammad Mahdi Taheri, Mohammad Gohari, and Muhd Zaimi Abd. Majid. A global review of energy consumption, CO2 emissions and policy in the residential sector (with an overview of the top ten CO2 emitting countries). *Renewable and Sustainable Energy Reviews*, 43:843–862, 2015. doi:10.1016/j.rser.2014.11.066.

31 Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999. doi:10.1023/A:1018930122475.

32 Panayiotis M Papadopoulos, Vasso Reppa, Marios M Polycarpou, and Christos G Panayiotou. Distributed diagnosis of actuator and sensor faults in HVAC systems. *IFAC-PapersOnLine*, 50(1):4209–4215, 2017. doi:10.1016/j.ifacol.2017.08.816.

33 Ingo Pill and Thomas Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1053–1059. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6595.

34 Liliana Marie Prikler and Franz Wotawa. Faster diagnosis with answer set programming. In *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*, pages 24–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/OASIcs.DX.2024.24.

35 Gregory Provan. Generating reduced-order diagnosis models for HVAC systems. In *Itnl. Workshop on Principles of Diagnosis, Murnau, Germany*, 2011.

36 Aibing Qiu, Ze Yan, Qiangwei Deng, Jianlan Liu, Liangliang Shang, and Jingsong Wu. Modeling of HVAC systems for fault diagnosis. *IEEE Access*, 8:146248–146262, 2020. doi:10.1109/ACCESS.2020.3015526.

37 Olivier Raiman, Johan de Kleer, Vijay Saraswat, and Mark Shirley. Characterizing non-intermittent faults. In *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*, AAAI'91, pages 849–854. AAAI Press, 1991. URL: http://www.aaai.org/Library/AAAI/1991/aaai91-132.php.

38 Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

39 Erich Sewe. *Automatisierte Fehlererkennung in Heizungsanlagen [Automated Fault Detection in Heating Systems]*. PhD thesis, Universität Dresden, 2018.

40 Roni Stern, Meir Kalech, Shelly Rogov, and Alexander Feldman. How many diagnoses do we need? *Artificial Intelligence*, 248:26–45, 2017. doi:10.1016/J.ARTINT.2017.03.002.

41 Peter Struss, Raymond Sterling, Jesús Febres, Umbreen Sabir, and Marcus M. Keane. Combining engineering and qualitative models to fault diagnosis in air handling units. In *Proceedings of the Twenty-First European Conference on Artificial Intelligence*, ECAI'14, pages 1185–1190. IOS Press, NLD, 2014. doi:10.3233/978-1-61499-419-0-1185.

42 Michael Thielscher. A theory of dynamic diagnosis. electronic transactions on artificial intelligence. *Electron. Trans. Artif. Intell.*, 1:73–104, 1997. URL: http://www.ep.liu.se/ej/etai/1997/004/.

43 Michael Thielscher. Introduction to the fluent calculus. *Electron. Trans. Artif. Intell.*, 2:179–192, 1998. URL: http://www.ep.liu.se/ej/etai/1998/006/.

44 Balaje T. Thumati, Miles A. Feinstein, James W. Fonda, Alfred Turnbull, Fay J. Weaver, Mark E. Calkins, and S. Jagannathan. An online model-based fault diagnosis scheme for HVAC systems. In *2011 IEEE International Conference on Control Applications (CCA)*, pages 70–75, 2011. doi:10.1109/CCA.2011.6044486.

45 Christoffer Thuve and Hema Pushphika Yuvraj. Life cycle assessment of a combustion engine-mapping the environmental impacts and exploring circular economy. Master's thesis, Chalmers University of Technology, 2022.

46 Louise Travé-Massuyès and Teresa Escobet. Bridge: Matching model-based diagnosis from FDI and DX perspectives. *Fault Diagnosis of Dynamic Systems: Quantitative and Qualitative Approaches*, pages 153–175, 2019. doi:10.1007/978-3-030-17728-7_7.

47 Franz Wotawa. On the use of answer set programming for model-based diagnosis. In Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Jun Sasaki, editors, *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices - 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings*, volume 12144 of *Lecture Notes in Computer Science*, pages 518–529. Springer, 2020. doi:10.1007/978-3-030-55789-8_45.

48 Franz Wotawa and David Kaufmann. Model-based reasoning using answer set programming. *Appl. Intell.*, 52(15):16993–17011, 2022. doi:10.1007/S10489-022-03272-2.

Are Diagnostic Concepts Within the Reach of LLMs?

Anna Sztyber-Betley
Warsaw University of Technology, Poland

Elodie Chanthery
LAAS-CNRS, INSA, University of Toulouse, France

Louise Travé-Massuyès
LAAS-CNRS, University of Toulouse, France

Silke Merkelbach
Fraunhofer IEM, Paderborn, Germany

Karol Kukla
Warsaw, Poland

Maxence Glotin
LAAS-CNRS, INSA, University of Toulouse, France

Alexander Diedrich
Helmut-Schmidt-University, Hamburg, Germany

Oliver Niggemann
Helmut-Schmidt-University, Hamburg, Germany

Abstract

Model-based diagnosis is a cornerstone of system health monitoring, allowing for the identification of faulty components based on observed behavior and a formal system model. However, obtaining a useful and reliable model is often an expensive and manual task. While the generation of a formal model was the aim of previous work, in this paper, we propose a methodology to use large language models to generate Minimal Structurally Overdetermined sets (MSOs). MSOs are specific subsets of the model equations from which diagnosis tests can be obtained. We investigate two different directions: (i) the large-language-models' ability to generate MSO sets for hybrid systems, similar to those generated by the well-known Fault Diagnosis Toolbox (FDT) (ii) the automated generation of MSOs for Boolean circuits, as well as the computation of the diagnoses. We thus show how both dynamic and static systems can be analysed by large-language models and how their output can be used for effective fault diagnosis. We evaluate our approach on a set of arithmetic and logic circuits, using OpenAI's LLMs 4o-mini, o1, and o3-mini.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Fault Diagnosis, Large Language Models, LLMs, Model Based Diagnosis, MSO, Redundancy Relations, Conflicts, Diagnoses

Digital Object Identifier 10.4230/OASIcs.DX.2025.2

Supplementary Material

Software (Source code): `https://github.com/asztyber/LLM_diagnostic_concepts` [18]
archived at `swh:1:dir:d265fcfc25d623bff2fa34d48590c7eb66d08995`

Software (Source code): `https://github.com/jadlownik/FaultDiagnosis/tree/develop` [10]
archived at `swh:1:dir:b8a40cc86347d60c8efb7ce10df9343400eee88e`

Funding *Elodie Chanthery*: The work is supported by ANITI through the French "Investing for the Future – P3IA" program under the Grant agreement n°ANR-19-P3IA-0004.
Louise Travé-Massuyès: The work is supported by ANITI through the French "Investing for the Future – P3IA" program under the Grant agreement n°ANR-19-P3IA-0004.

Acknowledgements This work has benefited from participation in Dagstuhl Seminar 24031 "Fusing Causality, Reasoning, and Learning for Fault Management and Diagnosis".

1 Introduction

Model-based diagnosis (MBD) is a cornerstone of system health monitoring, allowing for the identification of faulty components based on observed behavior and a formal system model. One common approach within MBD is structural analysis [8]. Structural analysis is used to obtain Analytical Redundancy Relations (ARRs) for fault diagnosis. Performing structural analysis means abstracting the system model by keeping only the links between equations and variables. The main advantages are that it can be applied to large-scale systems, linear or non-linear, and even under uncertainty. Structural analysis uses the computation of Minimal Structurally Overdetermined sets (MSOs), i.e. sets that contain one equation more than the number of variables. Due to this overdetermined property, they play a key role in detecting and isolating faults [4]. Traditionally, the computation of MSOs is performed using dedicated tools such as the Fault Diagnosis Toolbox (FDT) [5], which apply algorithmic methods grounded in analytical redundancy and structural analysis. On the other hand, the main disadvantage of using tools such as the FDT is that they require a correct and reliable model of the underlying system.

With the recent rise of large language models (LLMs), which demonstrate impressive capabilities in reasoning, synthesis, and symbolic manipulation, a question naturally arises: can these models be repurposed to perform structured engineering tasks, such as generating MSO sets directly from system diagrams, thus reducing the need to rely on manually specified models? If so, this could open new avenues for intuitive, rapid prototyping and integration of natural language interfaces into diagnostic systems.

In this work, we propose an exploratory evaluation of LLMs' ability to generate MSO sets from engineering documentation, comparing its outputs to those produced by the Fault Diagnosis Toolbox [5]. We aim to assess both the correctness and completeness of the MSO sets proposed, as well as to identify the strengths and limitations of using LLMs in such a formal and structured context. We perform a similar analysis for the generation of conflicts and diagnoses. Our goal is not only to measure performance, but also to understand the potential and boundaries of LLMs in a field traditionally dominated by rule-based and algorithmic approaches.

We carry out the test on the OpenAI models: 4o-mini, o1[1], and o3-mini[2], as the o-series of models shows leading performance in math, coding, and science questions[3], and 4o-mini is a time and cost effective alternative.

Recent advancements in LLMs have spurred their application in fault diagnosis across a range of complex systems. These models are being explored not only for their fault detection capabilities but also for their ability to provide interpretability, adaptability, and generalization in data-driven diagnostics. Several works have proposed hybrid frameworks that integrate LLMs with traditional diagnosis tools to improve explainability and operator support. For instance, the authors of [3] combine a physics-based diagnostic tool with an LLM to enhance fault interpretability in nuclear power plants, demonstrating improved transparency and operator interaction through natural language explanations. In software engineering, AutoFL [6] employs LLMs for fault localization, enhancing developer trust by providing rationales for fault hypotheses and navigating large code-bases using prompt engineering. Similarly, LLMAO [21] introduces a fine-tuned LLM architecture capable of

[1] https://openai.com/o1/
[2] https://openai.com/index/openai-o3-mini/
[3] Shortly before the submission of the paper the new models were released: o3 and o4-mini, so further improvement can be expected.

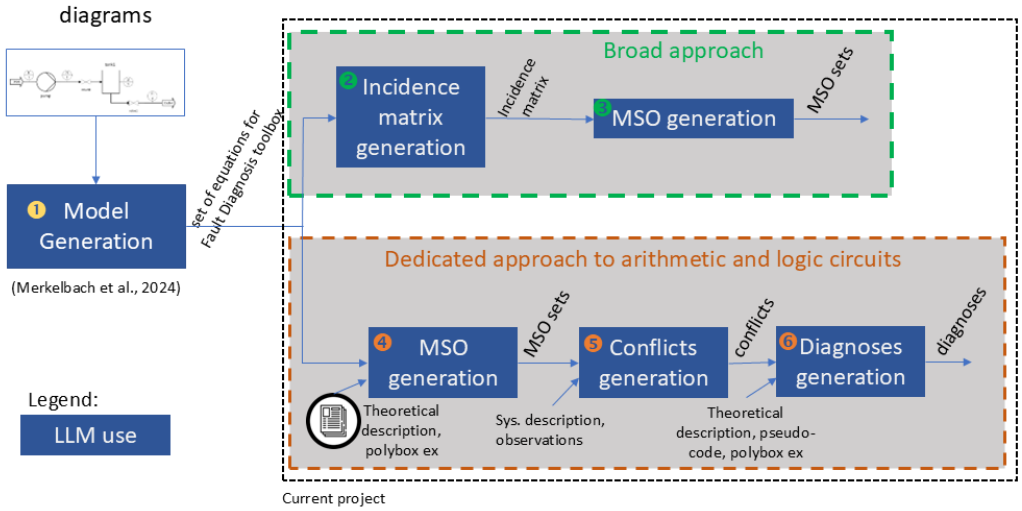


Figure 1 The overall methodology. Model generation was covered in previous work [12]. In both approaches MSOs are generated. The broad approach is applicable to many types of systems, in particular hybrid systems. The dedicated approach is for logic circuits.

fault localization at a line level without relying on test coverage data, surpassing traditional ML-based methods. From an industrial systems perspective, various methods have been proposed to adapt LLMs to multi-modal and time-series data. FD-LLM [11] aligns LLMs with engineering data using modal alignment and prompt learning to handle overlapping features in complex equipment diagnostics. FaultExplainer [7] uses LLMs to generate plausible and actionable explanations, but also highlights its limitations, including reliance on PCA-selected features and some hallucinations. In cyber-physical systems, FLEX [13] and FaultLines [14] demonstrate how open-source LLMs can perform anomaly and fault detection using retrieval-augmented generation and prompt engineering. These works reveal that model performance is closely tied to prompt design and textual encoding strategies.

LLMs also show promise in handling challenges such as cross-domain generalization and data scarcity. The authors of [19] present an LLM-based framework for bearing fault diagnosis that combines textual representation of sensor data with fine-tuning strategies to achieve superior performance across different datasets and operational conditions. The AAD-LLM framework [17] further extends this adaptability by enabling multi-modal, zero-shot anomaly detection in manufacturing systems, using pre-trained LLMs without requiring retraining or finetuning. They also propose a very interesting state of the art on LLMs use for Time Series forecasting. Overall, the field is evolving toward more robust, interpretable, and adaptive diagnostic solutions leveraging LLMs. However, none of these works really links LLM results to concepts developed in the field of model-based diagnostics. This is the major contribution of our article.

This article is organized as follows: Section 2 recalls some important concepts for diagnosis. Section 4 briefly details previous work. Section 5 presents our two main contributions. The road approach for creating MSOs from piping and instrumentation diagrams, and a Dedicated Approach for arithmetic and logic circuits. Finally, Section 6 presents our empirical evaluation. At the end of the article we discuss our results and present some future research directions.

2 Problem Formulation

Figure 1 illustrates our methodology. Input to the methodology are P&ID diagrams and circuit diagrams, i.e. a common form of engineering documentation for use-cases in the process industry and hardware design. Block 1 "Model Generation" has been described in [12]. It generates a set of equations that can be used by the Fault Diagnosis Toolbox to obtain MSO sets. Section 4 briefly recalls the model generation process.

For our solution in this article, we used two independent prompting approaches:

1. A **Broad Approach** (in green box), able to handle continuous systems described by the differential equations that generates MSO sets from a set of differential equations
2. A **Dedicated Approach to arithmetic and logic circuits** (in orange box) that generates MSO sets, conflicts, and finally obtains diagnoses for the considered system.

Blocks 2 and 3 are used for the **Broad Approach**, described in Section 5.2. Blocks 4, 5, and 6 are used for the **Dedicated Approach to arithmetic and logic circuits**, described in Section 5.1. Each block uses an LLM for its reasoning.

In the case of arithmetic and logic circuits, checking the consistency of observations with the model is straightforward and leads to unambiguous answers. In the case of continuous, dynamical systems, in the presence of noise and dynamical fault profiles residual evaluation is more challenging. Therefore, for circuits, we perform all the steps that result in diagnoses (Blocks 4 to 6), and in the Broad Approach we only generate MSO sets (Blocks 2 and 3).

3 Background

A system model $\mathcal{M}(z, x, f)$ typically comprises both *unknown variables*, denoted by x, and *known variables* z, which are measured using sensors.

The variables x are partitioned into two categories: *differential variables* x_1 and *algebraic variables* x_2. Faults affecting the system are explicitly represented by a dedicated vector of parameters f. We denote the sets of known variables, unknown variables, and faults by $\mathcal{Z}$, $\mathcal{X}$, and $\mathcal{F}$, respectively.

A commonly used representation, known as a *semi-explicit Differential Algebraic system of Equations (semi-explicit DAE)*, for general systems in the continuous-time domain is as follows:

$$\mathcal{M}(z,x,f): \begin{cases} \dfrac{dx_1(t)}{dt} = h\big(x_1(t), x_2(t), z_c(t), f\big), \quad x_1(t_0) = x_0 \\ 0 = l\big(x_1(t), x_2(t), z_c(t), f\big), \\ z_s(t) = g\big(x_1(t), x_2(t), f\big). \end{cases} \tag{1}$$

Here, $x_1(t) \in \mathbb{R}^{n_{x_1}}$ and $x_2(t) \in \mathbb{R}^{n_{x_2}}$ denote the vectors of unknown variables. The vectors $z_s(t) \in \mathbb{R}^{n_{z_s}}$ and $z_c(t) \in \mathbb{R}^{n_{z_c}}$ represent the *measured outputs* and *controlled inputs*, respectively, both considered known. In systems without external actuation, $z_c(t)$ may be zero.

The functions h, l, and g can be linear or nonlinear and typically depend on a set of system parameters, denoted by $\mathcal{Z}_p$ (e.g., tank diameter, nominal flow rate, etc.).

Interestingly, dynamic systems (represented by differential equations) and static systems (represented by algebraic equations) are sub-classes of DAEs.

For any vector ν, we define $\bar{\nu}$ as the augmented vector comprising ν and its time derivatives up to a certain (unspecified) order.

▶ **Definition 1** (Analytical Redundancy Relations (ARR)). *ARRs are relations $\mathcal{M}'(\bar{z}) = \mathcal{M}''(\bar{f})$ obtained from $\mathcal{M}(z, x, f)$ by formally eliminating unknown variables x. While $\mathcal{M}''(\bar{f})$ is the internal form that depends on the faults and is not known, $\mathcal{M}'(\bar{z})$ is the computation form and can be computed from the known variables and their derivatives.*

$\mathcal{M}'(\bar{z})$ defines a set of ARRs. A single ARR takes the form $arr_i(\bar{z}') = r_i$, where r_i is a scalar signal named *residual* and $\bar{z}'$ a subvector of $\bar{z}$. It can be used as residual generator.

▶ **Definition 2** (Residual generator for $\mathcal{M}(z, x, f)$). *A relation of the form $arr_i(\bar{z}') = r_i$, with input $\bar{z}'$ a subvector of $\bar{z}$ and output r_i, a scalar signal named* residual*, is a residual generator for the model $\mathcal{M}(z, x, f)$ if, for all z consistent with $\mathcal{M}(z, x, f)$, it holds that $\lim_{t\to\infty} r(t) = 0$.*

In simple terms, a residual generator produces a signal called a *residual*, which ideally remains zero when the system operates correctly. Any deviation from zero may indicate the presence of a fault.

From the system model, it is possible to derive *Analytical Redundancy Relations* (ARRs) by exploiting the inherent analytical redundancy. This is typically achieved through variable elimination techniques, resulting in relations that involve only the known variables. ARRs serve as the foundation for *diagnosis tests*, enabling the verification of whether the measurements z are consistent with the system model $\mathcal{M}(z, x, f)$. If a fault is present in the system, it is expected to affect some of the measurements, and consequently, the residuals. If no such influence is observable, the fault is said to be *non-detectable*, and as such, it cannot be identified [9].

Building on the foundational work by Cassar and Staroswiecki [1], and Travé-Massuyès et al. [20], *structural analysis* has emerged as a powerful tool for deriving ARRs. This method abstracts the system model by focusing solely on the structural relationships between equations and variables, disregarding the specific functional forms. One of the key advantages of structural analysis is its applicability to large-scale systems, whether linear or nonlinear, and even in the presence of model uncertainty. Along the structural approach, the structure of a system $\mathcal{M}(z, x, f)$, or *structural system*, can be represented by a biadjacency matrix crossing variables and equations, named the *Incidence Matrix*.

▶ **Definition 3** (Incidence matrix). *The Incidence Matrix of a system $\mathcal{M}(z, x, f)$ is an $n_e \times (n_x + n_z)$ binary matrix, where $n_x = n_{x_1} + n_{x_2}$ is the number of unknown variables, $n_z = n_{z_s} + n_{z_c}$ is the number of known variables, and n_e is the number of equations. Each row stands for an equation and each column for a variable. A 1 in position (i, j) means that the equation of row i contains the variable of column j.*

When applied to fault diagnosis, structural analysis helps identify subsets of equations that exhibit redundancy. The *structural redundancy* $\rho_{\mathcal{M}'}$ of a set of equations $\mathcal{M}' \subseteq \mathcal{M}$ is defined as the difference between the number of equations and the number of unknown variables within that subset.

Of particular diagnostic relevance are the *Proper Structurally Overdetermined* (PSO) sets, which are sets of equations that together contain just enough redundancy to allow fault detection through consistency checks.

A *just determined part* refers to a set of equations where the number of equations matches the number of unknown variables, allowing for a unique solution. In contrast, an *underdetermined part* has fewer equations than unknowns, making it impossible to uniquely determine the values of all variables.

▶ **Definition 4** (PSO set). *A subset of equations $\psi \subseteq \mathcal{M}(z, x, f)$ is a PSO set if its number of equations is greater than the number of unknown variables, and this overdetermination applies to the entire set (i.e., the set is structurally redundant and contains no exactly or underdetermined parts).*

Another interesting concept are the minimal subsets of equations $\psi \subseteq \mathcal{M}(z, x, f)$ that possess exactly one degree of structural redundancy, i.e., $\rho_\psi = 1$. Such subsets are referred to as *Minimal Structurally Overdetermined* (MSO) sets.

▶ **Definition 5** (MSO set). *A subset of equations $\psi \subseteq \mathcal{M}(z, x, f)$ is an MSO set of $\mathcal{M}(z, x, f)$ if (1) $\rho_\varphi = 1$, and (2) no subset of ψ is overdetermined. The set of MSO sets of $\mathcal{M}$ is denoted Ψ.*

Certain MSO sets have been shown to support the formulation of diagnostic tests [9], making them essential components for residual generation and fault detection. These particular subsets are called *Fault-Driven Minimal Structurally Overdetermined* (FMSO) sets [15].

Let $\mathcal{F}_\varphi$ denote the set of faults involved in a given subset of equations $\varphi \subseteq \mathcal{M}(z, x, f)$, then FMSO sets can be defined as follows.

▶ **Definition 6** (FMSO set). *A subset of equations $\varphi \subseteq \mathcal{M}(z, x, f)$ is an FMSO set of $\mathcal{M}(z, x, f)$ if (1) φ is an MSO set and (2) $\mathcal{F}_\varphi \neq \emptyset$. The set of FMSO sets of $\mathcal{M}$ is denoted Φ.*

Given an FMSO set φ, $\mathcal{F}_\varphi$ is defined as the *fault support* of φ and the physical components modeled by the equations in φ define its *component support* $COMP_\varphi$.

In most cases, FMSO sets can be transformed into Analytical Redundancy Relations (ARRs) through sequential elimination. Due to their structural properties, all unknown variables appearing in an FMSO set φ can be algebraically resolved using $|\varphi| - 1$ equations. These expressions can then be substituted into the remaining $|\varphi|$-th equation, yielding an ARR formulated off-line. This ARR is subsequently employed on-line as a diagnostic test.

Building upon the findings in [2], which establish a connection between model-based approaches from the control community (FDI) and those from the artificial intelligence community (DX), the concept of an *R-conflict*, originally introduced by Reiter [16], can be related to FMSO sets under the assumption of *Model Representation Equivalence*.

▶ **Definition 7** (R-conflict). *The component support $COMP_\varphi$ of an FMSO set φ is an R-conflict if the diagnosis test derived from φ fails when evaluated with the measurements obtained from the physical system. A minimal R-conflict is an R-conflict that does not strictly include (set inclusion) any R-conflict.*

An R-conflict indicates that at least one component within the R-conflict must be faulty to explain the observed behavior; equivalently, it is not possible for all components in the R-conflict to be functioning normally. Interestingly, the set of minimal diagnoses can be generated from the set of minimal R-conflicts.

▶ **Definition 8** (Minimal diagnosis). *A set of physical components Δ, modeled by some of the equations of $\mathcal{M}(z, x, f)$, is a minimal diagnosis if and only if it is a minimal hitting [4] set of the collection of minimal R-conflicts.*

Figure 2 illustrates the different concepts that have been introduced on the well-known polybox example.

[4] A hitting set for a collection $\mathcal{C}$ of sets is a set $H \subseteq \bigcup\{S/S \in \mathcal{C}\}$ such that $H \cap S \neq \{\}$ for each $S \in \mathcal{C}$. A hitting set is minimal if and only if no proper subset of it is a hitting set for $\mathcal{C}$.

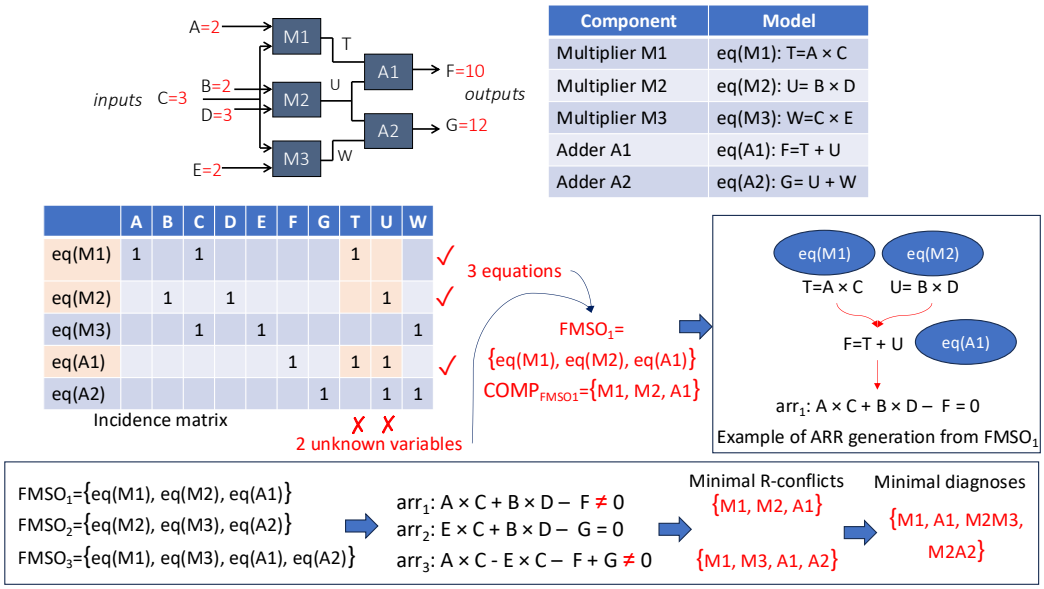


Figure 2 Illustration of the concepts using the polybox example. In the broad approach we output only the MSOs, while in the dedicated approach we output diagnoses.

The notion of FMSO set also plays a pivotal role in the formal definitions of *structural detectable* and *structural isolable* faults. In the following, we revisit these fundamental definitions as presented in [9].

▶ **Definition 9** (Detectable fault). *A fault $f \in \mathcal{F}$ is structurally detectable in the system $\mathcal{M}(z, x, f)$ if there exists an FMSO set $\varphi \in \Phi$ such that $f \in \mathcal{F}_\varphi$.*

▶ **Definition 10** (Isolable faults). *Given two structurally detectable faults f and f' of $\mathcal{F}$, $f \neq f'$, f is structurally isolable from f' if there exists an FMSO set $\varphi \in \Phi$ such that $f \in \mathcal{F}_\varphi$ and $f' \notin \mathcal{F}_\varphi$.*

The *Fault Diagnosis Toolbox* (FDT) [5] is a software framework designed for the analysis and synthesis of fault diagnosis in systems that are typically modeled with differential-algebraic equations. By leveraging a structural representation of the system, the toolbox enables the automatic extraction of FMSO sets. From these sets, it allows one to generate Analytical Redundancy Relations (ARRs), which can subsequently be used as diagnosis tests.

4 Model generation

Previous work [12] proposes an approach to create mathematical models for process industry systems using multi-modal large language models (MLLM). It presents a five-step prompting approach that uses a piping and instrumentation diagram (P&ID) and natural language prompts as its input. For clarity, we will briefly sketch the previous approach in this section.

The five steps of the prompting approach are the following:

1. **Read Diagram** – Let the MLLM read the diagram image data and represent it in some partially specified intermediate format. Partial specification is performed in the prompt.
2. **Identify Sensors** – The LLM creates a table containing existing sensors, their type, and their placement from the input diagram and from the output from step 1 in the form of *CompsConnections*. Context about P&IDs, possibly occurring sensors, and the placement of the sensors in the diagram are provided in the system message.

3. **Create the mathematical Equations** – The mathematical equations are created by the LLM using provided variable names and assumptions.
4. **Sensor Matching and Variable Assignment** – In this step, all the results from the previous steps are merged, the faults are added to the equations and the resulting mathematical model $\mathcal{M}$ is created.
5. **Format Model for Fault Diagnosis Toolbox** – In the final step, the mathematical model $\mathcal{M}$ is transformed to be suitable for the FDT into $\tilde{\mathcal{M}}$.

All steps provide the MLLM with the P&ID as input. In addition, the steps take a system message and a user prompt as input. The system message contains the generic task for the respective step. However, taking current abilities of MLLMs into account, it is still domain dependent. The user message contains a subset of the external information $\mathcal{I}' \subset \mathcal{I}$ which is specific for the system, where $\mathcal{I}$ is denotes all available external information. The prompts can be found on GitHub[5]. The approach aims to work for water tank systems with standard components, such as tanks, pumps, valves, flow indicators, and level indicators.

5 Generating MSO Sets, Conflicts, and Diagnoses

As described before, for our solution in this article, we use two independent prompting approaches:

1. **Dedicated approach** – designed to handle arithmetic and logic circuits,
2. **Broad approach** – able to handle continuous systems described by differential equations.

While the Dedicated Approach outputs proper diagnoses, the Broad Approach outputs the more general FMSO sets from which diagnoses can be computed using actual observations and tools such as the Fault Diagnosis Toolbox.

5.1 Dedicated approach to arithmetic and logic circuits

To generate the diagnoses, we test the following approach consisting of three steps:

1. **Generation of MSO sets** – We provide system equations and the sets of X, Z, and F variables as inputs. The model is asked to output all Minimal Structurally Overdetermined (MSO) Sets. This is illustrated by block 4 in Figure 1. Table 1 presents the prompt. The prompt contains a theoretical description and the Polybox system.
2. **Generation of Conflicts** – We provide the system description, the MSO sets, and observations as inputs. The LLM is asked to evaluate all MSOs for consistency and to generate all conflicts. The prompt contains the theoretical description, detailed steps, and hints for the Python implementation (we test variants with code interpreter allowed) and three simple examples. This is illustrated by block 5 in Figure 1.
3. **Generation of Diagnoses** – We provide the set of conflicts as inputs. The model is asked to generate all minimal diagnoses. The prompt contains the theoretical description, pseudo-code of the algorithm, and Polybox example. This is illustrated by block 6 in Figure 1.

Figure 3 illustrates the inputs and outputs of each step in the Dedicated Approach (we follow the example from Figure 2). We start with a system description given by a set of equations (system model $\mathcal{M}(z, x, f)$, Equation (1)) and generate MSO sets (Definition 5)

[5] `https://github.com/silkeme/DX24_Model_Creation_LLMs`
The repository contains the prompts, all mentioned information about the systems used for the evaluation, the resulting mathematical models, and all intermediate outputs.

using the prompt given in Table 1. Next, residual generators (Definition 2), resulting from each MSO, are evaluated given the values of known variables z. The value of a residual generator different from 0 indicates the existence of R-conflict (Definition 7). The output of the second step is the set of conflicts. In the last step, the conflicts are used to generate minimal diagnoses (Definition 8).

The prompts for the computation of conflicts and diagnoses can be found on GitHub[6].

Table 1 Prompt for computation of Minimal Structurally Overdetermined (MSO) Sets (Dedicated Approach).

Perform an analysis of the equation system equations and identify all Minimal Structurally Overdetermined (MSO) sets.

Definitions:

1. Unknown variables: These are only the variables that start with "`x`".
2. Structural redundancy: This is the difference between the number of equations and the number of unique unknown variables in those equations:

   $R =$ (number of equations) − (number of unique unknowns).
3. PSO (Properly Structurally Observable): A subset is PSO if its structural redundancy is greater than 0.

Conditions:

1. The selected set of equations must have exactly one structural redundancy.
2. None of its proper subsets can be PSO (all must have redundancy ≤ 0).

Output: JSON format: `{ "mso": [...] }`, where each inner list represents a valid MSO set.

RESPONSE MUST BE IN JSON FORMAT. DO NOT GENERATE CODE AND RETURN IT IN RESPONSE.

```
<example>
<input>
equations = {
  'M1':  'a * c = x01',
  'M2':  'b * d = x02',
  'M3':  'c * e = x03',
  'A2':  'x01 + x02 = f',
  'A1':  'x02 + x03 = g',
}
</input>
<output>
{  "mso" = [['M1', 'M2', 'A1'], ['M2', 'M3', 'A2'], ['M1', 'M3', 'A1', 'A2']] }
</output>
</example>
```

5.2 Broad approach

To generate the MSO sets, the approach consists of two steps:

1. **Generation of the Incidence Matrix** – As input, we provide system equations. The model is asked to return the incidence matrix as a Python dictionary. This is illustrated by block 2 in Figure 1. The prompt for the variant without code interpreter is shown in Table 2.

[6] `https://github.com/asztyber/LLM_diagnostic_concepts/blob/main/prompts/dedicated/prompts_Karol.py` The file contains three prompts for the three mentioned steps. Code to run the experiments can be found in `https://github.com/jadlownik/FaultDiagnosis/tree/develop`

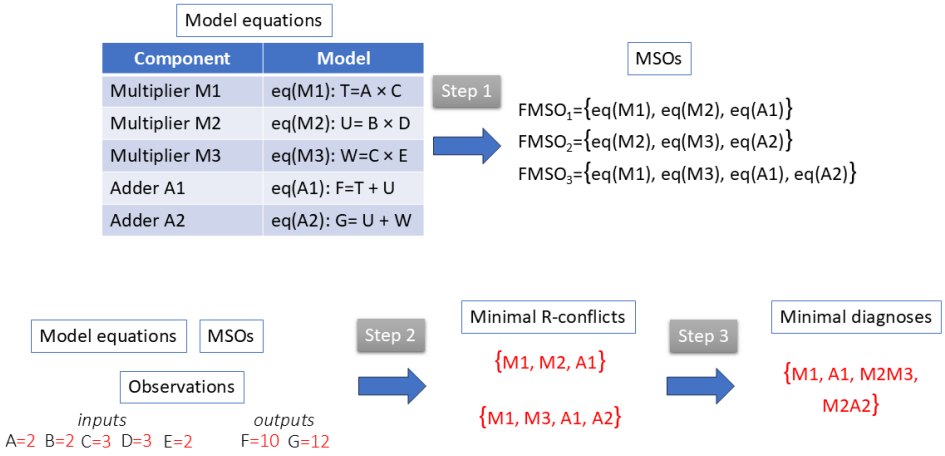


Figure 3 Inputs and outputs of each step in Dedicated Approach.

2. **Generation of MSO sets** – We provide the incidence matrix including only unknown variables as input. The model is asked to find all the MSO sets for the provided model in two sub-steps. First, find all the PSO sets (Definition 4), then, within these PSO sets, find the minimal ones. Two versions were implemented, one with code interpreter and one without code interpreter. This is illustrated by block 3 in Figure 1. The prompt for the variant without code interpreter is shown in Table 3.

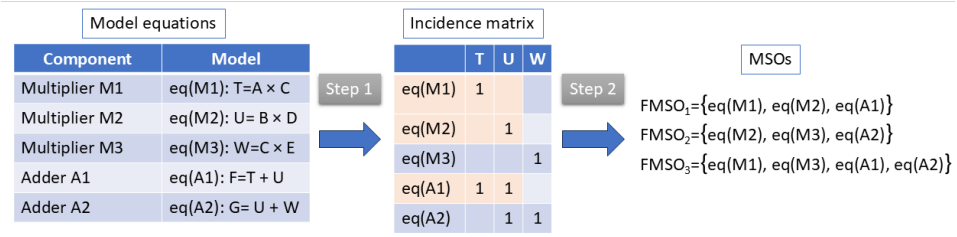


Figure 4 Inputs and outputs of each step in Broad Approach.

Figure 4 illustrates the inputs and outputs of the two steps for finding the MSO sets. In the first step, the incidence matrix is computed (Definition 3). Only the part containing unknown variables x is computed, which is the important part for MSO generation. In the second step, MSOs (Definition 5) are computed from the incidence matrix via PSO sets.

The full set of prompts (including variant for use of code interpreter) can be found on GitHub[7].

[7] `https://github.com/asztyber/LLM_diagnostic_concepts/blob/main/prompts/broad/` The directory contains prompts and code to run them.

Table 2 Prompt for computation of Incidence Matrices (Broad Approach).

Your job is to create an incidence matrix of the provided model. In the incidence matrix, each row represents an equation and each column represents an UNKNOWN variable. Note that unknown variables are stored under key 'x' in the provided model. Return the incidence matrix. Please return the matrix as a Python dictionary in the following format: 'eq0': ['unknown var in eq0', ...], 'eq1': ['unknown var in eq1', ...], ... Each line in the dictionary should correspond to one equation (do not write everything on the same line). Only return the dictionary. Be careful with the equations of type 'fdt.DiffConstraint'. Change those equations by a simple equality relation. If you have fdt.DiffConstraint(dt,t), change it by t - dt. Make sure to complete your analysis before responding to me.

Table 3 Prompt for computation of Minimal Structurally Overdetermined (MSO) Sets from Incidence Matrices (Broad Approach).

Your job is to find all the MSO (Minimal Structurally Overdetermined) sets for the provided model, represented here by the provided incidence matrix. In the incidence matrix, each line represents an equation with the unknown variables that belong to this equation. Note that the matrix contains only the unknown variables. MSO sets consist of equations and contain one more equation than the number of unknown variables. Proceed in two parts. First, find all the PSO (Proper Structurally Overdetermined) sets. Then, within these PSO sets, collect those that are minimal (PSO sets that do not contain any subsets that are PSO) to keep only the MSO sets. Store all the MSO sets in a python dictionary and return only the dictionary to me. Use one line for each MSO set. Additionally, include the number of MSO sets found in the dictionary. You MUST only return the dictionary. Don't make up an example, work with the provided incidence matrix. Make sure to complete your analysis before responding to me.

6 Test and Validation

All approaches were evaluated through OpenAI API. We evaluated the following models: gpt-4o-mini-2024-07-18 (referred in the following as 4o-mini), o3-mini-2025-01-31 (referred in the following as o3-mini), and o1-2024-12-17 (referred in the following as o1). 4o-mini was evaluated with the access to the code interpreter, therefore the model could write Python code and run it to generate the answer[8]. o1 and o3-mini are reasoning models[9], trained with reinforcement learning to perform complex reasoning, and performing all computations in internal chain of thought. Note that we cannot tell for sure that these models did not use a code interpreter in the background as well.

Experiments were performed with temperature 0 and top_p = 0 for 4o-mini. The reasoning effort for o1 and o3-mini was 'high' for the Dedicated Approach and 'medium' for the Broad Approach. Tests were repeated 10 times for each test case (test case including example description and the set of observations) for 4o-mini and o3-mini. Tests for o1 were performed only once, because of costs[10].

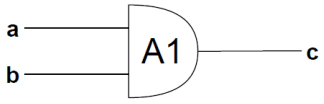
Each LLM reasoning step was evaluated independently, i.e. for each blue box in Figure 1 we provided *correct inputs*, irrespective of the result of the previous step. That way, we can evaluate the accuracy of each algorithm step reliably.

[8] We tried gpt-4o and gpt-4o-mini but the accuracy was close to zero on unseen examples. Performance of gpt-4o and gpt-4o-mini with code interpreter was similar in initial experiments, so we decided to evaluate full set of examples on gpt-4o-mini because of costs.

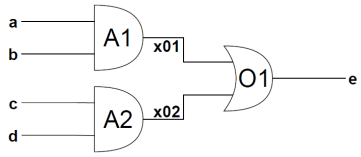
[9] `https://openai.com/index/learning-to-reason-with-llms/`

[10] One iteration costed around 30$

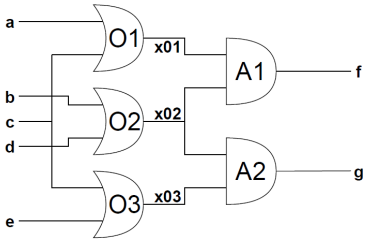
We carried out tests on 23 arithmetic and logic circuits with different levels of complexity (Figure 5). The circuits contain adders, multipliers, and AND, OR, NOR, NAND, and XOR gates. Table 4 summarizes examples' characteristics[11]. The number of minimal conflicts and diagnoses can vary for a given example depending on the observations[12]. For each example, we assumed all inputs and outputs are measured. We only considered components faults. Sensor faults and sensor equations were omitted.



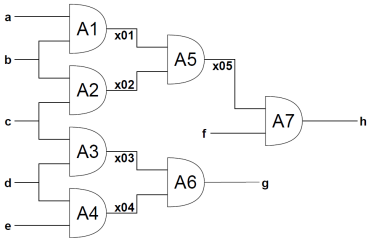
(a) Example 10.



(b) Example 11.



(c) Example 12.



(d) Example 13.

Figure 5 Selected examples with different level of complexity.

As shown in Figure 6, we evaluate the models' performance using complementary metrics that capture different aspects of accuracy in incidence matrix generation. For each equation, precision measures how many of the model's predicted variables are correct, while recall measures how many of the correct variables the model identified. The F1 score combines these into a single metric, penalizing both missing and spurious variables. The accuracy metric is the strictest, requiring perfect variable sets for each equation.

4o-mini often includes known variables and faults as variables involved in incidence matrix, resulting in the poor scores.

Ground truth for incidence matrices, MSO sets, and residual evaluation was computed using Fault Diagnosis Toolbox [5]. Correct minimal diagnoses were computed with the implementation of minimal hitting sets algorithm.

The evaluation metrics (Precision, Recall, and F1 score) were computed by comparing the generated MSO sets with the ground truth MSO sets. Precision measures the fraction of correctly identified MSO sets among all generated sets ($\frac{TP}{TP+FP}$), while Recall indicates the fraction of ground truth MSO sets that were successfully found ($\frac{TP}{TP+FN}$)[13]. The F1 score is the harmonic mean of Precision and Recall ($2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$). The metrics for conflicts and diagnoses were computed in an analogous way.

[11] You can find image for each example here `https://github.com/jadlownik/FaultDiagnosis/tree/develop/images` and descriptions here `https://github.com/asztyber/LLM_diagnostic_concepts/tree/main/examples`

[12] The results contain readable Excel files containing generated and correct MSOs, conflicts and diagnoses.

[13] In the case where the number of ground truth solutions is zero ($P = 0$), and the generated solution is also empty ($FP = 0$) we set Precision and Recall to 1, to avoid penalizing correct answers for the empty solutions.

Table 4 Example Characteristics.

Example	$\|X\|$	$\|Z\|$	$\|F\|$	Relations	MSOs	Conflicts	Diagnoses
Example 2	10	7	5	5	3	0–3	0–8
Example 3	13	8	7	7	2	0–2	0–12
Example 4	13	8	7	7	2	0–2	0–12
Example 5	13	8	7	7	2	0–2	0–12
Example 6	17	11	8	8	3	0–3	0–21
Example 7	13	9	6	6	3	0–2	0–5
Example 8	18	10	11	11	6	0–5	0–45
Example 9	29	17	15	15	6	0–6	0–165
Example 10	3	3	1	1	1	0–1	0–1
Example 11	7	5	3	3	1	0–1	0–3
Example 12	10	7	5	5	3	0–2	0–5
Example 13	13	8	7	7	2	0–2	0–12
Example 14	13	8	7	7	2	0–2	0–12
Example 15	13	8	7	7	2	0–2	0–12
Example 16	17	11	8	8	3	0–3	0–21
Example 17	13	9	6	6	3	0–3	0–9
Example 18	18	10	11	11	6	0–5	0–57
Example 19	29	17	15	15	6	0–5	0–153
Example 20	12	9	5	5	3	0–3	0–8
Example 21	12	9	5	5	3	0–3	0–8
Example 22	7	5	3	3	1	0–1	0–3
Example 23	3	3	1	1	1	0–1	0–1
Example 24	7	5	3	3	1	0–1	0–3

The Figures 8, 10, and 12 show F1 scores for different model versions (4o-mini, o1, and o3-mini). Each data point represents the mean F1 score for examples with the same number of solutions (MSOs, conflicts, or diagnoses), with error bars indicating the standard deviation across these examples. The dashed lines show the linear trend of performance as the number of solutions increases. The Figures 7, 9, and 11 show F1 scores sorted by the number of relations in the example.

The bar plots (Figures 13, 14, 15) show the aggregated performance metrics (F1 score, Precision, and Recall) for each model (4o-mini, o1, and o3-mini) across all test examples. Each bar represents the mean value of the corresponding metric, with error bars indicating the standard deviation.

7 Discussion

An analysis of the obtained results leads to the following conclusions.

For incidence matrix generation (Figure 6), we can observe that o1 and o3-mini achieve perfect accuracy. 4o-mini tends to include additional variables (not in X but correctly assigned to the equations); otherwise, the results are correct. Incidence matrix generation is the simplest algorithmic task.

For MSO generation (Figure 7), 4o-mini has problems with examples containing only one relation. Otherwise, the performance is relatively stable with an increasing number of examples. 4o-mini has access to a code interpreter, making handling different input

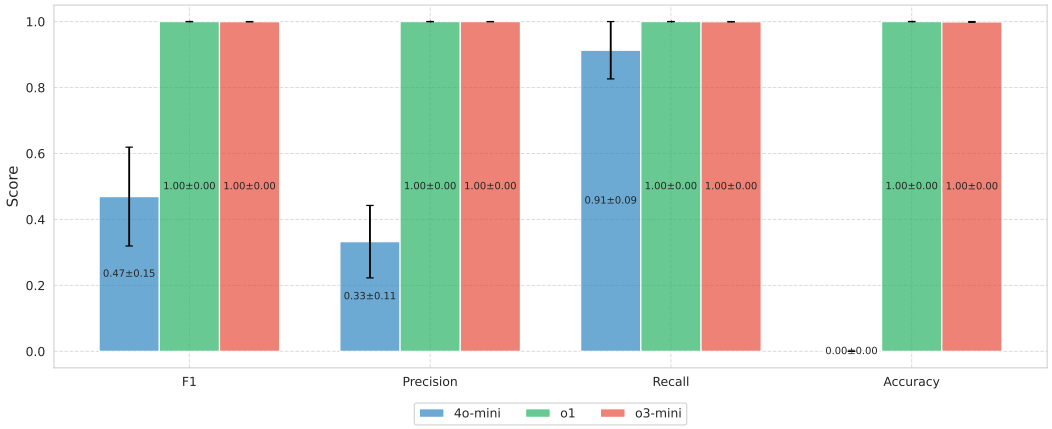


Figure 6 Performance comparison of three models (4o-mini, o1, and o3-mini) on incidence matrix generation. The bars show mean values with standard deviations computed across all examples and runs. For each model, we evaluate four metrics: F1 score (harmonic mean of precision and recall), precision (ratio of correctly identified variables to all variables predicted in an equation), recall (ratio of correctly identified variables to all variables that should be in an equation), and accuracy (percentage of equations where all variables match exactly). The precision, recall, and F1 scores are first computed for each equation separately and then averaged across all equations in an example. The accuracy represents a stricter metric, as it only counts equations where the model identified exactly the right set of variables, with no missing or extra variables.

sizes easier. The performance of o1 and o3-mini decreases with the number of relations in the example. We can observe similar tendencies depending on the number of MSOs to be generated (Figure 8). Plots on Figure 7 and Figure 8 show results aggregated over two prompting approaches (Dedicated and Broad). Figure 13 compares the performance of different models and prompting approaches. The performance of the Dedicated and Broad Approach is similar despite much more domain knowledge included in the dedicated prompts. Additionally, the Broad Approach includes a correct incidence matrix as an input for MSO generation, but it does not influence the results significantly. o1 shows the best performance across all metrics, while 4o-mini is the worst despite the access to the code interpreter.

Performance in conflict generation, depending on the number of relations and conflicts, is presented respectively in Figure 9 and Figure 10. Trend lines show increasing performance with increasing complexity, but this is caused by poor performance when the number of conflicts is zero (i.e. there are no inconsistencies in the system description and observation). o1 performs best for conflict generation (Figure 14).

Diagnoses generation is the most computation-heavy task. We observe a clear decrease in the performance with increasing complexity (Figure 11 and Figure 12). o3-mini performs best for diagnoses generation (Figure 15).

As LLMs were not primarily designed to handle algorithmic and computational tasks, the performance of reasoning models is surprisingly good. o3-mini achieves $F1 = 0.917 \pm 0.249$ on conflicts generation and $F1 = 0.784 \pm 0.322$ on diagnoses generation. The examples were generated manually and were unavailable in the training data (correct solutions were released after the tested models' release).

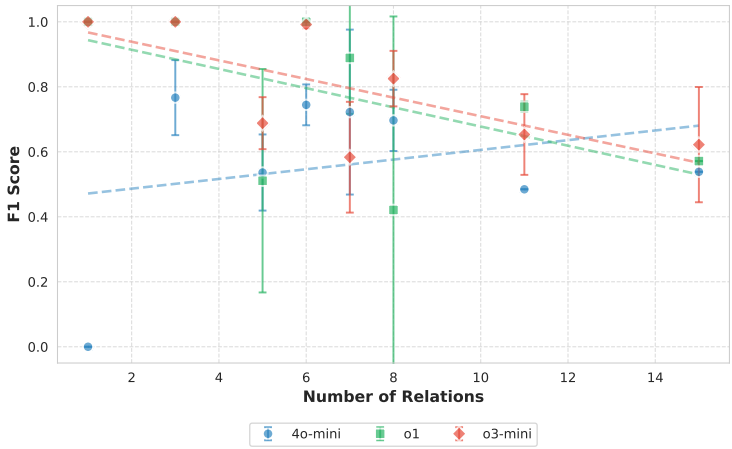


Figure 7 Performance comparison of MSO generation across different versions (4o_mini, o1, o3_mini) relative to the number of relations in examples.

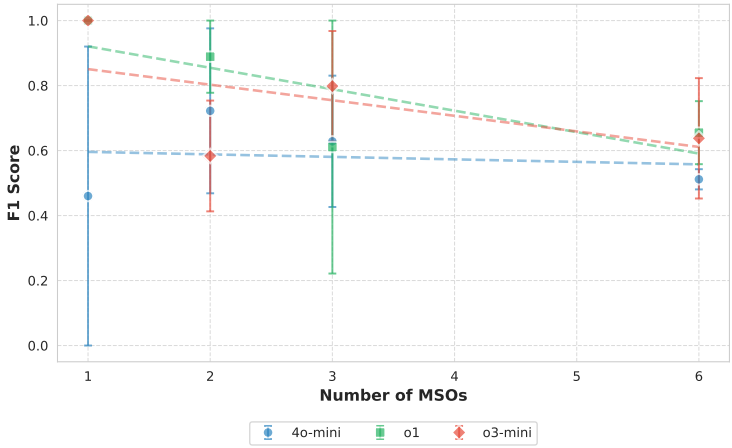


Figure 8 Performance comparison of MSO generation across different versions (4o_mini, o1, o3_mini) relative to the number of MSOs in examples.

8 Conclusion

This paper explores the use of large language models (LLMs) as a novel tool for performing structural analysis in model-based diagnosis (MBD), specifically focusing on the automatic generation of Minimal Structurally Overdetermined (MSO) sets, from which diagnosis tests can be obtained, from engineering documentation. The generation of conflicts and diagnoses is also explored. The authors evaluate the performance of various OpenAI LLMs (including o1, o3-mini, and 4o-mini) by comparing their generated MSOs, conflicts, and diagnoses against those produced by traditional tools. The paper situates its contribution within a growing body of work on LLM-based fault diagnosis but emphasizes its unique connection to foundational MBD concepts. An empirical evaluation highlights both the potential and limitations of LLMs in structured engineering tasks, offering insights into their applicability and future integration into hybrid diagnostic frameworks.

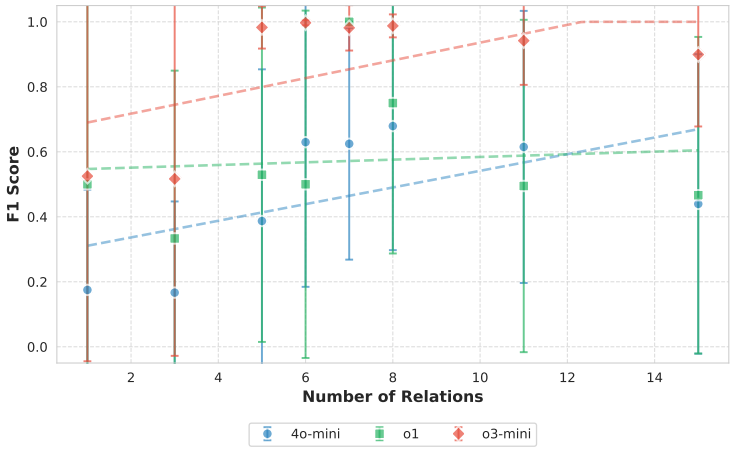


Figure 9 Performance comparison of minimal conflicts generation across different versions (4o_mini, o1, o3_mini) relative to the number of relations in examples.

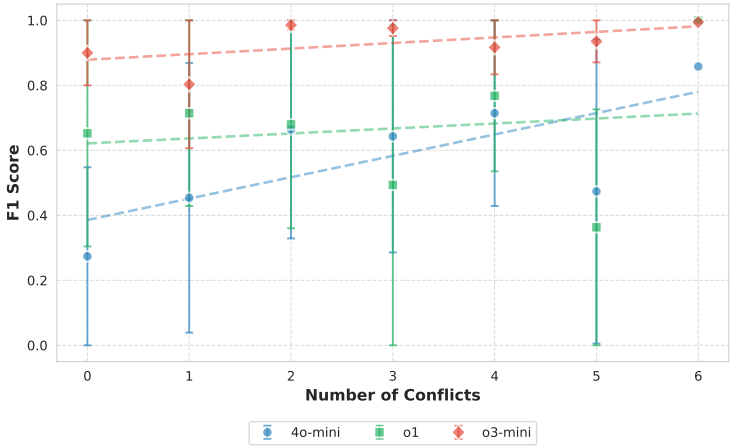


Figure 10 Performance comparison of minimal conflicts generation across different versions (4o_mini, o1, o3_mini) relative to the number of conflicts in examples.

The main goal of this work is to evaluate LLM capabilities in the context of MBD fault diagnosis. While we find some results surprisingly good, at this point, LLMs do not provide an alternative to diagnosis algorithms. The current level of accuracy is not sufficient to replace traditional tools. Additionally, as the results are obtained by API calls, the time to achieve the results depends on the network load and other factors and cannot be bounded robustly. Another practical consideration is the cost. Lastly, it can be observed that the quality of the results vary highly in different runs.

Several avenues can be explored to extend this work. First, broader benchmarking on more diverse and complex systems, including hybrid systems or distributed systems, would help assess the generalization capabilities and scalability of LLMs in diagnostic tasks. Another promising direction involves fine-tuning or instruction-tuning language models on domain-specific corpora to improve their performance. Additionally, integrating multi-modal inputs – such as P&ID diagrams, circuit schematics, or CAD models – via vision-language models may allow more intuitive and direct processing of engineering data.

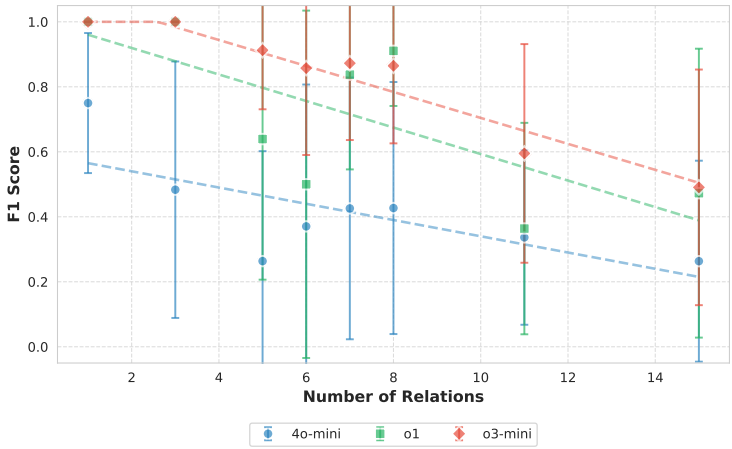


Figure 11 Performance comparison of minimal diagnoses generation across different versions (4o_mini, o1, o3_mini) relative to the number of relations in examples.

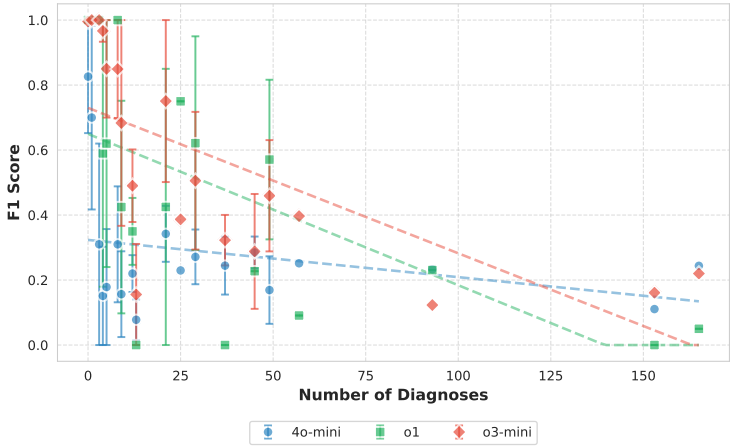


Figure 12 Performance comparison of minimal diagnoses generation across different versions (4o_mini, o1, o3_mini) relative to the number of diagnoses in examples.

Furthermore, future work could explore the use of the recent Model Context Protocol (MCP) in diagnostic frameworks involving distributed architectures. MCP provides a structured way to manage model contexts and execution flows across different computational components. By integrating MCP with language models, it becomes possible to coordinate diagnostic reasoning over a set of specialized servers, each responsible for distinct functionalities – such as structural analysis, variable elimination, or diagnosis generation from conflicts. This would enable a modular and scalable diagnostic pipeline, where LLMs interact with context-aware services rather than operating as monolithic agents. On the other hand, this modular structure is particularly well suited for diagnosis in distributed systems, where different subsystems may operate independently, expose only partial observability, or follow different modeling paradigms. MCP can enable seamless orchestration of context-aware services across these subsystems, allowing LLMs to query relevant components dynamically and integrate partial diagnosis results into a coherent global view. Such an approach enhances scalability, reusability, and flexibility in large-scale or heterogeneous diagnosis environments, while also opening new opportunities for intelligent fault management across distributed infrastructures.

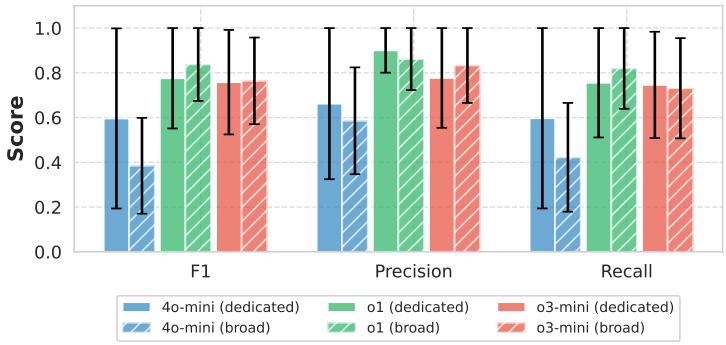


Figure 13 Comparison of F1 score, Precision, and Recall for MSO generation across different models. Each bar shows the mean value with standard deviation.

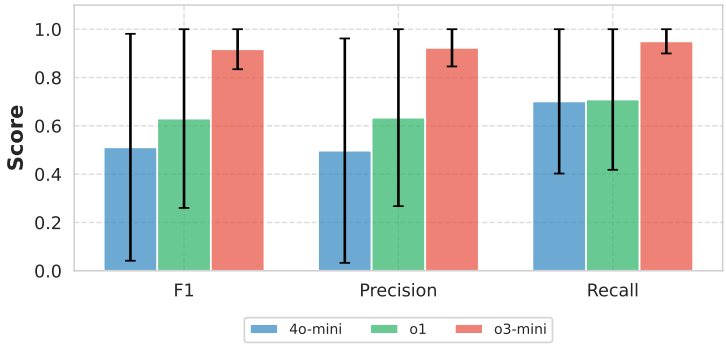


Figure 14 Comparison of F1 score, Precision, and Recall for minimal conflicts generation across different models. Each bar shows the mean value with standard deviation.

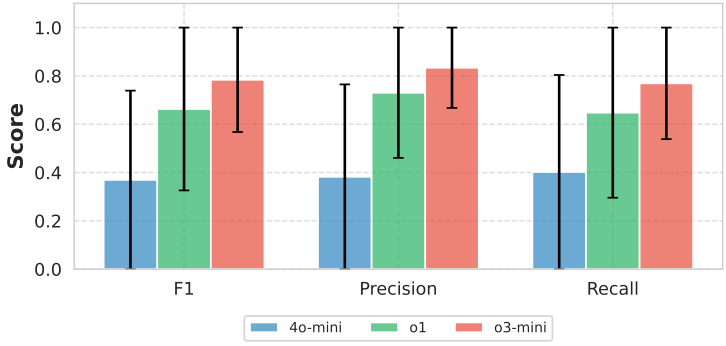


Figure 15 Comparison of F1 score, Precision, and Recall for minimal diagnoses generation across different models. Each bar shows the mean value with standard deviation.

References

1 J-Ph Cassar and M Staroswiecki. A structural approach for the design of failure detection and identification systems. *IFAC Proceedings Volumes*, 30(6):841–846, 1997.

2 Marie-Odile Cordier, Philippe Dague, Michel Dumas, François Lévy, Jacky Montmain, Marcel Staroswiecki, and Louise Travé-Massuyes. A comparative analysis of ai and control theory approaches to model-based diagnosis. In *ECAI*, pages 136–140, 2000.

3 Akshay J Dave, Tat Nghia Nguyen, and Richard B Vilim. Integrating llms for explainable fault diagnosis in complex systems. *arXiv preprint arXiv:2402.06695*, 2024. doi:10.48550/arXiv.2402.06695.

4 Teresa Escobet, Anibal Bregon, Belarmino Pulido, and Vicenç Puig. *Fault Diagnosis of Dynamic Systems*. Springer, 2019.

5 Erik Frisk, Mattias Krysander, and Daniel Jung. A toolbox for analysis and design of model based diagnosis systems for large scale models. *IFAC-PapersOnLine*, 50(1):3287–3293, 2017.

6 Sungmin Kang, Gabin An, and Shin Yoo. A quantitative and qualitative evaluation of llm-based explainable fault localization. *Proceedings of the ACM on Software Engineering*, 1(FSE):1424–1446, 2024. doi:10.1145/3660771.

7 Abdullah Khan, Rahul Nahar, Hao Chen, Gonzalo E Flores, and Can Li. Faultexplainer: Leveraging large language models for interpretable fault detection and diagnosis. *arXiv preprint arXiv:2412.14492*, 2024.

8 Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(1):197–206, 2007. doi:10.1109/TSMCA.2007.909555.

9 Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 38(1):197–206, 2008. doi:10.1109/TSMCA.2007.909555.

10 Karol Kukla. jadlownik/FaultDiagnosis. Software, swhId: swh:1:dir:b8a40cc86347d60c8efb7ce10df9343400eee88e (visited on 2025-10-20). URL: https://github.com/jadlownik/FaultDiagnosis/tree/develop, doi:10.4230/artifacts.24965.

11 Lin Lin, Sihao Zhang, Song Fu, and Yikun Liu. Fd-llm: Large language model for fault diagnosis of complex equipment. *Advanced Engineering Informatics*, 65:103208, 2025. doi:10.1016/J.AEI.2025.103208.

12 Silke Merkelbach, Alexander Diedrich, Anna Sztyber-Betley, Louise Travé-Massuyès, Elodie Chanthery, Oliver Niggemann, and Roman Dumitrescu. Using multi-modal llms to create models for fault diagnosis. In *The 35th International Conference on Principles of Diagnosis and Resilient Systems (DX'24)*, volume 125, 2024.

13 Herbert Muehlburger and Franz Wotawa. Flex: Fault localization and explanation using open-source large language models in powertrain systems (short paper). In *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*, pages 25:1–25:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/OASIcs.DX.2024.25.

14 Herbert Mühlburger and Franz Wotawa. Faultlines-evaluating the efficacy of open-source large language models for fault detection in cyber-physical systems. In *2024 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pages 47–54. IEEE, 2024. doi:10.1109/AITEST62860.2024.00014.

15 CG Pérez-Zuniga, E Chanthery, L Travé-Massuyès, and J Sotomayor. Fault-driven structural diagnosis approach in a distributed context. *IFAC-PapersOnLine*, 50(1):14254–14259, 2017.

16 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

17 Alicia Russell-Gilbert, Alexander Sommers, Andrew Thompson, Logan Cummins, Sudip Mittal, Shahram Rahimi, Maria Seale, Joseph Jaboure, Thomas Arnold, and Joshua Church. Aad-llm: Adaptive anomaly detection using large language models. In *2024 IEEE International Conference on Big Data (BigData)*, pages 4194–4203. IEEE, 2024. doi:10.1109/BIGDATA62323.2024.10825679.

18 Anna Sztyber Betley. asztyber/LLM_diagnostic_concepts. Software, swhId: `swh:1:dir:d265fcfc25d623bff2fa34d48590c7eb66d08995` (visited on 2025-10-20). URL: `https://github.com/asztyber/LLM_diagnostic_concepts`, `doi:10.4230/artifacts.24966`.

19 Laifa Tao, Haifei Liu, Guoao Ning, Wenyan Cao, Bohao Huang, and Chen Lu. Llm-based framework for bearing fault diagnosis. *Mechanical Systems and Signal Processing*, 224:112127, 2025.

20 Louise Travé-Massuyes, Teresa Escobet, and Xavier Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 36(6):1146–1160, 2006. `doi:10.1109/TSMCA.2006.878984`.

21 Aidan ZH Yang, Claire Le Goues, Ruben Martins, and Vincent Hellendoorn. Large language models for test-free fault localization. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–12, 2024.

Combining Dynamic Slicing and Spectrum-Based Fault Localization – A First Experimental Evaluation

Jonas Schleich
Graz University of Technology, Austria

Franz Wotawa[1,2]
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Abstract

Identifying and localizing bugs in programs has always been considered a complex but essential topic. Whereas the former has led to substantial progress in areas like formal verification and testing with a high degree of automation, the latter has not been satisfactorily automated. Approaches like program slicing, model-based diagnosis, and, more recently, spectrum-based fault localization can be used to find possible causes of a misbehaving program automatically, but often come with high computational complexity or a larger list of diagnoses, which require additional manual effort. In this paper, we present the first experimental results of an approach that combines program slicing with spectrum-based fault localization aiming at improving the outcome of automated debugging methods. In contrast to previous work, where we illustrated potential improvements only by considering a particular use case, we present an evaluation based on 22 different example programs in this paper. The approach improves the wasted effort on average by around 5 to 15% on average.

2012 ACM Subject Classification Software and its engineering → Software testing and debugging; Computing methodologies → Causal reasoning and diagnostics

Keywords and phrases Software fault localization, program slicing, spectrum-based fault localization, automated debugging

Digital Object Identifier 10.4230/OASIcs.DX.2025.3

Supplementary Material *Software (Source Code)*: `https://github.com/SchleichJonas/JSR` archived at `swh:1:dir:062e5cd4325d9b47fafadb80f521129b36a5c8c6`

Funding *Franz Wotawa*: The work was supported by the Austrian Science Fund (FWF) Cluster of Excellence Bilateral AI under contract number 10.55776/COE12.

1 Introduction

Debugging comprises detecting, localization, and repairing faults in programs, which still is mainly carried out manually, causing a lot of effort. There are several approaches supporting the automation of fault detection in use, but almost none for fault localization and repair. However, automating debugging has been of interest for more than four decades, e.g., see Ehud Shapiro [21] or Mark Weiser [25, 26]. The latter investigated how programmers perform debugging utilizing program slices, which are calculated only considering the source code. Korel and Laski [16] introduced dynamic slicing, where program executions are used to eliminate part of the source code not involved in current executions. However, despite the interest in slicing from academia, its use in practice is limited, leading to other debugging approaches like spectrum-based fault localization or model-based debugging [6, 10, 31].

[1] Corresponding author
[2] Authors are listed in alphabetical order.

Regarding spectrum-based fault localization (SFL), Jones and Harrold [13] introduced the basic concepts in the corresponding tool Tarantula. In contrast to slicing, which utilizes program dependencies, SFL uses a probability-based approach considering different execution runs. In particular, the idea is to assign a suspicious value to each statement (or other part of a program), which can be motivated as follows. Any statement that is not executed in failing runs is very unlikely to be faulty. A statement that is only executed in failing runs is likely faulty. All other statements executed in passing or failing runs might be faulty. A suspicious value for each statement can be computed considering the execution of statements.

It is worth noting that SFL has gained much attention in the debugging community. Wong and colleagues [28] provided a survey showing that SFL holds the largest share of publications of about 30%, followed by slicing-based debugging approaches. However, there is only a little work combining slicing and SFL for improving debugging, i.e., Wen et al. [27], Hofer and Wotawa [11], Reis et al. [18], Soha [22], and more recently Wotawa [30]. In this paper, we mainly focus on the last cited publication and provide an initial experimental evaluation of the impact of the combined approach on the quality of the debugging results. In his original paper, Wotawa [30] only discussed the potential impact but did not provide any experimental evidence of the superiority of the combined slicing and SFL approach. It is further worth noting that combining approach is one possibility to bring different approaches together, which often leads to a better performance, e.g., see Mukhtar et al. [17]. The other way is to show that one approach can be subsumed by the other. For example, Wotawa [29] showed that the results of static slicing can be achieved using an abstract model of program statements considering only dependencies between variables and statements.

Hence, in this paper, we contribute to automated software debugging as follows. We present a first experimental evaluation that considers a combined debugging approach utilizing SFL and dynamic slicing. The experimental evaluation was carried out considering available tools for Java programs. The research objective of the study was to clarify whether the combined debugging approach improves debugging or not. The question is important because adding slicing to debugging comes with a substantial computational overhead compared to SFL. Hence, without substantial improvements, a real benefit of the combined method may not be arguable. In addition to the study, we also discuss challenges and issues we experienced when carrying out the experimental evaluation. In particular, there is a huge influence of tools on the outcome, which cannot be neglected and where only partial mitigation is possible. Note that the provided experimental evaluation cannot be considered an exhaustive one incorporating a vast number of faulty versions of different programs. It is the first study to clarify whether conducting additional experimental evaluation is worthwhile.

We organize the paper as follows: We first introduce the foundations behind SFL, dynamic slicing, and the combined approach. Afterward, we describe the experimental setup, followed by a detailed presentation of the results. We discuss and summarize the obtained results and finally conclude this paper.

2 Foundations

In this section, we discuss the foundations behind SFL and its variant that utilizes dynamic slicing. To illustrate definitions and underlying ideas, we make use of an illustrative example program. In Figure 1, we have a simple Java program `Car.warn` that computes the braking distance using the current velocity, the friction coefficient `mu`, and the physical equation given in Line 12. This braking distance is used together with a given distance to a vehicle in front of a car to raise a warning in the case where we cannot stop within 80% of the provided

```
1. public class Car {
2.     public static double mu = 1.10758097;
3.     public static double constG = 9.80665;
4.     public static void warn(
5.           double velocity, double distance, boolean raining) {
6.       double braking_distance;
7.       double current_mu = mu;
8.       boolean warning;
9.       if (raining) {
10.          current_mu = 0.8*current_mu;
11.      }
12.      braking_distance = (velocity*velocity)/(2*current_mu*constG);
13.      if (braking_distance > 0.8*distance) {
14.          warning = true;
15.      } else {
16.          warning = false;
17.      }
18.      store(warning,braking_distance);
19.    }
20. }
```

Figure 1 An illustrative example program computing an expected braking distance and comparing it with a distance to a vehicle in front for raising alarm messages.

Table 1 A test suite for program `Car.warn`.

test case	velocity	distance	raining	warning	braking_distance
T_1	15	10	false	true	10.357533856871605
T_2	30	64	false	false	41.430135427486420
T_3	30	64	true	true	51.787669284358020
T_4	15	64	true	false	12.946917321089504

distance. The 80% is a safety margin. In addition, a situation where we have rain influencing the friction and, therefore, the braking distance is considered. In the case of rain, the friction coefficient is set to 80% of its original value.

To test `Car.warn`, we need test cases. A test case comprises given input values and the resulting expected output values. In our case, we consider the warning and the calculated braking distance as output. Table 1 comprises 4 test cases for `Car.warn`. Via executing `Car.warn`, we are able to confirm the correct behavior of the program. Let us now assume that we have a bug in the program. Instead of Line 10, we have `current_mu = 0.85*current_mu`. We refer to this faulty variant as `Car.warn'`. Executing `Car.warn'` on the same test suite, however, leads to a different outcome, which we depict in Table 2. Obviously, the values of the braking distance are wrong whenever it is raining, but only once the warning is false instead of true.

After identifying deviations from expectations, we are interested in identifying the root cause behind them. This is similar to ordinary model-based diagnosis [19, 7] but considers a program instead of a system comprising interacting (hardware) components.

Table 2 Running the test suite from Table 1 on program `Car.warn'`. Values in **bold** face indicate differences to expectations.

test case	velocity	distance	raining	warning	braking_distance
T_1	15	10	false	true	10.357533856871605
T_2	30	64	false	false	41.430135427486420
T_3	30	64	true	**false**	**48.74133579704284**
T_4	15	64	true	false	**12.18533394926071**

Table 3 The program spectrum of `Car.warn'` considering the 4 test cases.

Statement	T_1	T_2	T_3	T_4	a_{00}	a_{01}	a_{10}	a_{11}	c_O
4. public static void warn(									
5. double velocity, double distance, boolean raining) {									
6. double braking_distance;	1	1	1	1	0	0	2	2	0.707
7. double current_mu = mu;	1	1	1	1	0	0	2	2	0.707
8. boolean warning;	1	1	1	1	0	0	2	2	0.707
9. if (raining) {	1	1	1	1	0	0	2	2	0.707
10. **current_mu = 0.85*current_mu;**	0	0	1	1	2	0	0	2	1.000
11. }									
12. braking_distance = (velocity*velocity)/...;	1	1	1	1	0	0	2	2	0.707
13. if (braking_distance > 0.8*distance) {	1	1	1	1	0	0	2	2	0.707
14. warning = true;	1	0	0	0	1	2	1	0	0.000
15. } else {									
16. warning = false;	1	0	1	1	1	0	1	2	0.816
17. }									
18. store(warning,braking_distance);	1	1	1	1	0	0	2	2	0.707
19. }									
Error vector	0	0	1	1					

2.1 Spectrum-Based Fault Localization

We first explain the ideas behind SFL [13, 2, 3, 1]. SFL utilizes the program spectrum for debugging. A program spectrum is a matrix considering the program statements on one axis and the test cases on the other. A cell of the matrix has a value of 1 if the statement is executed in its corresponding test case and 0, otherwise. To access an element of the program spectrum in row i and column j, we write x_{ij}. In addition, we have an error vector e_j indicating whether a test case is passing or failing for each column j. A passing test case is a test case where the program delivers the expected outcome. Otherwise, a test case is said to be a failing one. Table 3 shows the program spectrum and the error vector for our illustrative example program `Car.warn'`.

To compute a suspicious value for each statement, we need information on whether a statement is executed in a passing or failing run. SFL introduces for this purpose 4 metrics values a_{ij} for each statement, i.e., row i, which are defined as follows: $a_{nm}(i) = |\{j | x_{ij} = n \wedge e_j = m\}|$. Hence, each a_{nm} indicates whether a statement is executed or not in a passing or failing test case. We find the metrics information for `Car.warn'` in Table 3. What is missing is the computation of a suspicious value. In SFL, the four metrics values are used. There are a lot of papers introducing the computation of different suspicious values, which are also called SFL coefficients, that lead to a ranking of statements. The one with the highest value is the most suspicious, followed by the next value, etc.

In our experiments, we used three different SFL coefficients, i.e., Tarantula [13] c_T (Equation 1), Ochiai [2] c_O (Equation 2), and Sarhan-Beszedes [20] c_S (Equation 3):

$$c_T = \frac{\frac{a_{11}}{a_{11}+a_{01}}}{\frac{a_{10}}{a_{10}+a_{01}} + \frac{a_{11}}{a_{11}+a_{01}}} \tag{1}$$

$$c_O = \frac{a_{11}}{\sqrt{(a_{11}+a_{01}) \cdot (a_{11}+a_{10})}} \tag{2}$$

$$c_S = a_{11} + \left(\frac{a_{11} - a_{01}}{a_{11} + a_{01} + a_{10}} \right) \tag{3}$$

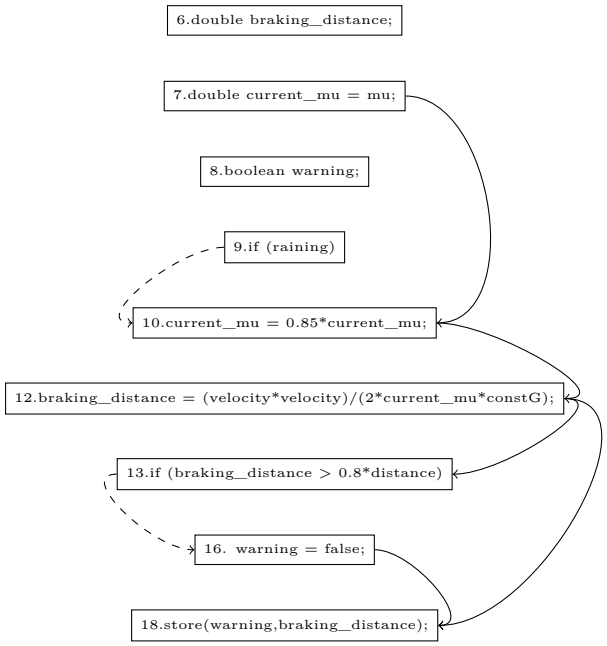
For our running example, Table 3 shows the results when applying the Ochiai coefficient. We see that the statement comprising the fault, i.e., Line 10, is the most suspicious with a c_O value of 1.0, followed by Line 16. Hence, in this case, SFL would enable a programmer to localize the fault in one step, only having a look at the statement in Line 10.

2.2 Dynamic Slicing

Dynamic slicing [16] is different compared to SFL as it utilizes the data and control dependencies of a program considering statements that are executed for a particular test case. Dynamic slicing only uses one test case at a time for extracting statements that cause a value of a given variable at a certain position of the execution. A particular execution can be seen as a trace, i.e., the execution trace. Together with all dependencies, we obtain a directed graph, i.e., the execution trace graph. Let us illustrate this using our `Car.warn'` method and test case T_3. When calling `warn(30,26,true)` the following statements are executed ignoring syntax details:

```
6.      double braking_distance;
7.      double current_mu = mu;
8.      boolean warning;
9.      if (raining) {
10.         current_mu = 0.85*current_mu;
12.     braking_distance = (velocity*velocity)/(2*current_mu*constG);
13.     if (braking_distance > 0.8*distance) {
16.         warning = false;
18.     store(warning,braking_distance);
```

We now add information regarding data and control dependencies and obtain the following graph:



If we now want to compute a dynamic slice for a variable, e.g., `braking_distance`, we only need to mark the node from the end of the graph where the variable is defined for the last time. For `braking_distance` this is Line 12. Afterward, we traverse the graph backward and mark all nodes we can reach. For `braking_distance` we obtain lines 10, 9, and 7. The final dynamic slice comprises all marked nodes, which are 7, 9, 10, and 12 for `braking_distance`. For `warning`, the dynamic slice is: 7, 9, 10, 12, 13, 16. Considering a single fault assumption, we may only want to focus on the intersection of both slices, which finally is the slice for `braking_distance`. Note that in dynamic slicing, we cannot rank statements in the slice. We further do not incorporate knowledge about failing or passing runs. Dynamic slicing is mainly used for failing test cases to identify root causes for misbehavior. However, due to using dependency information, dynamic slicing allows distinguishing statements in a block that would not be possible when only considering SFL. Therefore, a combination of the approaches seems to be a good idea to compensate for the weaknesses of the respective other approach.

2.3 Dynamic slicing enhanced spectrum-based fault localization

To be self-contained, we briefly explain the idea and algorithm of the combined approach. For more details, we refer to the original publication of Wotawa [30]. The idea behind combining slicing and SFL is simple. Instead of considering statement executions and solely the outcome of a test, we distinguish all output variables and consider the dynamic slices for them. When distinguishing the outputs of a test, we can also add the information of whether an output is correct or incorrect directly into SFL. Moreover, statements that are not in any dynamic slice can be ignored. The computation of the coefficients is then working as usual. Hence, it is only the way a test is separated into parts that distinguish the hybrid method from ordinary SFL.

In Figure 4, we depict the outcome of the hybrid method for our running example program `Car.warn'`. We see that the buggy statement is still the highest-ranked one. It is worth noting that, for this example, the hybrid method does not provide an improvement. However, for other example programs, there are improvements and, therefore, we are interested in a more detailed evaluation considering different faults and programs.

Table 4 The program spectrum of `Car.warn'` considering the 4 test cases and the hybrid approach. Note that the columns named `B` and `W` are the ones for variables `braking_distance` and `warning` respectively. Further, note that values are only computed for statements appearing in slices.

Statement	T_1		T_2		T_3		T_4		a_{00}	a_{01}	a_{10}	a_{11}	c_O
	W	B	W	B	W	B	W	B					
4. public static void warn(													
5. double velocity, ...) {													
6. double braking_distance;													
7. double current_mu = mu;	1	1	1	1	1	1	1	1	0	0	5	3	0.612
8. boolean warning;													
9. if (raining) {	1	1	1	1	1	1	1	1	0	0	5	3	0.612
10. **current_mu = 0.85*cur...**	0	0	0	0	1	1	1	1	4	0	1	3	0.866
11. }													
12. braking_distance = (velocity...;	1	1	1	1	1	1	1	1	0	0	5	3	0.612
13. if (braking_distance > 0.8*...) {	1	1	1	1	1	1	1	1	0	0	5	3	0.612
14. warning = true;	1	0	0	0	0	0	0	0	4	3	1	0	0.000
15. } else {													
16. warning = false;	0	0	1	0	1	0	1	0	3	2	2	1	0.333
17. }													
18. store(warning,braking_distance);													
19. }													
Error vector	0	0	0	0	1	1	0	1					

2.3.1 Implementation

We implemented the hybrid SFL approach utilizing the existing JSR framework that computes the checked coverage of Java programs [23, 15, 14]. Checked coverage is a coverage measure that considers whether statements influence the values of a given test property, which are used, for example, in JUnit as test oracles. The JSR framework utilizes the Java code coverage library JaCoCo [9] and the dynamic slicing tool Slicer4J [4] for computing checked coverage.

Using JSR, we obtain the spectrum for a program and a test suite as follows: For each test and output variable, we generate a JUnit test that comprises a test property for the variable. We run JSR on the test suite and compute the checked coverage for all test properties, which means to compute all statements that influence the variable value and are executed. This information is added to the program spectrum. For the error vector, we record whether the test property indicates a passing or a failing run and finally obtain all the needed information.

It is worth noting that we made some changes to JSR and also implemented the computation of the already introduced SFL coefficients, i.e., Ochiai, Tarantula, and Sarhan-Beszedes. The implementation is available on GitHub `https://github.com/SchleichJonas/JSR`. Besides the implementation, we put all programs for obtaining the experimental evaluation into the repository to ensure reproducibility.

3 Experimental evaluation

The objective of the experimental evaluation is to show the impact of adding dynamic slicing to spectrum-based fault localization. In particular, we are interested in answering the question of whether utilizing dynamic slicing improves the outcome of spectrum-based

Table 5 Programs used for the initial experiments.

Program	LoC	Test cases	Inputs	Outputs
BMI	24	6	2	1
Expint	88	5	2	1
Fisher	74	5	3	1
Gammq	91	5	2	1
Luhn	91	7	2	1
Middle	27	5	3	1
Tcas	152	8	12	1

fault localization when applied to ordinary programs. To answer this question, we first need to define what "improves" means in this context. From the literature, there have been several metrics specified for comparing different debugging methods, e.g., Hit Ratio@k and the wasted effort [12]. Whereas the former indicates how often a debugging approach ranks the correct diagnosis within the first n elements, the latter captures how many non-faulty statements are ranked before (or have to be inspected before) the buggy statement. The wasted effort is an absolute evaluation measure and not a percentage rank. Note that wasted effort is also sometimes referred to as EXAM score and is considered favorable to assess software debugging techniques [5]. In the context of this paper, we use the following definition of wasted effort (WE) where s_f is the suspicious score of the actual error:

$$WE = |\text{lines with score} > s_f| + 0.5 \cdot (|\text{lines with score} = s_f|) - 1)$$

Note that the hit ratio given in this paper is always the best, i.e., if there are multiple statements having the same SFL coefficient value, we assume that the statement comprising the fault is ranked the best.

For the experimental evaluation, we used different sets of programs. We introduced bugs and test suites that are able to capture the bug. For every program, we only introduced one fault manually. We ran the different debugging approaches and obtained the hit ratio and wasted effort for each program. We finally computed average values. In the following, we report the obtained results, considering each of the program sets separately. We start with the initial experiments.

3.1 Initial experiments

We carried out the initial experiments considering simple programs such as a BMI calculator, the Luhn algorithm that is used for credit card number verification, TCAS, and others, for which we summarize some metrics information in Table 5. These programs have been used in the context of software testing and also debugging research. The calculations vary in complexity from elementary calculations to more complex ones. In each class, multiple intentional errors were added manually, triggering SFL and dynamic slicing. The introduced errors differ, aiming at achieving as much variety as possible. Every calculation has multiple test cases, including passing and failing test cases. Table 6 depicts the applied changes for obtaining the programs used for the initial experiments.

For each program, we manually developed 5–8 test cases, ensuring that some are passing and some are failing. For the initial evaluation, we only considered one variable as the output variable. Hence, we did not expect any negative impact of the hybrid method on the SFL result. There might be a positive impact on the results because the hybrid SFL method utilizes data dependencies, which might allow the elimination of statements that do not contribute to the final outcome and, therefore, improve the ranking.

Table 6 Alteration of the source code used in the initial experiments.

Program	LineNr.	Original code	Altered code
BMI	17	else if (bmi < 30)	else if (bmi > 30)
BMI2	7	bmi_score = weight / (height * height);	bmi_score = weight / (height);
BMI3	12	calculateBMI(height, weight);	calculateBMI(weight, height);
Expint	43	h *= del;	h = del;
Expint2	47	return h*Math.exp(-x);	return h*Math.exp(x);
Expint3	38	a = -i*(nm1+i);	a = -i*nm1+i;
Fisher4	10	a = 2*(m/2)-m+2;	a = 2*(m/2)-n+2;
Fisher2	27	d = 0.5*p*z/w;	d = 0.4*p*z/w;
Fisher3	54	p = p*zk+w*z*(zk-1.0)/(z-1.0);	p = p*zk+w*(zk-1.0)/(z-1.0);
Gammq	37	an = -i*(i-a);	an = i*(i-a);
Gammq2	72	gamser=sum*Math.exp(-x+a*Math.log(x)-gln);	gamser=sum*Math.exp(-x+a*Math.log(x));
Gammq3	84	return 1-gamser;	return gamser-1;
Luhn	14	if (number.length() != 16 ||	if (number.length() != 15 ||
Luhn2	76	for (int i = number.length-2; i > -1; i-=2) {	for (int i = number.length-2; i > 0; i-=2) {
Middle	6	if((a<b && b<c)|| (c<b && b<a)){	if((a<b && b<c)|| (c<b && b>a)){
Middle2	15	return a;	return b;
Tcas	54	return ((Climb_Inhibit!=0) ? Up_Separation + NOZCROSS : Up_Separation);	return ((Climb_Inhibit!=0) ? Up_Separation - NOZCROSS : |Up_Separation);
Tcas2	124	alt_sep = UPWARD_RA;	alt_sep = UNRESOLVED;

In tables 7 and 8, we depict the obtained results of the initial experiments considering SFL alone and the hybrid approach, respectively. We see that there are differences both in the rank and the wasted effort (WE). In some cases, WE improves when using the hybrid approach, and in three cases, the WE becomes worse. This holds especially for the Expint and the Tcas programs, where we obtained severe declines in performance, which is also visible in the following table, which captures the Hit ratio and the average values of the WE results for the three SFL coefficients:

	Ochiai			Tarantula			Sarhan-Beszedes		
	Hit@1	**Hit@5**	**WE**	**Hit@1**	**Hit@5**	**WE**	**Hit@1**	**Hit@5**	**WE**
SFL	0.8333	1.0000	4.8056	0.7222	1.0000	5.8611	0.8333	1.0000	4.8056
Hybrid	0.7778	0.9444	6.4722	0.7222	0.9444	7.6389	0.7778	0.9444	6.4722

For all three SFL coefficients, we see a decline of the WE when using the hybrid approach, which is unexpected. Therefore, we further investigated the underlying reasons.

After having a look at the usual suspects for causes of the unexpected results, like the implementation or the experimental setup, which seemed to be correct, we further investigated the dynamic slicer used in the JSR framework. A detailed analysis of the resulting traces and slices showed that sometimes the slicer does not give back statements comprising faults. For both programs Expint and Tcas we obtained a similar outcome. The slicer uses Jimple, which is a 3-address intermediate representation, to simplify analysis and transformation of Java bytecode [24] as the internal representation of the program. Unfortunately, this transformation removes important parts of the program, not allowing the slicer to return a correct output in every case. It is worth noting that dynamic slicers themselves do not always deliver back a correct slice, which leads to the development of critical slicing [8] and other variants.

Table 7 Results of the initial experiments only considering SFL without slicing. WE stands for wasted effort.

Program	Ochiai		Tarantula		Sarhan-Beszedes	
	Rank	WE	Rank	WE	Rank	WE
BMI	1	0.0	1	1.5	1	0.0
BMI2	1	0.5	3	5.0	1	0.5
BMI3	1	1.0	1	1.0	1	1.0
Expint	1	9.0	1	9.0	1	9.0
Expint2	1	9.0	1	9.0	1	9.0
Expint3	1	9.0	1	9.0	1	9.0
Fisher	1	8.0	1	19.0	1	8.0
Fisher2	1	0.5	1	0.5	1	0.5
Fisher3	2	4.5	2	4.5	2	4.5
Gammq	1	10.0	1	10.0	1	10.0
Gammq2	1	6.0	1	6.0	1	6.0
Gammq3	1	6.0	1	6.0	1	6.0
Luhn	1	0.0	2	2.0	1	0.0
Luhn2	1	6.5	1	6.5	1	6.5
Middle	2	1.5	2	1.5	2	1.5
Middle2	1	0.0	1	0.0	1	0.0
Tcas	4	15.0	4	15.0	4	15.0
Tcas2	1	0.0	1	0.0	1	0.0

Table 8 Results of the initial experiments considering the hybrid approach with dynamic slicing. An arrow up indicates an improvement of the result compared to the one given in Table 7, and an arrow down the opposite. WE stands for wasted effort.

Program	Ochiai		Tarantula		Sarhan-Beszedes	
	Rank	WE	Rank	WE	Rank	WE
BMI	1	0.0	1	1.5	1	0.0
BMI2	1	0.0↑	3	4.5↑	1	0.0↑
BMI3	1	1.0	1	1.0	1	1.0
Expint	2↓	26.0↓	2↓	26.0↓	2↓	26.0↓
Expint2	1	7.5↑	1	7.5↑	1	7.5↑
Expint3	1	7.5↑	1	7.5↑	1	7.5↑
Fisher	1	4.5↑	1	19.5↓	1	4.5↑
Fisher2	1	0.5	1	0.5	1	0.5
Fisher3	2	4.5	2	4.5	2	4.5
Gammq	1	8.0↑	1	8.0↑	1	8.0↑
Gammq2	1	4.5↑	1	4.5↑	1	4.5↑
Gammq3	1	5.0↑	1	5.0↑	1	5.0↑
Luhn	1	0.0	1↑	0.0↑	1	0.0
Luhn2	1	6.5	1	6.5	1	6.5
Middle	2	5.5↓	2	5.5↓	2	5.5↓
Middle2	1	0.0	1	0.0	1	0.0
Tcas	6↓	35.5↓	6↓	35.5↓	6↓	35.5↓
Tcas2	1	0.0	1	0.0	1	0.0

Table 9 Programs used for second experiments.

Program	LoC	Test cases	Inputs	Outputs
Armstrong	49	7	1	1
Bubblesort	38	7	1	1
ChineseRemainder	47	7	2	1
Factorial	18	7	1	1
GCD	27	7	2	1
InverseCounter	26	7	1	1
IsPrime	20	7	1	1
LCM	24	7	2	1
LogExp	23	7	2	1
Minimax	50	7	1	1
ModInverse	32	7	2	1
Mult	30	7	2	1
RSA	58	7	3	1
RussianPeasant	34	7	2	1
Sqrt	23	7	1	1

Hence, we can summarize the findings obtained from the initial experiment as follows:

Finding 1: There is a huge impact of the underlying slicer on the obtained results of the hybrid method in some cases leading to worst results when using the hybrid SFL approach.

More experiments are required to further investigate the effect of the impact of the underlying slicer and on other influencing factors like the way an error was introduced in a program. Therefore, we carried out a second and a third experiment.

3.2 Second experiments

The objectives of the second experiment are to clarify whether there is an influence on the way intentional errors are introduced, covering well-known algorithm implementations. For this purpose, we selected 15 algorithms, from very simple ones like the greatest common divisor (GCD) to more complex ones like RSA encryption and decryption. See Table 9 for the list of implementations and their corresponding statistics. We provided the implementations to four students and asked them to place one error into the source code, not giving them a lot of input to avoid introducing bias. Note that the implementations were equally distributed among the students. The only requirement was that the error still lead to the execution of the program without any exceptions. The selected students come from different fields; where two are from software engineering, one from mathematics, and another one is not from any science discipline at all. Table 10 depicts the changes to the implementations introduced by the students.

After running the second experiment, we obtained the debugging outcome of SFL and the hybrid method that we depict in tables 11 and 12, respectively. The debugging methods performed better in this experiment than the initial one. This is probably because the programs of the second experimental evaluation are simpler. The improvements can be observed both for the Tarantula coefficient and the hybrid SFL method. However, Tarantula still performed worse than the other SFL coefficient. The already well-performing Sarhan-Beszedes coefficient performed even better than Ochiai, which performed the same as Sarhan-Beszedes for the initial setup.

Table 10 Changed code of programs used for the second experiments.

Program	LineNr.	Original code	Altered code
Armstrong	34	int r = temp % 10;	int r = temp / 10;
Bubblesort	36	return arr;	return res;
ChineseRemainder	26	if (x%num[j] != rem[j])	if (x%num[j] == rem[j])
Factorial	7	int res = 1;	int res = 0;
GCD	12	if (a % result == 0 && b % result == 0) {	if (a / result == 0 && b % result == 0) {
InverseCounter	12	for (int j = i + 1; j < n; j++) {	for (int j = i + 1; j < n - 1; j++) {
IsPrime	10	for (int i = 2; i < n; i++)	for (int i = 2; i <= n; i++)
LCM	10	return gcd(b % a, a);	return gcd(b % a, b);
LogExp	12	pow = pow * b;	pow = pow % b;
Minimax	48	return minimax(0, 0, true, scores, h);	return -minimax(0, 0, true, scores, h);
ModInverse	30	return modInverse(A, M);	return modInverse(M, M);
Mult	17	return -multiply(x, -y);	return multiply(x, -y);
RSA	14	z = (p - 1) * (q - 1);	z = (p - 1) * (q + 1);
RussianPeasant	19	res = res + a;	res = res % a;
Sqrt	17	return res - 1;	return res - res;

Table 11 Results of the second experiments only considering SFL without slicing. WE stands for wasted effort.

Program	Ochiai		Tarantula		Sarhan-Beszedes	
	Rank	WE	Rank	WE	Rank	WE
Armstrong	1	4.5	1	4.5	1	4.5
BubbleSort	1	3.0	1	7.5	1	3.0
ChineseRemainder	1	3.5	1	5.0	1	3.5
Factorial	1	2.0	1	2.5	1	2.0
GCD	1	0.5	1	0.5	1	0.5
InverseCounter	2	4.5	2	4.5	1	3.5
Isprime	1	1.0	1	1.0	1	1.0
LCM	1	0.0	1	0.0	1	0.0
LogExp	1	0.0	1	0.0	1	0.0
Minimax	1	3.0	1	3.0	1	3.0
ModInverse	1	3.0	1	5.0	1	3.0
Mult	1	0.5	1	0.5	1	0.5
RSA	1	12.0	1	12.0	1	12.0
RussianPeasant	1	4.0	1	4.0	1	4.0
Sqrt	2	3.0	2	3.0	2	3.0

Table 12 Results of the second experiment considering the hybrid approach with dynamic slicing. An arrow up indicates an improvement of the result compared to the one given in Table 11, and an arrow down the opposite. WE stands for wasted effort.

Program	Ochiai		Tarantula		Sarhan-Beszedes	
	Rank	WE	Rank	WE	Rank	WE
Armstrong	1	5.0↓	2↓	7.0↓	1	5.0↓
BubbleSort	1	0.5↑	1	7.5	1	0.5↑
ChineseRemainder	3↓	7.0↓	1	5.0	3↓	7.0↓
Factorial	1	1.0↑	1	2.0↑	1	1.0↑
GCD	1	1.0↓	1	1.0↓	1	1.0↓
InverseCounter	1↑	2.0↑	1↑	2.0↑	2↓	5.0↓
Isprime	1	1.0	1	1.0	1	1.0
LCM	1	0.0	1	0.0	1	0.0
LogExp	1	0.0	1	0.0	1	0.0
Minimax	1	2.0↑	2↓	4.0↓	1	2.0↑
ModInverse	1	2.5↑	1	4.5↑	1	2.5↑
Mult	1	0.5	1	0.5	1	0.5
RSA	1	9.5↑	1	9.5↑	1	9.5↑
RussianPeasant	1	3.5↑	1	3.5↑	1	3.5↑
Sqrt	2	3.0	2	3.0	2	3.0

These improvements are also very well visible in the summary of the results, where we also consider the hit ratio and not only wasted effort, which is given in the following table:

	Ochiai			Tarantula			Sarhan-Beszedes		
	Hit@1	**Hit@5**	**WE**	**Hit@1**	**Hit@5**	**WE**	**Hit@1**	**Hit@5**	**WE**
SFL	0.8667	1.0000	2.9667	0.8667	1.0000	3.5333	0.9333	1.0000	2.9000
Hybrid	0.8667	1.0000	2.5667	0.8000	1.0000	3.3667	0.8000	1.0000	2.7667

Interestingly, while HitRatio@1 decreased with the hybrid method for Tarantula and Sarhan-Beszedes, the wasted effort improved slightly for both. For Ochiai, we see no change in the hit ratio, but the wasted effort also improved. Hence, what we can conclude is the following:

Finding 2: The hybrid method for SFL behaves slightly better considering the wasted effort for simple algorithm implementations, whereas the hit ratio is slightly worse.

Hence, we see a different outcome in the experiments where all underlying programs have only one output. Therefore, we are interested in investigating the effect of considering programs with more than one output on the debugging results. To answer this question, we carried out a third experimental evaluation.

3.3 Third experiments

To answer the question of whether the number of output variables impacts the ranking when comparing SFL with our hybrid method, we used the same example programs as in the first experiments, which are depicted in Table 5. But for most of the programs of the initial experiments, we used available variables in the source code as additional outputs for which we defined the expected values. For the programs and their variants, we have the following number of outputs in the third experimental setup: BMI (2), Expint (2), Fisher (4), Gammq (4), Luhn (1), Middle (1), Tcas (5). Note also that the increase of outputs (with

Table 13 Results of the third experiments only considering SFL without slicing. WE stands for wasted effort.

Program	Ochiai		Tarantula		Sarhan-Beszedes	
	Rank	WE	Rank	WE	Rank	WE
BMI	1	0.0	1	1.5	1	0.0
BMI2	1	2.5	4	7.5	1	2.5
BMI3	1	3.0	1	3.0	1	3.0
Expint	1	7.5	1	7.5	1	7.5
Expint2	1	7.5	1	7.5	1	7.5
Expint3	1	7.5	1	7.5	1	7.5
Fisher	1	8.0	1	13.5	1	8.0
Fisher2	1	0.5	1	0.5	1	0.5
Fisher3	2	4.5	2	4.5	2	4.5
Gammq	1	10.0	1	10.0	1	10.0
Gammq2	1	6.0	1	6.0	1	6.0
Gammq3	1	6.0	1	6.0	1	6.0
Luhn	1	0.0	2	2.0	1	0.0
Luhn2	1	6.5	1	6.5	1	6.5
Middle	2	1.5	2	1.5	2	1.5
Middle2	1	0.0	1	0.0	1	0.0
Tcas	4	16.5	4	16.5	2	11.5
Tcas2	1	0.0	1	0.0	1	0.0

the exception of Luhn and Middle) also leads to an increase in test cases because we have one JUnit test for each property specifying the expected behavior of one output. Hence, the results for SFL also might change, which is well visible when comparing Table 7 with Table 13.

When comparing the results of SFL and the hybrid approach, which are depicted in Table 13 and Table 14 respectively, we see an improvement. The hybrid approach provides a better wasted effort for almost all programs. Moreover, the ranking improved as well and is now the same or better for most of the coefficients. Considering more outputs even compensates for the problem with the slicer. In the following table, we summarize the findings for all programs, also considering the hit ratio:

	Ochiai			Tarantula			Sarhan-Beszedes		
	Hit@1	**Hit@5**	**WE**	**Hit@1**	**Hit@5**	**WE**	**Hit@1**	**Hit@5**	**WE**
SFL	0.8333	1.0000	4.8611	0.7222	1.0000	5.6389	0.8333	1.0000	4.5833
Hybrid	0.8333	1.0000	2.1667	0.7778	1.0000	3.1389	0.8333	1.0000	2.1389

We see that with one exception for Tarantula, the hit ratio of the hybrid approach is always the same as the one for SFL alone. The wasted effort improves between around 5% and 15% on average for the different SFL coefficients. Hence, the third experiment confirms that adding slicing has the potential to improve the outcome, at least for the wasted effort, which is an important measure for the efficiency of debugging. This leads to the following third finding of our preliminary experimental evaluation:

Finding 3: For programs with more than one output, the hybrid approach, which integrates dynamic slicing into SFL, the wasted effort improves.

Table 14 Results of the third experiments considering the hybrid approach with dynamic slicing. An arrow up indicates an improvement of the result compared to the one given in Table 13, and an arrow down the opposite. WE stands for wasted effort.

Program	Ochiai		Tarantula		Sarhan-Beszedes	
	Rank	WE	Rank	WE	Rank	WE
BMI	1	0.0	1	1.5	1	0.0
BMI2	1	0.5↑	2↑	1.5↑	1	0.5↑
BMI3	1	0.5↑	1	0.5↑	1	0.5↑
Expint	1	0.5↑	1	0.5↑	1	0.5↑
Expint2	1	1.0↑	1	1.0↑	1	1.0↑
Expint3	1	5.5↑	1	7.0↑	1	5.5↑
Fisher	1	0.0↑	1	12.0↑	1	0.0↑
Fisher2	1	0.5	1	0.5	1	0.5
Fisher3	2	4.0↑	2	4.5	2	3.5↑
Gammq	1	6.5↑	1	7.0↑	1	6.5↑
Gammq2	1	3.5↑	1	4.0↑	1	3.5↑
Gammq3	1	0.0↑	1	0.0↑	1	0.0↑
Luhn	1	0.0	1↑	0.0↑	1	0.0
Luhn2	1	6.5	1	6.5	1	6.5
Middle	2	5.5↓	2	5.5↓	2	5.5↓
Middle2	1	0.0	1	0.0	1	0.0
Tcas	4	4.0↑	3↑	4.0↑	4↓	4.0↑
Tcas2	1	0.5↓	1	0.5↓	1	0.5↓

3.4 Discussion

The experimental evaluation's objective was to carry out experiments showing whether combining dynamic slicing with SFL has a positive impact on debugging, i.e., a better hit ratio or wasted effort. From the experiments, which also considered different SFL coefficients, we see not a big difference except when considering multiple output variables, which was somehow expected. What was not expected to see a slight decrease in the debugging efficiency of the hybrid method for some examples? A detailed analysis reveals that the slicer causes trouble in some cases. Hence, a result of this evaluation is that the slicer may have a huge impact depending on the program we want to debug.

As already mentioned, this experimental evaluation is an initial study. It uses small and at least partially simple example programs. Hence, the outcome may vary when using larger and more complex programs. However, the programs implement well-known algorithms and comprise ordinary data and control structures. They have not been selected for a particular purpose but used because they have already served as examples of different studies. Besides the selection of the programs, there are other threats to the validity. First, the implementation makes use of available tools and frameworks, which might be buggy, causing a bias. Second, we introduced faults manually. However, we did not introduce a fault having a certain result in mind. Third, the number of programs and their faulty variants are limited. The same holds for the test suites, which have also been manually generated. Considering more variants and different test suites might change the outcome. Finally, we used only one programming language, i.e., Java, for providing examples. Hence, there might also be an influence on the outcome. Therefore, further studies have to be carried out, including replications of the experiments, to prove or disprove the obtained findings.

4 Conclusions

In this paper, we present a first experimental evaluation that compares ordinary spectrum-based fault localization with a variant that utilizes dynamic slicing to overcome some limitations, like handling data dependencies. The experimental evaluation relied on several smaller Java programs where we introduced faults. The evaluation revealed a huge impact of the implementation, and in particular, the use of a dynamic slicer, on the outcome. Furthermore, we showed that for programs having more output, the hybrid method performs better in terms of wasted effort. Hence, the outcome of the study is promising. Howewver, further experiments are required for a final judgment of the combined debugging methodology. This includes identifying the influence of test suites or the complexity of programs on the outcome. Moreover, more faulty variants and larger programs should be used for the evaluation. The latter is of particular interest to show whether additional computational complexity of the computationally more demanding hybrid approach really pays off, leading to a substantially improved debugging outcome. All these open issues we want to consider in our future research.

References

1 Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J. C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009. doi:10.1016/j.jss.2009.06.035.

2 Rui Abreu, Peter Zoeteweij, and Arjan J.C. van Gemund. On the accuracy of spectrum-based fault localization. In *Proceedings TAIC PART'07*, pages 89–98. IEEE, 2006.

3 Rui Abreu, Peter Zoeteweij, and Arjan J.C. van Gemund. Spectrum-based multiple fault localization. In *Proc. IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 88–99, 2009. doi:10.1109/ASE.2009.25.

4 Khaled Ahmed, Mieszko Lis, and Julia Rubin. Slicer4J: A Dynamic Slicer for Java. In *The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2021.

5 Aaron Ang, Alexandre Perez, Arie Van Deursen, and Rui Abreu. Revisiting the practical use of automated software fault localization techniques. In *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 175–182, 2017. doi:10.1109/ISSREW.2017.68.

6 Luca Console, Gerhard Friedrich, and Daniele Theseider Dupré. Model-based diagnosis meets error diagnosis in logic programs. In *Proceedings* 13^{th} *International Joint Conf. on Artificial Intelligence*, pages 1494–1499, Chambery, August 1993.

7 Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987. doi:10.1016/0004-3702(87)90063-4.

8 Richard A. DeMillo, Hsin Pan, and Eugene H. Spafford. Critical slicing for software fault localization. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 121–134, 1996. doi:10.1145/229000.226310.

9 EclEmma team. Jacoco java code coverage library. https://www.eclemma.org/jacoco/. Last accessed 20 January 2025. URL: https://www.eclemma.org/jacoco/.

10 Gerhard Friedrich, Markus Stumptner, and Franz Wotawa. Model-based diagnosis of hardware designs. *Artificial Intelligence*, 111(2):3–39, July 1999. doi:10.1016/S0004-3702(99)00034-X.

11 Birgit Gertraud Hofer and Franz Wotawa. Spectrum enhanced dynamic slicing for better fault localization. In Luc de Raedt, editor, *ECAI 2012 - 20th European Conference on Artificial Intelligence.*, volume 242 of *ECAI*, pages 420–425, Netherlands, 2012. IOS Press. doi:10.3233/978-1-61499-098-7-420.

12 Jiajun Jiang, Ran Wang, Yingfei Xiong, Xiangping Chen, and Lu Zhang. Combining spectrum-based fault localization and statistical debugging: An empirical study. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 502–514, 2019. doi:10.1109/ASE.2019.00054.

13 J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings ASE'05*, pages 273–282. ACM Press, 2005.

14 Roxane Koitz-Hristov, Thomas Sterner, Lukas Stracke, and Franz Wotawa. On the suitability of checked coverage and genetic parameter tuning in test suite reduction. *Journal of Software: Evolution and Process*, 36(8):e2656, 2024. doi:10.1002/smr.2656.

15 Roxane Koitz-Hristov, Lukas Stracke, and Franz Wotawa. Checked coverage for test suite reduction – is it worth the effort? In *2022 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 6–16, 2022. doi:10.1145/3524481.3527216.

16 Bogdan Korel and Janusz Laski. Dynamic Program Slicing. *Information Processing Letters*, 29:155–163, 1988. doi:10.1016/0020-0190(88)90054-3.

17 Adil Mukhtar, Birgit Hofer, Dietmar Jannach, Franz Wotawa, and Konstantin Schekotihin. Boosting spectrum-based fault localization for spreadsheets with product metrics in a learning approach. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*, pages 175:1–175:5. ACM, 2022. doi:10.1145/3551349.3559546.

18 Sofia Reis, Rui Abreu, and Marcelo d'Amorim. Demystifying the combination of dynamic slicing and spectrum-based fault localization. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4760–4766. International Joint Conferences on Artificial Intelligence Organization, July 2019. doi:10.24963/ijcai.2019/661.

19 Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

20 Qusay Idrees Sarhan and Arpad Beszedes. Experimental evaluation of a new ranking formula for spectrum based fault localization. In *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 276–280, 2022. doi:10.1109/SCAM55253.2022.00038.

21 Ehud Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, Massachusetts, 1983.

22 Péter Attila Soha. On the efficiency of combination of program slicing and spectrum-based fault localization. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 499–501, 2023. doi:10.1109/ICST57152.2023.00061.

23 Lukas Stracke. Jsr - the java test suite reduction framework. https://github.com/Lms24/JSR?tab=readme-ov-file#jsr---the-java-test-suite-reduction-framework. Last accessed 20 January 2025. URL: https://github.com/Lms24/JSR?tab=readme-ov-file#jsr---the-java-test-suite-reduction-framework.

24 Raja Vallée-Rai and Laurie J. Hendren. Jimple: Simplifying java bytecode for analyses and transformations, 1998. URL: https://api.semanticscholar.org/CorpusID:10529361.

25 Mark Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(7):446–452, July 1982. doi:10.1145/358557.358577.

26 Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984. doi:10.1109/TSE.1984.5010248.

27 Wanzhi Wen, Bixin Li, Xiaobing Sun, and Jiakai Li. Program slicing spectrum-based software fault localization. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Miami Beach, USA, 2011.

28 W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. Software Eng.*, 42(8):707–740, 2016. doi:10.1109/TSE.2016.2521368.

29 Franz Wotawa. On the Relationship between Model-Based Debugging and Program Slicing. *Artificial Intelligence*, 135(1–2):124–143, 2002. doi:10.1016/S0004-3702(01)00161-8.

30 Franz Wotawa. Surveying and generalizing methods for combining dynamic slicing with spectrum-based fault localization. In *Proceedings of the 34th International Workshop on Principles of Diagnosis (DX)*, Loma Mar, CA, USA, 2023. URL: `https://dx-2023.sai.tugraz.at/DX23_Wotawa.pdf`.

31 Franz Wotawa, Mihai Nica, and Iulia Moraru. Automated debugging based on a constraint model of the program and a test case. *J. Log. Algebraic Methods Program.*, 81(4):390–407, 2012. `doi:10.1016/j.jlap.2012.03.002`.

Using Qualitative Simulation Models for Monitoring and Diagnosis

Ankita Das[1]
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Roxane Koitz-Hristov
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Franz Wotawa[2]
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Abstract

Many systems in our daily lives control physical processes, which are parametrized and adapted, such as heating systems in buildings. Faults and non-optimized settings lead to a high energy demand and, therefore, need to be detected as early as possible. Unfortunately, due to specific adaptations, only the basic principles remain the same, but not the concrete implementations, making the use of techniques like machine learning difficult. Therefore, we suggest using abstract models that cover the basic behavior in a way that allows us to reuse the models in different installations. In particular, we discuss the application of qualitative simulation for fault detection and introduce a formal definition of conformance between the results of qualitative simulation and the monitored behavior. We discuss arising difficulties and provide a basis for further research and applications.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics; Computing methodologies → Spatial and physical reasoning; Computing methodologies → Modeling methodologies

Keywords and phrases Qualitative Simulation, Fault Detection, Model-based Diagnosis, Monitoring, Application

Digital Object Identifier 10.4230/OASIcs.DX.2025.4

Funding The work presented in this paper has been supported by the FFG project Artificial Intelligence for Smart Diagnosis in Building Automation (ALFA) under grant FO999914932.
Ankita Das: Supported by by the Christian-Doppler Forschungsgesellschaft (CDG) project AI-based Diagnosis for Energy Transition and the Circular Economy (AID4ETCE).

1 Introduction

In 2022, the International Energy Agency estimated the buildings' energy-related CO_2 emissions at around 27% of the total worldwide CO_2 emissions. To bring the emissions down, several actions have to be taken, including monitoring and diagnosis of heating and cooling systems that often operate in non-optimal operational spaces using far more energy than necessary. This is well visible when considering previous work (i) in the context of radiant ceiling cooling systems, where model predictive control can reduce energy consumption by up to 27% (see, e.g., [10]), or (ii) heat pumps where fault diagnosis can reduce energy loss when detected, diagnosed, and repaired early by about 40% [3]. As a consequence, there is a strong need for automated diagnosis of heating and cooling systems in buildings to reduce the overall CO_2 emissions.

[1] Authors are listed in alphabetical order.
[2] Corresponding author.

Unfortunately, monitoring and fault localization of such systems is complicated because each building is unique, comprising tailored heating and cooling facilities. When using machine learning, we need either to retrain every model for every building, which is expensive and time-consuming or to find a way to adapt trained models. This similarly holds for physical simulation models that would need to be parametrized for every building. However, the basic principles and their corresponding abstract representation of the behavior of each of the facilities are the same, which follow physical principles. Hence, when being able to represent the behavior of systems that follow physical principles in an abstract way, we can use this for at least detecting faults when monitoring the current behavior of a system.

In this paper, we tackle this challenge and suggest utilizing qualitative reasoning to provide an abstract model of a system that can be used for fault detection and, finally, localization. In particular, we discuss how to use and couple qualitative simulation [20] to ordinary monitoring systems. Rinner and Kuipers [24] already suggested the use of qualitative simulation in monitoring. In contrast, we formally define conformance between the outcome of qualitative simulation and the observations obtained from a system. This formal conformance relationship serves as the basis for detecting faults and may also be of use for fault localization, providing that the qualitative simulation models also capture faulty behavior. The idea is to compare the different qualitative states over time with the abstracted observations. This comparison requires not only abstraction but also specific enhancements. For example, we do not continuously observe the behavior over time but at certain time points. Hence, we might not observe reaching landmarks, which would raise false alarms. Therefore, we need to add reasonable qualitative states to the observations for comparison.

We organize this paper as follows: In Section 2, we discuss related work. Afterward, in Section 3, we introduce a simple tank example, which we use in the rest of our paper for illustration purposes. Section 4 introduces qualitative simulation to be self-contained and discusses conformance in detail. Finally, we conclude the paper.

2 Related work

Qualitative reasoning provides a powerful method for modeling dynamic systems under uncertainty by abstracting continuous variables into symbolic associations. Three formalisms form the foundation [26]: Qualitative Process Theory (QPT) [12], which models physical processes using causality and process activation; Qualitative Physics [15], which uses component-level modeling and constraint propagation to envision possible system behaviors; and Qualitative Simulation (QSIM) [20], which simulates all consistent qualitative trajectories from an initial state using a constraint-based model. QSIM represents system dynamics using Qualitative Differential Equations (QDEs), which abstract sets of Ordinary Differential Equations (ODEs). Starting from an initial qualitative state, QSIM generates a behavior tree by enumerating all possible successor states using a transition table of permissible qualitative changes, and then pruning inconsistent states based on qualitative constraints [21].

QSIM has been extended and applied in various diagnostic frameworks. Subramanian and Mooney [25] extended QSIM to model systems with multiple concurrent faults by associating fault-specific constraints and using constraint-based reasoning to isolate them. Similarly, the SEXTANT system [23] combines QSIM with the HS-DAG algorithm [14] to compute minimal diagnoses based on behavioral conflicts. Another example is Mimic [11], a semi-quantitative system that integrates QSIM-style simulation with model-based reasoning for process monitoring. Mimic identifies faults by detecting discrepancies between simulated

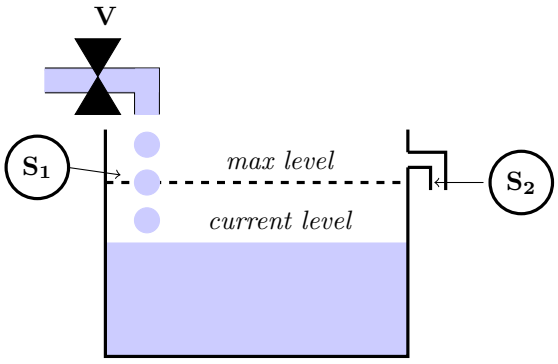


Figure 1 A simple tank example, comprising a input pipe with a valve, a sink, a sensor $\mathbf{S_1}$ measuring the maximum water level and a sensor $\mathbf{S_2}$ measuring whether water is passing the sink. The ordinary behavior is that the valve is open enabling an inflow until the maximum water level is reached and the valve is closed. The outward flow is only for cases where the inflow cannot be controlled anymore to prevent from flooding.

and observed behavior, and then tests fault hypotheses by adapting the model. Similarly, DIAMON [22] combines QSIM with consistency-based diagnosis in a layered monitoring framework. It uses qualitative simulation to detect abnormal behavior and incrementally refines the model abstraction level to isolate the root cause.

Beyond classical QSIM applications, efforts have been made to scale qualitative simulation to more complex hybrid systems. Klenk et al. [16, 17] propose translating a subset of Modelica models into qualitative constraints to enable QSIM-style simulation. The approach reduces spurious trajectories by incorporating continuity, higher-order derivatives, and landmark ordering.

QSIM has been successfully combined with frameworks based on constraints, including Constraint Logic Programming over Finite Domains (CLP(FD)) [2] and Answer Set Programming (ASP) [29]. Wiley et al. [29, 28] demonstrate that robots may autonomously learn qualitative action sequences, such as scaling obstacles or navigating uneven terrain, without requiring accurate quantitative representations of the robot or its environment when using their ASP approach to QSIM called ASP-QSIM.

Our approach builds directly on these QSIM-based diagnosis principles but introduces a conformance-based interpretation of system behavior for system monitoring and fault detection. By exploiting ASP-QSIM, our method identifies deviations between observed system trajectories and simulation-derived qualitative behaviors.

3 Illustrative example

In this section, we illustrate the suggested diagnosis approach considering a tank comprising a pipe where water flows in that has a valve for enabling water flow or stopping it, and a sink for preventing overflowing of the tank. We assume that the sink is designed such that even the amount of water coming from a fully opened input valve can be handled. The overall system has also two sensors. Sensor $\mathbf{S_1}$ is for indicating that the maximum water level is reached. In this case the valve will be closed. Otherwise, the valve is set to open. Another sensor $\mathbf{S_2}$ measures whether there is an outflow, which indicates a trouble. The second sensor can be seen as a form of safeguard for the outpipe. Figure 1 graphically depicts the tank example.

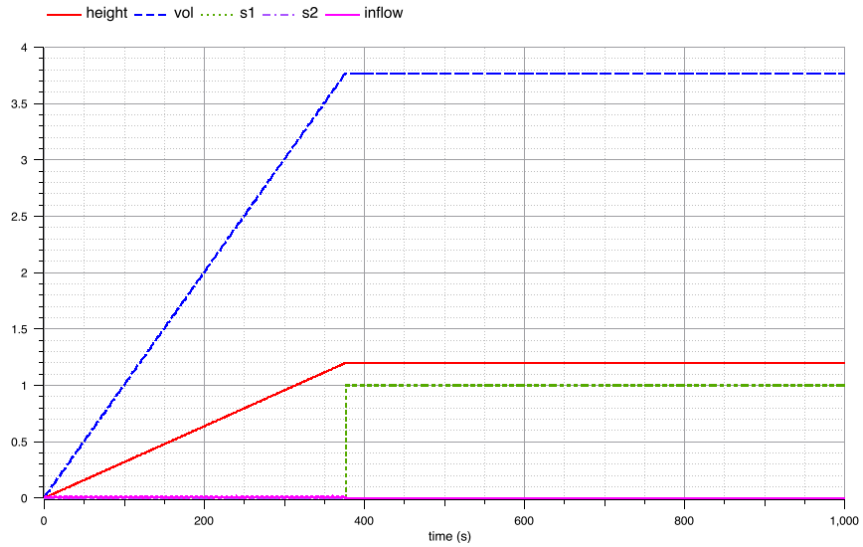


Figure 2 The correct behavior of the water tank example considering the height of the water level, the volume of water in the tank, the sensors and the inflow value. Before time $400s$, the water level reaches the maximum causing sensor $\mathbf{S_1}$ to fire which leads to closing the valve preventing from additional inflow.

A mathematical model of this tank systems comprises one input, i.e., the volumetric flow rate, which fills the tank until reaching the maximum level. If there is still an inflow, then the water will flow through the output of the tank (of course unless it is open and not blocked). The ordinary behavior of this tank example can be modeled as follows: The tank has an area A (e.g., in form of a circle of a given radius r where $A = r^2 \cdot \pi$), a maximum height h_{max} where the water should stop, and the height of the tank h_{top}, which is higher than h_{max}. The inflow in is given in the amount of water per time, i.e., in m^3/s. The current volume of water is a function of time $V(t) = h(t) \cdot A$ where $h(t)$ is the current height at time t. This volume depends on the sum of the inflows over time, i.e., $V(t) = \int_0^{t_C} in(t)dt$, and, hence, $\frac{dV(t)}{dt} = in(t)$. If the water reaches the maximum level, the valve should close. If this is not the case the water level reaches the outpipe causing an outflow. This outflow should be the same than the inflow, i.e., $out = in$, and no further water is added to the tank. When implementing this model in a simulation language like Modelica [13], we obtain a behavior like the one depicted in Figure 2. For this behavior we assumed a tank with a radius of $1m$, a maximum level of $1.2m$, a top level of $1.5m$, and a flow rate of $0.001m^3/s$. Note that we varied the inflow over time using a sinus function. This allows us also to see that the inflow into the tank stops when the valve closes. There are fluctuations of the inflow until reaching the maximum level in Figure 2. Afterward, the inflow is zero without fluctuations.

Note that the behavior is similar in case of changed parameters but the steepness of the increase would be different. However, when using concrete values for diagnosis, i.e., the detection and explanation of misbehavior, we need to adapt any comparison function allowing us to show differences between the real and the expected behavior. If we use simulation, we further would need to adapt parameters to fit the simulation outcome with the one of the real systems due to variations between real components and ideal ones. Hence, a qualitative representation of the behavior to be used to check conformance of behavior would be a good idea. To confirm that a water tank system will operate as expected under typical operating

conditions, a qualitative simulation using ASP-QSIM was carried out. Figure 3 depicts the qualitative behavior our simulation produces for the tank example. Analyzing the plots, we see that as the tank fills, the inflow progressively decreases and eventually stabilizes at zero, while both volume and water level increase until they reach a maximum state, representing the threshold of the tank. Once this maximum is reached, the sensor $\mathbf{S_1}$ switched off as expected. Hence, our simulation accurately captures the tank systems nominal behavior.

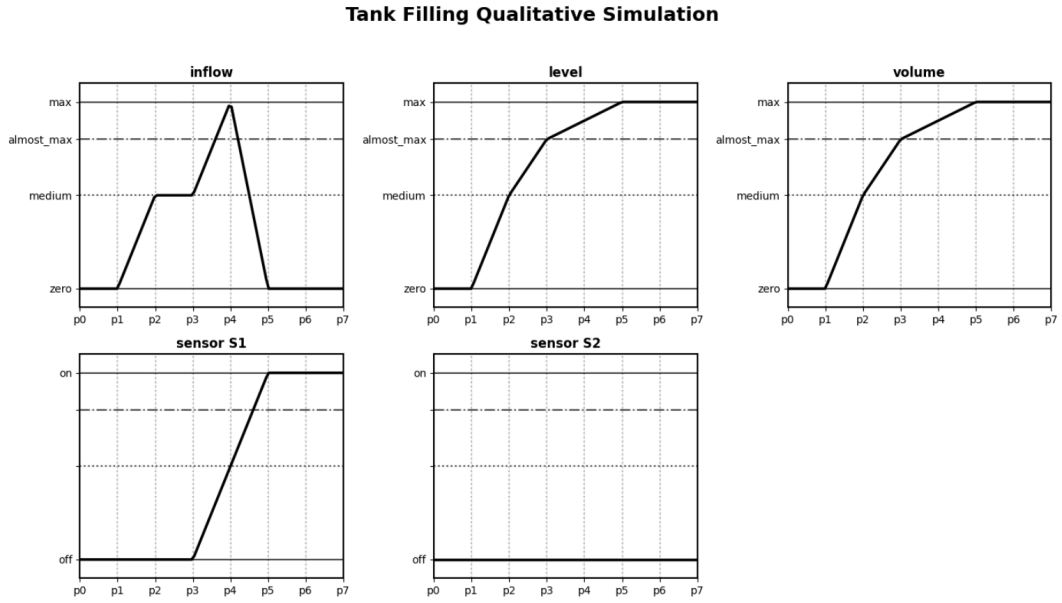


Figure 3 Qualitative simulation behavior for tank system, showing qualitative transitions of inflow, level, volume and sensors across key landmarks.

In the following, we discuss the applicability of qualitative simulation for diagnosis focusing on fault detection. In particular, we want to clarify the following questions:

- What means conformance of a concrete system run and the qualitative simulation result?
- What are the limitations of qualitative simulation when applied for diagnosis? Here, we want to outline limitations considering different fault scenarios. Which type of faults can hardly or not being detected?

For the discussions on limitations, we consider the following fault cases of the tank example, assuming that the fault happens at $200s$ and remains permanent until the end of simulation:

Fault case 1: Sensor $\mathbf{S_1}$ gets stuck at false. In this case, the controller would not get any information and not initiate closing the valve. Hence, the water will reach the outpipe causing sensor $\mathbf{S_2}$ to go from false to true.

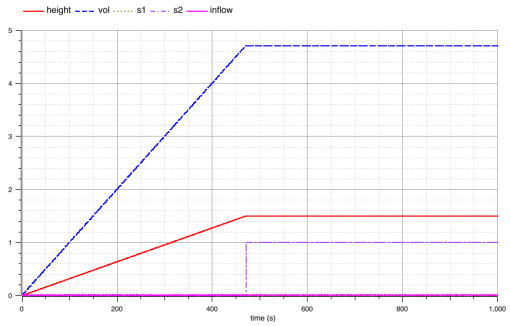
Fault case 2: Sensor $\mathbf{S_1}$ is stuck at true, leading to closing the valve immediately. Hence, the tank will not be filled completely.

Fault case 3: The valve is stuck and always closed leading to zero inflow after $200s$. Hence, none of the sensor will go to true and the tank cannot be filled.

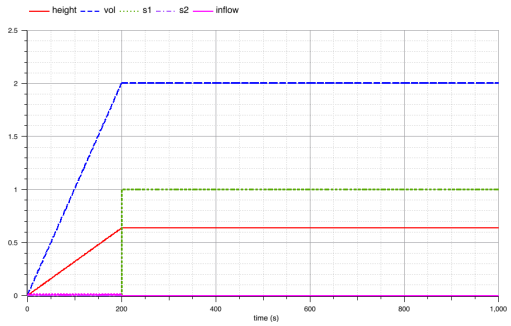
Fault case 4: The valve is stuck and remains open. In this case sensor $\mathbf{S_1}$ indicates reaching the maximum level but having no effect. Hence, the top level is reached and sensor $\mathbf{S_2}$ will go from false to true.

Fault case 5: The valve gets stuck at 50% leading to continuous and not stopping inflow. Both sensor indicate reaching a certain water level but the water inflow does not stop.

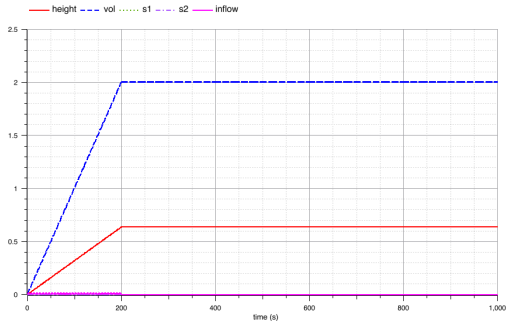
Figure 4 shows the faulty behaviors resulting from the different fault cases. Before discussing the limitations and challenges, we introduce the basic definitions of qualitative simulation in the next section of this paper.



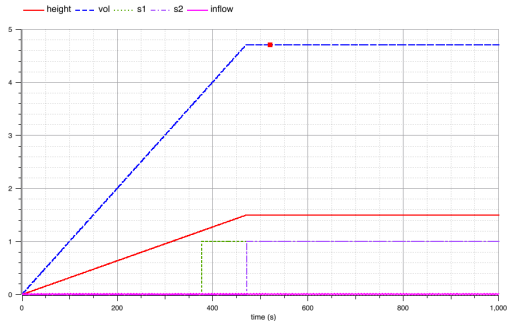
(a) Fault case 1: Sensor $\mathbf{S_1}$ stuck at false.



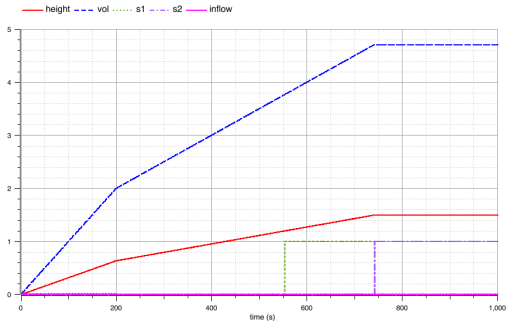
(b) Fault case 2: Sensor $\mathbf{S_1}$ stuck at true.



(c) Fault case 3: Valve stuck and always closed.



(d) Fault case 4: Valve stuck and always opened.



(e) Fault case 5: Valve stuck at 50% open.

Figure 4 Faulty behavior cases of the tank example.

4 Qualitative Simulation (QSIM)

QSIM provides a formal basis for reasoning about physical systems in situations where precise numerical data is unavailable or not needed. Unlike numerical simulation techniques, which require precise parameter values, in QSIM continuous variables are abstracted into symbolic categories such as increasing, decreasing, or landmark values. QSIM simulates all possible behaviors of a system given a qualitative model. Its inputs are: (1) QDEs represented by variables and constraints, and (2) an initial state. QSIM completes the initial state by solving

a constraint satisfaction problem (CSP) over the QDEs. For each consistent state, it then generates all valid successors and filters them using constraint consistency. Rather than predicting a single numeric outcome or trace of a simulation, QSIM captures all plausible system behaviors over time [20].

▶ **Example** (cont.). To illustrate how QSIM is applied in practice, consider out tank system and its system variables V. We model a single *control variable* – the `inflow` – whose value can be externally manipulated. The remaining variables are *state variables*, which represent internal or observable properties of the system. The state variables include the current water level in the tank (`level`), the volume (`volume`), and `sensor S1` and `sensor S2`, which represent sensor readings related to the tank state.

Each variable in a qualitative model is described over a symbolic structure called a *quantity space*. This space consists of key reference points in the domain of the variable, denoted as *landmarks*, and the intervals between them. Landmarks represent semantically meaningful thresholds without requiring precise numeric values. We formalize the *quantity space* as follows.

▶ **Definition 1** (Quantity Space (adapted from [30])). *For each variable $v \in V$, the quantity space $Q(v)$ is defined as an ordered set of landmark values l_i, i.e, $Q(v) = \{l_0, l_1, \ldots, l_m\}$ with $l_0 < l_1 < \ldots < l_m$.*

▶ **Example** (cont.). In the tank system, the landmarks for `level` may include `zero` (representing an empty tank), `medium`, `almost_max`, and `max` (representing the threshold for the tank level). The sensors `S1` and `S2` may use the landmarks `on` and `off` as shown in Figure 3.

Based on the quantity space of a variable, we can now define its *qualitative value* at a particular time step t:

▶ **Definition 2** (Qualitative Value (adapted from [19])). *A qualitative value $QV(V,t)$ of a variable $v \in V$ at time step t is represented as a tuple $\langle qmag, qdir \rangle$, where:*

- *$qmag$ is either a landmark or an interval between two landmarks,*
- *$qdir \in \{$increasing (`inc`), steady (`std`), decreasing (`dec`)$\}$ is the direction of change of v.*

▶ **Example** (cont.). Suppose that in our tank system model, landmarks define the quantity space for the variable `level`, which represents the water level in the tank. The ordered landmarks for this variable are: `zero < medium < almost_max < max`. Suppose that at time t, the water level lies within the interval (`medium`, `almost_max`) and the level is increasing. Then the qualitative value of `level` at time t is given as $QV(\texttt{level}, t) = \langle(\texttt{medium}, \texttt{almost\_max}), \texttt{inc}\rangle$.

These qualitative values form the building blocks of a system's *qualitative state*, which captures the complete system snapshot at a given time:

▶ **Definition 3** (Qualitative State (adapted from [30])). *A qualitative state $S(t)$ at time t is defined as $S(t) = \{QV(v_0,t), QV(v_1,t), \ldots, QV(v_n,t)\}$ over all $v \in V$.*

The collection of the qualitative values of all the variables in a system at a given point in time is its *qualitative state* [30].

▶ **Example** (cont.). For instance, in our tank example in Figure 3, at time point p_4 we obtain the following qualitative state:

$$S(p_4) = \left\{ \begin{array}{c} \texttt{inflow} \mapsto \langle \texttt{max}, \texttt{dec} \rangle,\ \texttt{level} \mapsto \langle (\texttt{almost\_max}, \texttt{max}), \texttt{inc} \rangle, \\ \texttt{volume} \mapsto \langle (\texttt{almost\_max}, \texttt{max}), \texttt{inc} \rangle, \texttt{sensor S1} \mapsto \langle \texttt{off}, \texttt{inc} \rangle, \\ \texttt{sensor S2} \mapsto \langle \texttt{off}, \texttt{std} \rangle \end{array} \right\}$$

4.1 Qualitative dynamics

A qualitative model defines valid qualitative states of the system and how changes in variables lead to transitions between them. When describing a model, qualitative constraints which restrict the magnitude and direction of variable change are represented by Qualitative Differential Equations (QDEs), which include basic constraints such as derivatives, sums, and monotonic dependencies [29].

To better reflect actual system causality in different system operational scenarios, Wiley et al. [29] introduced *qualitative rules*, which allow constraints to be applied conditionally only within specific regions of the system's state space, i.e., when the associated preconditions are met. The rule-based approach models system behavior in a more context-sensitive and dynamic manner.

▶ **Example** (cont.). In our tank system model, the following domain specific qualitative constraints (`C1`–`C4`) and rules (`R1` and `R2`) are applied: [3]

- `C1: deriv(level, inflow)`: The rate of change in water level is determined by inflow.
- `C2: mplus(level, volume)`: Volume increases proportionally with tank level.
- `C3: const(sensor_S1, land(off))`: Under condition `sensor_conditions_1` (see `R1`), the sensor `sensor_S1` is constrained to remain `off`.
- `C4: const(sensor_S1, land(on))`: Under condition `sensor_conditions_2` (see `R2`), the sensor `sensor_S1` is constrained to remain `on`.
- `R1: precond(sensor_conditions_1, bound, level, i, zero, i, almost_max)` ⇒ `const(sensor_S1, land(off))`[4]: When the tank level is between `zero` and `almost_max`, sensor `S1` remains `off`.
- `R2: precond(sensor_conditions_2, equal, level, land(max))` ⇒ `const(sensor_S1, land(on))`: When the tank level reaches `max`, sensor `S1` is switched `on`.

These qualitative constraints reflect physical principles such as flow regulation (`C1`) and conservation of volume (`C2`), while rules reflect the physical behavior of the systems. For instance, the sensor stays inactive during normal filling (`R1`) and activates only at the threshold (`R2`).

Qualitative transitions define how the system evolves over time by moving from one qualitative state to another. For example, if `inflow` exceeds `zero`, the `level` of the tank increases. While *qualitative constraints* describe relationships between variables (e.g., how `level` depends on `inflow`), *transitions* capture the system's dynamics over time. The QSIM algorithm [8] systematically computes these transitions. At each time step t_i and intervall (t_i, t_{i+1}) QSIM uses a transition table, which encodes all valid value combinations permitted by the active constraints [20], to generate a set of candidate successor states S_{i+1} for the current qualitative state S_i. For simplicity, we do not distinguish whether we go from one time point to an interval or vice versa. We only distinguish sequences of states. Further note that this transition process has been extended in ASP-QSIM [29] to support qualitative rules. As a result, only contextually valid transitions – as defined by the qualitative rules – are considered, ensuring that the qualitative simulation reflects both the physical structure and operational logic of the system.

---

[3] We express constraints and rules using the ASP-QSIM formalism [29], where `const` indicates a constraint, `land` a landmark, and `precond` a precondition.

[4] `bound` indicates that the variable `level` is between two values. The `i` indicates that the range of the variable can be between `zero` and `almost_max` including those two values.

The output of a qualitative simulation is a set of *qualitative behaviors* – each a possible sequence of qualitative states that the system might exhibit given an initial and goal condition. We refer to a single qualitative behavior also as a qualitative trajectory and one qualitative behavior reflects one consistent qualitative path the system may follow.

▶ **Definition 4** (Qualitative Behavior [20]). *A qualitative behavior QB is an ordered sequence of qualitative states $QB = \langle S_0, S_1, \ldots, S_n \rangle$, where each state S_i maps each variable $v \in V$ to its qualitative value $\langle qmag, qdir \rangle$ at a time point or interval corresponding to i.*

▶ **Example** (cont.). Consider this first portion of qualitative behavior $QB = \langle S_0, S_1, S_2, \ldots \rangle$ from the tank example with the following qualitative states:

$$S_0 = \{\texttt{inflow} \mapsto \langle \texttt{max}, \texttt{dec} \rangle, \texttt{level} \mapsto \langle \texttt{zero}, \texttt{inc} \rangle, \texttt{volume} \mapsto \langle \texttt{zero}, \texttt{inc} \rangle,$$
$$\texttt{sensor S1} \mapsto \langle \texttt{off}, \texttt{std} \rangle, \texttt{sensor S2} \mapsto \langle \texttt{off}, \texttt{std} \rangle\},$$

$$S_1 = \{\texttt{inflow} \mapsto \langle \texttt{(zero,max)}, \texttt{dec} \rangle, \texttt{level} \mapsto \langle \texttt{medium}, \texttt{inc} \rangle,$$
$$\texttt{volume} \mapsto \langle \texttt{(zero,medium)}, \texttt{inc} \rangle, \texttt{sensor S1} \mapsto \langle \texttt{off}, \texttt{std} \rangle,$$
$$\texttt{sensor S2} \mapsto \langle \texttt{off}, \texttt{std} \rangle\},$$

$$S_2 = \{\texttt{inflow} \mapsto \langle \texttt{(zero,max)}, \texttt{dec} \rangle, \texttt{level} \mapsto \langle \texttt{almost\_max}, \texttt{inc} \rangle,$$
$$\texttt{volume} \mapsto \langle \texttt{(zero,medium)}, \texttt{inc} \rangle, \texttt{sensor S1} \mapsto \langle \texttt{off}, \texttt{std} \rangle,$$
$$\texttt{sensor S2} \mapsto \langle \texttt{off}, \texttt{std} \rangle\}$$

4.2 Conformance

Several approaches to define conformance between a system and its qualitative model have been proposed over the years. In the context of testing, Aichernig et al. [1] introduced the *qrioconf* relation to formally define when an implementation under test (IUT) conforms to a qualitative model, based on whether the observed outputs under a sequence of inputs are a subset of the outputs predicted by the model. Similarly, the *ioco*-based [27] approach for qualitative action systems checks if outputs after a given input trace remain within the allowed qualitative behaviors [7]. In both cases, the qualitative models were constructed using *Garp3* [9], which is founded in Qualitative Process Theory [12]. Both approaches treat qualitative traces as symbolic abstractions of continuous behaviors. Our work adopts a similar idea by comparing traces from system observations against qualitative simulations to check the conformance of the real-world system with the simulation.

First, let us define an *Abstract Qualitative Behavior* (AQB). An AQB is a qualitative trace produced by the simulation or the result of abstracting a quantitative trace from a system run using a system-appropriate discretization. Defining AQBs allows us to formally compare expected behaviors from the model with observed behaviors from the physical system using a conformance relation.

▶ **Definition 5** (Abstract Qualitative Behavior). *An abstract qualitative behavior AQB is an ordered sequence of qualitative states $QB = \langle S'_0, S'_1, \ldots, S'_n \rangle$, where each state S'_i maps each variable $v \in V$ to its reduced qualitative state only consisting of $qmag$.*

In our AQB, we focus solely on the qualitative magnitude *qmag* of each variable and omit the direction of change *qdir*. First, directionality can be more susceptible to noise and measurement imprecision in real-world systems, making it unreliable for behavior alignment. Second, the key behavioral distinctions we want to detect are typically captured by changes in magnitude (e.g., crossing a landmark), which directly reflect system state transitions.

In addition, as we are mainly interested in capturing changes within the system and the qualitative simulation, we abstract away redundant repetitions in a trajectory by minimizing the ABQs, i.e., $\min(AQB)$. That is, we only consider transitions from a state to a successor in case the qualitative value changes; otherwise, we consider two states to be qualitatively equivalent [6]. Consecutive repetitions of qualitatively equivalent states are removed.

▶ **Definition 6** (Minimal Abstract Qualitative Behavior). *Given an AQB, we define the minimal abstract qualitative behavior,* $\min(AQB)$*, as the subsequence of AQB in which all consecutive states are qualitatively distinct, i.e.,* $\min(AQB) = \langle S'_0, S'_1, \ldots, S'_n \rangle \quad where \quad \forall 0 \leq i < n : S'_i \neq S'_{i+1}$.

▶ **Example** (cont.). For our tank example, we created a Modelica simulation to generate system behavior data in place of real-world measurements. However, in practice, the same abstraction process we apply to this simulated data is used to transform time-series data from real physical sensor measurements.

In order to create an abstract qualitative behavior (AQB) from the concrete quantitative data, we first create a quantitative-to-qualitative mapping, i.e., an abstraction. First, we down sample the original time series at a predetermined rate (i.e, per timestep in our case). Then, each sampled value is compared to a set of predefined landmarks $\{\theta_{\text{zero}}, \theta_{\text{medium}}, \theta_{\text{almost\_max}}, \theta_{\text{max}}\}$. If it is within a small tolerance of one of these thresholds, it is considered a landmark (e.g., land(θ)); if not, it is assigned to the open interval between two adjacent landmarks (e.g., interval(θ_i, θ_{i+1})).

However, direct changes from one sample to another may also affect intermediate qualitative areas. To mitigate this loss of information, we introduce n equally spaced "intermediate steps" between each pair of sampled timepoints by linearly interpolating the quantitative values and reclassifying each intermediate value. By altering the number of intermediate stages, trace length and qualitative fidelity can be compromised. We found that $n = 1$ is sufficient to capture all landmark and interval transitions in the underlying numeric trace, while $n = 0$ may overlook significant qualitative changes in our tank example. After building the complete qualitative trace, including intermediate steps, a minimization step is employed. This process, denoted as min(AQB), removes successive states that are qualitatively similar or in which all system variables retain the same qualitative values. This decrease significantly lowers the computing complexity for subsequent reasoning tasks and improves interpretability by focusing exclusively on significant behavioral adjustments. The result satisfies the formal criterion of minimal abstract qualitative behavior by ensuring that no two successive states are identical and encouraging precise and effective qualitative reasoning.

Based on the definition of AQBs, we can define a conformance relation. The goal of the conformance relation is to determine whether the real system behaves in accordance with the simulation, i.e., whether at least one of the AQBs generated by the qualitative simulation matches the AQB derived from an observed system run.

▶ **Definition 7** (Conformance). *Let* $\min(AQB_{real})$ *be the minimal abstract qualitative behavior derived from a system run, and let* $\mathcal{A}$ *be the set of AQBs produced by the qualitative simulation and afterwards minimized. We define* conformance *as:*

$$conformance(AQB_{real}, \mathcal{A}) \iff \exists AQB_{model} \in \mathcal{A} : \min(AQB_{real}) = \min(AQB_{model})$$

This conformance relation requires an exact match over the entire trace. In principle, both the simulation and the real system could produce infinite traces, especially when viewed as executions of transition systems. However, in practical applications we only ever observe

finite prefixes of such behaviors. To address this, we adopt a bounded trajectory conformance approach, inspired by similar bounding techniques in model checking (e.g., bounded model checking [5]). Instead of requiring a full match across infinite behaviors, we check whether the real system trace matches a simulated behavior up to a fixed prefix length m.

▶ **Definition 8** (Bounded Conformance). *We define bounded conformance w.r.t. a prefix length m as:*

$$conformance_m(AQB_{real}, \mathcal{A}) \iff \exists AQB_{model} \in \mathcal{A} : \lceil \min(AQB_{real}) \rceil_m = \lceil \min(AQB_{model}) \rceil_m$$

Here, $\lceil \min(AQB) \rceil_m$ denotes the prefix of a minimal AQB of length m. Thus, bounded conformance reflects cases were the trajectories are equivalent for the first m qualitatively distinct states. While this bounded conformance approach is practical and scalable, it inherently limits fault detection to the prefix of length m, i.e., fault occurring after the prefix cannot be captured posing a challenge in scenarios where faults manifest beyond the checked horizon.

▶ **Example** (cont.). Once we have determined all qualitative observations by abstraction, including the interpolated states from the real-world data, we encode them as ASP constraints:

```
:- not holds(p(t), var, land(), _), time(p(t)).
```

`holds(p(t), var, land(), _), time(p(t))` is a fact used in ASP to define a qualitative state, where `p(t)` is a time point, `var` refers to the qualitative variable (e.g., `level`), `land()` is the qualitative value, and `_` refers to the qualitative direction that is absent in our abstraction as mentioned earlier. `holds(....)` stipulates that a particular qualitative state must exist at a particular point in time, while `:- not holds(....)` ensures that the model is rejected if the fact `holds(....)` is absent. In essence it guarantees that the observed behavior is present in all valid answer sets. Under the guidance of its internal rules, landmarks, and preconditions, ASP-QSIM attempts to generate valid pathways that logically "fill in the gaps" around an initial state and a goal state taken from the quantitative trace. The result of this, as stated earlier, is a set of minimal AQBs.

If one of the simulation model's produced AQBs exactly matches the abstract qualitative behavior AQB_{real} of the real system, then we have shown bounded conformance. In our case, we compared the full observed trace with the complete simulated trajectories. Our experimental results confirmed that the qualitative model accurately captures the system's dynamics, i.e., every qualitative trace derived from the quantitative data of the nominal system behavior matched at least one valid trajectory generated by the simulation model.

4.3 Fault detection

While conformance checking determines whether an observed system behavior matches one of the expected qualitative behaviors generated by the model, it also offers a principled basis for revealing faults. In practice, deviations from nominal qualitative trajectories – especially in the ordering or presence of sensor events – may indicate abnormal or unexpected conditions. By systematically comparing abstracted observations to simulated behaviors, we can detect faults as violations of the conformance relation.

▶ **Example** (cont.). Qualitative simulation is especially effective in detecting any anomaly that adds, removes, or reorganizes sensor "on/off" transitions relative to the expected nominal model behavior. We were able to detect the following faults, as those faults alter the logical structure of the qualitative state transitions, which QSIM explicitly models:

Fault case 1 (Sensor `S1` stuck `false`): Sensor `S2` turns `true` without a prior Sensor `S1` activation, breaking the expected causal sequence.

Fault case 2 (Sensor `S1` stuck `true`): Sensor `S1` changes to `true` prematurely, i.e., before the tank reaches the defined threshold.

Fault case 4 (Valve stuck and always opened): Sensor `S1` turns `true` but fails to trigger valve closure; Sensor `S2` becomes `true` as the valve is not closed.

Fault case 5 (Valve stuck at 50% open): An unanticipated pattern of sensor events appears that does not match any valid qualitative trajectory.

However, we cannot detect **Fault case 3 (Valve stuck and always closed)**. The system enters a stagnant condition as both sensors remain off and the water level remains constant due to the absence of inflow. Since QSIM observes only changes in landmarks or directional trends, such "silent" failures, where there are no new transitions, are indistinguishable from nominal behaviors, e.g., the tank has partially stabilized [20]. Slight quantitative differences that do not cross these symbolic boundaries (i.e., `increasing`, `decreasing`, `steady`) and landmarks are ignored [18].

5 Conclusions

By abstracting quantitative system behavior into qualitative representations, we demonstrated how qualitative simulation can effectively detect anomalies across various scenarios by formally defining conformance between the qualitative simulation and concrete quantitative data. At the same time, our evaluation uncovered important limitations. In particular, the exclusion of directional information hindered the detection of one of the faults, which might have been caught had directionality been preserved during abstraction. This highlights a central problem of exclusively qualitative models, even though they can capture high-level patterns and behavior, they may not be able to detect defects that depend on small or deliberate aberrations.

However, the power of qualitative simulation is in its ability to provide a methodical and understandable approach to diagnosis, especially in situations where complete quantitative data is unavailable or unreliable. When considering fault scenarios, it offers a model-based foundation and excels at identifying anomalies and discrepancies across different system behaviors.

Future improvements could include the integration of directional cues to enhance sensitivity to small or gradual deviations, which are currently lost in the reduced abstraction. While the exclusion of the direction of changes simplifies the qualitative behavior space and reduces computational overhead, selectively reintroducing it either for specific variables or during suspected fault windows could strike a balance between fidelity and tractability.

Additionally, the development of hybrid approaches that combine qualitative and quantitative reasoning [4] could further improve fault detection robustness. Such methods may allow the system to benefit from the interpretability and flexibility of qualitative models, while leveraging quantitative models for precision in ambiguous or borderline cases.

Another area of enhancement lies in the application of bounded conformance not just as a binary check but as a tool for localizing faults temporally within the trace. By identifying the prefix length at which divergence from expected behavior occurs, diagnosers could potentially infer the onset time and probable subsystem associated with the deviation. To address the challenge of "silent" faults failures that do not result in a change of qualitative state and

thus evade detection future work could explore the use of context-aware thresholds, state persistence checks, or anomaly scoring models that monitor for suspicious invariance in critical variables.

Moreover, QSIM may generate spurious or physically implausible trajectories, particularly in more complex systems. Incorporating domain knowledge, constraint filtering, or ranking heuristics will therefore be crucial for scaling the approach to real-world applications.

References

1 Bernhard K. Aichernig, Harald Brandl, and Franz Wotawa. Conformance testing of hybrid systems with qualitative reasoning models. *Electronic Notes in Theoretical Computer Science*, 253(2):53–69, 2009. Proceedings of Fifth Workshop on Model Based Testing (MBT 2009). doi:10.1016/j.entcs.2009.09.051.

2 Aleksander Bandelj, Ivan Bratko, and Dorian Suc. Qualitative simulation with clp. In *Proceedings of the 16th International Workshop on Qualitative Reasoning (QR02)*, 2002.

3 I. Bellanco, E. Fuentes, M. VallÃšs, and J. Salom. A review of the fault behavior of heat pumps and measurements, detection and diagnosis methods including virtual sensors. *Journal of Building Engineering*, 39:102254, 2021. doi:10.1016/j.jobe.2021.102254.

4 Daniel Berleant and Benjamin J. Kuipers. Qualitative and quantitative simulation: bridging the gap. *Artificial Intelligence*, 95(2):215–255, 1997. doi:10.1016/S0004-3702(97)00050-7.

5 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Adv. Comput.*, 58:117–148, 2003. doi:10.1016/S0065-2458(03)58003-2.

6 Harald Brandl, Gordon Fraser, and Franz Wotawa. Qr-model based testing. In *Proceedings of the 3rd international workshop on Automation of software test*, pages 20–17, 2008.

7 Harald Brandl, Martin Weiglhofer, and Bernhard K. Aichernig. Automated conformance verification of hybrid systems. In Ji Wang, W. K. Chan, and Fei-Ching Kuo, editors, *Proceedings of the 10th International Conference on Quality Software, QSIC 2010, Zhangjiajie, China, 14-15 July 2010*, pages 3–12. IEEE Computer Society, 2010. doi:10.1109/QSIC.2010.53.

8 Ivan Bratko. *Learning Qualitative Models*. Pearson Education, 4 edition, 2012.

9 Bert Bredeweg, Floris Linnebank, Anders Bouwer, and Jochem Liem. Garp3 — workbench for qualitative modelling and simulation. *Ecological Informatics*, 4(5):263–281, 2009. Special Issue: Qualitative models of ecological systems. doi:10.1016/j.ecoinf.2009.09.009.

10 Qiong Chen and Nan Li. Model predictive control for energy-efficient optimization of radiant ceiling cooling systems. *Building and Environment*, 205:108272, 2021.

11 Daniel Dvorak and Benjamin Kuipers. Process monitoring and diagnosis: a model-based approach. *IEEE expert*, 6(3):67–74, 1991. doi:10.1109/64.87688.

12 Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24(1-3):85–168, December 1984. doi:10.1016/0004-3702(84)90038-9.

13 Peter Fritzson. Modelica - a language for equation-based physical modeling and high performance simulation. In *Proceedings of the 4th International Workshop on Applied Parallel Computing, Large Scale Scientific and Industrial Problems*, PARA '98, pages 149–160, London, UK, UK, 1998. Springer-Verlag. doi:10.1007/BFB0095332.

14 Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *AI*, 41(1):79–88, 1989. doi:10.1016/0004-3702(89)90079-9.

15 Johan De Kleer and John Seely Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24(1-3):7–83, December 1984. doi:10.1016/0004-3702(84)90037-7.

16 Matthew Klenk, Johan de Kleer, Daniel G. Bobrow, and Bill Janssen. Using modelica models for qualitative reasoning. In *Proceedings of the 27th International Workshop on Qualitative Reasoning (QR 2013)*, pages 1–8, 2013.

17 Matthew Klenk, Johan de Kleer, Daniel G. Bobrow, and Bill Janssen. Qualitative reasoning with modelica models. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1084–1090. AAAI Press, 2014. doi:10.1609/AAAI.V28I1.8876.

18 Benjamin Kuipers. The limits of qualitative simulation. In *IJCAI*, 1985.

19 Benjamin Kuipers. Qualitative reasoning: Modeling and simulation with incomplete knowledge. *Automatica*, 25(4):571–585, July 1989. doi:10.1016/0005-1098(89)90099-X.

20 Benjamin J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29(3):289–338, 1986. doi:10.1016/0004-3702(86)90073-1.

21 Benjamin J. Kuipers. Qualitative simulation, 2001. In Encyclopedia of Physical Science and Technology, pages. 287–300.

22 Franz Lackinger and Wolfgang Nejdl. Integrating model-based monitoring and diagnosis of complex dynamic systems. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, pages 1123–1128. Morgan Kaufmann, 1991. URL: http://ijcai.org/Proceedings/91-2/Papers/074.pdf.

23 Bruno Lucas, Jean Michel Evrard, and Jean Pierre Lorre. A qualitative diagnosis method for a continuous process monitor system. In *Proceedings of the IMACS International Workshop on Qualitative Reasoning and Decision Technologies -QUARDET'93*. CIMNE, 1993.

24 Bernhard Rinner and Benjamin Kuipers. Monitoring piecewise continuous behaviors by refining semi-quantative trackers. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 1080–1086. Morgan Kaufmann, 1999. URL: http://ijcai.org/Proceedings/99-2/Papers/059.pdf.

25 Siddharth Subhramanian and Raymond J. Mooney. Multiple fault-diagnosis using general qualitative models with fault modes. In *Working Papers of the 5th International Workshop on Principles of Diagnosis*, pages 321–325, New Paltz, NY, October 1994.

26 Louise Travé-Massuyès, Liliana Ironi, and Philippe Dague. Mathematical foundations of qualitative reasoning. *AI Magazine*, 24(4):91–106, 2003. doi:10.1609/AIMAG.V24I4.1733.

27 Jan Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Comput. Networks ISDN Syst.*, 29(1):49–79, 1996. doi:10.1016/S0169-7552(96)00017-7.

28 Timothy Wiley, Claude Sammut, and Ivan Bratko. Using planning with qualitative simulation for multistrategy learning of robotic behaviors. In *Proceedings of the 27th International Workshop on Qualitative Reasoning*, pages 24–30, 2013.

29 Timothy Wiley, Claude Sammut, and Ivan Bratko. Qualitative simulation with answer set programming. In *European Conference on Artificial Intelligence (ECAI)*, May 2014.

30 Timothy Colin Wiley. A planning and learning hierarchy for the online acquisition of robot behaviors, 2017. PhD Thesis Defence Paper.

Assessing Diagnosis Algorithms: Of Sampling, Baselines, Metrics and Oracles

Ingo Pill
Institute of Software Engineering and Artificial Intelligence, TU Graz, Austria

Johan de Kleer
c-infinity, Mountain View, CA, USA

Abstract

Assessing and comparing diagnosis algorithms is a surprisingly complex challenge. We have to make decisions ranging from identifying the implications of the chosen baseline, via defining and ensuring a representative sampling strategy, to the choice of metric best suited to capture the computational, probing, or repair costs as well as the deviations from the baseline. We discuss several aspects of the overall challenge, identify related issues, and evaluate a special economic metric.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics

Keywords and phrases Model-based Diagnosis, Diagnosis, Algorithms

Digital Object Identifier 10.4230/OASIcs.DX.2025.5

1 Introduction

Given a symbolic or subsymbolic description of a system's structure along with a set of observations, the task of a diagnostic algorithm is to identify which system components might be faulty so that their failure explains the observed behavior. In this paper, we address the question of how to evaluate the *quality* of such an algorithm's conclusions and its actions.

At first glance, this might seem to be a straightforward task, but it is surprisingly difficult. As a thought experiment, imagine that we have all the faulty exemplars of a system that might ever exist their actual faults. For this exhaustive sample set, we then employ our diagnostic algorithm and determine the *costs* that the algorithm incurs while locating the *actual* fault(s).

There are several inherent difficulties with this simple concept: (1) it is impossible to implement in practice due to the required exhaustive set of faulty exemplars. (2) We need to specify what we mean by "costs." That is, whether we are interested in computational, probing, or economic costs. If we are interested in a combination, we need to define a weighted portfolio metric. In practice, the ideal cost function depends on the actual application scenario, so that it is (3) impossible to anticipate a specific user's needs and preferences in terms of costs. (4) A user will most certainly not have the resources to perform their own detailed evaluation of all combinations of available algorithms, potential cost functions, and faulty exemplars. Consequently, an evaluation will suffer from limitations in terms of sampling and quantification of costs and quality. Those limitations affect, in turn, the representativeness of the conclusions that we can draw from the experiments.

In practice, diagnostic algorithms are usually evaluated with simulated data. In particular, we take a simulation model of one or more systems, inject some fault(s) [22, 20], run the simulation for a (set of) specific input scenarios [22, 14], and then deploy the diagnostic algorithm to determine the incurred costs; we repeat this until we decide to have *enough* samples. That is, until we gained enough confidence in the representativeness of the results. We assume implicitly that the diagnostic algorithm is deterministic. Otherwise, we would need to run the algorithm multiple times for each diagnosis problem.

Further issues are related to the observability of signals, to uncontrollable contingencies such as noise that we face in the real world, or to an algorithm's individual characteristics such as whether it is based on a precise [4, 19] or probabilistic framework [12]. These issues most certainly affect the incurred costs, but it is their effect on the algorithm's results' quality that is most noticeable. A particularly intriguing challenge in this direction is that of defining the baseline against which we compare an algorithm's results. Related to these issues, we

- maybe can diagnose some fault(s) only after a delay [3] that the fault(s) need(s) to manifest [26] on observable signals.
- have to take into account that an algorithm can trade precision with resource expenditure [1]. This can certainly result in incomplete or over-approximated sets of diagnoses that individually over- or under-approximate the fault situation. This requires us to decide how to individually penalize delays, approximations, erroneous, and absent diagnoses.
- might not be able to isolate the actual fault(s) from the available data [11]. For example, if the fault is not triggered by the observed input scenarios. A fault may also remain hidden in observed signals while manifesting only in unobserved ones – a phenomenon that can be either triggered or suppressed by interactions among multiple faults. In extreme cases, some specific fault combinations could even result in an equivalent mutant. That is, a mutated system that is indistinguishable in its I/O behavior from the original one [7].
- have to define the desired baseline: Should the fault(s) injected into a simulation model (or the faults present in a real system) be considered as the ideal result which is obviously problematic for equivalent mutants (see Sec. 2.4), or should we seek to define a well-founded gold standard of diagnoses? To obtain the latter, we would take into account all the limitations of the available data and specify exactly what is theoretically diagnosable from the available (limited) observations *OBS* and the structural model *SD* [17].

In this paper, we isolate and discuss a variety of challenges that we face when aiming to assess diagnostic algorithms. Motivated by those discussions, we formulate definitions and outline metrics that allow us to systematically address the evaluation of diagnosis algorithms. Our ulterior aim is to support educated and quantitatively comparable answers to the fundamental question *Did a specific algorithm return the correct result, and if not, how good are the results?*. Our discussions aim to address evaluation needs that range from simple single-algorithm assessment to multi-algorithm comparisons for specific purposes, and finally to comprehensive evaluations in general contexts like competitions with multiple benchmarks.

We do not aim at a qualitative analysis that focuses on algorithmic concepts. We rather aim at an empirical evaluation of an algorithm's results that is (1) mostly agnostic of the implemented symbolic or subsymbolic concepts and (2) supports a quantitative assessment of an algorithm's results in terms of the quality of the conclusions and their costs. When we say mostly agnostic, we refer to the requirement of having to know how to interpret an algorithm's results. For example, algorithms like [4, 25, 19] report ambiguity groups formed by a subset-minimal diagnosis and its supersets implicitly in the results.

We organize our paper as follows. We analyze a variety of issues to illustrate the complexity of the task in Section 2. We then raise the discussion to a more formal level in Section 3, isolating appropriate notions and definitions that allow us to define and tackle two specific subtasks. Before we conclude in Section 5, we discuss in Section 4 a special economic metric that captures the effectiveness of the repair process and has been used in previous DX competitions [8].

2 Why is the representative evaluation/comparison of a diagnosis algorithm not as straightforward as it seems?

In this section, we seek to identify and illustrate a variety of issues that make the tasks of evaluating and comparing diagnosis algorithms not as straightforward and simple as they would seem at first glance. In the following subsections, we walk through a set of illustrating examples to expose individual challenges that we have to address. The observations we make in these sections will support us in specifying precise definitions in Section 3 that will allow us to lead formal discussions of the concepts envisaged in Sections 3.2 and 4.

2.1 Metrics based on sampling and the notion of ambiguity groups

Suppose that we want to design a good metric for single fault diagnosis algorithms. Assume that we have perfect measurements, which is really only possible in digital circuits. Let the components of the system be $X = \{x_1, ..., x_n\}$. Determining the samples to use to fairly test a diagnostic engine for a competition is a very complex problem that deserves a separate treatment, so let us assume for now that the set of *samples* is given. Let a simple diagnostic algorithm A return only one fault for a specific diagnosis problem s defined by an exemplar SD (with a known fault) and some observations OBS. If A returns the actual fault, the score $r(A, s)$ achieved for sample s is 1 and otherwise it is 0. An overall score $S(A)$ of algorithm A can, in turn, be defined as:

$$S(A) = \frac{\Sigma_{s \in samples} r(A, s)}{|samples|}, \tag{1}$$

such that we would sample over a set of diagnosis problems. Intuitively, the overall score estimates the probability of A producing the correct diagnosis for a random sample $s \in samples$. This simple assessment has numerous shortcomings:

1. *Probabilities:* How to take into account that different components fail at different rates?
2. *Ambiguity Groups:* Sometimes the same symptoms can arise from very different system failures and combinations of faults.
3. *Multiple Faults:* How to take into account that a system can suffer from multiple simultaneous faults?
4. *False Positives:* The sampling approach obviously penalizes false negatives (missed diagnoses), but does not adequately penalize false positives (reporting invalid diagnoses).

It is easy to see that the way we sample for computing $S(A)$ has a great impact on the overall score. A diagnostic algorithm is evaluated against a set of benchmark samples. Imagine a simple system of 4 components $\{x_1, x_2, x_3, x_4\}$ with priors $p(x_1) = 0.001, p(x_2) = 0.01, p(x_3) = 0.1, p(x_4) = 0.889$. If we sampled the fault scenarios representatively in the set of components, *i.e.*, considering an equal distribution, an algorithm A_1 that always scored correctly for components x_1, x_2, x_3 but not for x_4 would achieve a much better overall score $S(A_1)$ than A_2 that scored correctly on x_4, but not for x_1, x_2, x_3. And it does so, even though it is much less useful in practice, *i.e.*, since it scores well, s.t. $r(A_2, s) = 1$ for rare samples only. A possible solution to circumvent this would be to draw samples according to the prior distribution, but this requires far too many samples (*i.e.*, expensive simulations) and the score $S(A)$ would not be generalizable for varying priors.

An exponentially more efficient approach is to sample over each of the possible faults and weigh each sample with its prior. This greatly reduces the number of samples needed to obtain a reasonable score $S(A)$:

$$S(A) = \sum_x p(x) \frac{\Sigma_{s \in samples(x)} r(A,s)}{|samples(x)|}, \tag{2}$$

Due to $\sum_x p(x) = 1$, we see that $S(A)$ will be 1 iff $r(A,s)$ is always correct and 0 if it is always wrong. But this is a poor approach, since ambiguity groups introduce considerably more complexity. An *ambiguity group* is a set of diagnoses among which a diagnoser cannot distinguish given a specific set of observations. For example, consider diagnosing a sequence of n digital buffers where we can observe the input of $buffer_1/x_1$ and the output of $buffer_n/x_n$. If the input is $0/low/\bot$, and the output is $1/high/\top$, we know that one of the buffers is faulted, but not which one. So all n buffers form an ambiguity group. A similar challenge arises in the analog case where there are n resistors in a row, or the two inverter example from Section 2.4 illustrated in Fig. 1. Let us assume for now that the fault is in x_1. Considering the effects that an ambiguity group entails, we need to modify $r(A,s)$. That is, requiring A to return x_1 to achieve $r(A,s) = 1$ makes no sense. Indeed, a diagnoser should not be penalized because it returns a different element of the ambiguity group. We need to redefine what $r(A,s)$ returns. Let $G(s) = \{x | x \text{ is a diagnosis of } s\}$. If we require $r(A,s)$ to return x, it will be right only $\frac{1}{|G(s)|}$ of the time. If we require $r(A,s)$ to return one element of $G(s)$, a diagnoser may always pick the same element of the ambiguity group, making it useless for diagnosing other faults in that ambiguity group. If we require $r(A,s)$ to return the entire $G(s)$, then the diagnoser never gets partial credit. One approach is to give the diagnoser credit for the fraction of the ambiguity group it returns. So, for example, if there are two members in the ambiguity group and the diagnoser returns only one member of the ambiguity group, it would receive a score of $\frac{1}{2}$ for that sample s. Rewriting the prior equation to capture this intuition, we obtain

$$S(A) = \frac{\sum_{s \in allsamples} p(G(s)) \frac{|r(A,s)|}{|G(s)|}}{\sum_{s \in allsamples} p(G(s))} \tag{3}$$

where we assume the diagnoser is sound. This reduces the prior equation in the case there are no interesting ambiguity groups (of size greater than 1.)

In the multiple fault case, these challenges become only more difficult. Assume that we have a priori probability distribution over the set of diagnoses $\Delta_i \in 2^X$. Following the single fault case, we would like to write (where the Δ_i can be of any cardinality):

$$S(A) = \sum_{\Delta_i} p(\Delta_i) \frac{\Sigma_{s \in samples(\Delta_i)} r(A,s)}{|samples(\Delta_i)|} \tag{4}$$

But this has the same kinds of problems with ambiguity groups as discussed earlier. Consider the $n = 10$ buffer example. It has a large number of multiple faults in the ambiguity group as well: $\binom{10}{3} + \binom{10}{5} + ...$ A worse challenge arises in the analog case where there are n resistors in a row where the ambiguity group can be of size $2^n - 1$ because any subset of the resistors can be a diagnosis.

Contributing to this confusion is the fact that we have been mixing together two distinct ideas: (1) the construction of a fair benchmark and (2) the fair evaluation of a diagnostic algorithm on a benchmark. There are two important questions: (1) how to construct a benchmark that truely reflects actual occurring faults, and (2) given a set of samples, how do we fairly evaluate a diagnostic algorithm on this benchmark. Designers of a benchmark must concern themselves with the first question. This paper is primarily focused on the second question. So what we start with is the benchmark which is simply a set of samples

(input-output pairs). Let $G(s) = \{\Delta_i | \Delta_i \text{ is a diagnosis of } s\}$. This defines the ambiguity group of a sample. The prior $p(G(s)) = \sum_{\Delta_i \in G(s)} p(s)$. The prior single fault equation remains unchanged with the updated definition of $G(s)$

Some diagnostic algorithms return a posterior probability $V(\Delta)$ with each diagnosis. This requires a new approach, which we will discuss later.

So far we have penalized a diagnoser for false negatives, but not directly for false positives. But what should happen if the diagnoser returns a false positive, *i.e.,* a diagnoser returns a non-diagnosis. An approach for this has been well studied in machine learning. One can construct a confusion matrix with the diagnoser's results on the vertical axis and the actual faults on the horizontal. (This applies for both single faults and multiple faults.) Then accuracy, true positive rate, false positive rate, and precision can be defined in the usual way as in ML. This also needs to be adapted for ambiguity groups. In ML techniques like Top-k-accuracy can be used.

An approach which avoids the problems we have seen thus far is to base a metric on the difference between the true distribution of faults and the one determined by the diagnostic algorithm.

Let $Q(x)$ be the gold standard – the probability distribution which best approximates the state of affairs the single fault diagnoser encounters. Let $P(x)$ be the distribution the diagnoser reports. The most familiar metric is cross-entropy – which is used widely in ML to compare the predicted probability distributions. In our case, we want to compare how close the calculated probability distribution $P(x)$ is to the perfect one $Q(x)$. For this case, KL-divergence is a better metric:

$$D_{KL}(P||Q) = \Sigma_{x \in \mathcal{X}} log \frac{P(x)}{Q(x)} \tag{5}$$

If $P(x)$ and $Q(x)$ are identical, KL-divergence is 0, unlike cross-entropy. One problem is that if the diagnoser assigns non-zero probability to a possibility which never occurs, then KL-divergence is indeterminate. So this is problematic as well.

We hope we have convinced you that all the sampling based metrics described in this section are problematic measures of a diagnoser's capabilities. Although these metrics are often used because they appear to make sense on the surface, none of them is a decent scoring mechanism for a diagnostic algorithm. Instead, we argue diagnostic algorithms need to be scored based on the costs they incur in use.

Complementing an understanding of the problems associated with the metrics, we hope that we have convinced you also that the sampling strategy can have a strong impact on how we perceive some algorithms' performances. When designing benchmarks, we thus also need to anticipate the issues discussed in this section and define a set of samples that keeps those issues within certain and well-understood limits. In this respect, we would like to point out also that there are diagnosis algorithms that consider multiple samples in one diagnostic process [24], such that *SD* would stay the same for all s but *OBS* would change. As Pill and Wotawa mused in [23], supposedly representative concepts for generating test suites like combinatorial testing could help with defining representative sample sets for such algorithms. Ideally, their employment would allow us in turn to explain not only some occurred faults, but to isolate all the faults present in a system via an integrated approach at constructing representative I/O scenarios and diagnosing the corresponding observations. Such related work could help us not only in exploring options to assess multi-scenario diagnosis algorithms, but also when facing the challenge of designing benchmarks for diagnosis algorithms in general.

2.2 Metrics estimating repair costs

As we concluded in Sec. 2.1, sampling has a major impact on how an algorithm performs. But there are also a variety of concepts for assessing the quality of a diagnostic process, and our choice can significantly influence how we perceive an algorithm to perform for a specific sample s (and in general). In this subsection, our focus is on exploring the effects that a diagnosis algorithm's results have on the repair process.

Let us start with single fault diagnosis scenarios, and this time, the algorithm returns a probability distribution $p(x_1), ..., p(x_n)$ over the system's components $x_i \in X$. These are the posterior probabilities given some specific observations. For example, a system of three components $\{x_1, x_2, x_3\}$ might have posterior probabilities $p(x_1) = 0.1, p(x_2) = 0.6, p(x_3) = 0.3$. Assume for the moment that these posterior probabilities are the true ones. A simple repair approach might now replace x_i in decreasing posterior probability order. We first replace x_2 (due to the highest posterior probability), and if the symptoms are not gone, we consider x_3, and lastly x_1. Assuming the cost of a replacement $rep(x_i)$ was unity, the estimated repair costs C for these posterior probabilities are $rep(x_2) + rep(x_3) * (1 - p(x_2)) + rep(x_1) * (1 - p(x_2) - p(x_3)) = 1 + 0.4 + 0.1$. More formally, we can express C via the following equation for a given list L of diagnoses Δ_i that were sorted with decreasing probability.

$$C = \Sigma_{0<i\leq|L|}(rep(x_i) * (1 - \Sigma_{0<j<i}p(x_j))) \tag{6}$$

This simple repair process takes advantage of the fact that repairing a single component and making all repairs suggested by a diagnosis Δ_i is the same in a single-fault scenario. Obviously, we need to distinguish between the two for multi-fault scenarios though, which, in turn, requires us to adapt the repair strategy. Before we reason about such an update, let us first explore a sometimes hidden assumption though. That is, the assumption of whether we have *Perfect Fault Understanding (PFU)* when inspecting a component, or not.

In particular, similar to *perfect bug understanding* from the software engineering community (as it is relevant in a diagnostic context when assessing spectrum-based fault localization metrics), we have to decide whether or not we can assume the ability to recognize faults with complete certainty when inspecting some component. A simple illustrating scenario would be that of when we would inspect a software program by looking at a part of the code (the component), recognize the faulty code in it, and then fix the component's code accordingly. In the single fault case, we can observe whether the problematic behavior is gone when executing the system after the repair of some component x_j (as suggested by a single-fault diagnosis Δ). So, without requiring PFU, we could just replace x_j and avoid a detailed inspection and fault-oriented repair of x_j.
PFU would be an important assumption in multi-fault scenarios insofar, since we would need to replace all x_j in a diagnosis Δ to supposedly make the observed symptoms disappear. So, when checking the quality of the individual repairs via observing the system's behavior, we would first have to repair *all* $x_i \in \Delta$, and then we would observe executions of the supposedly fault-free system. Consequently, judging the situation after inspecting/repairing a single component x_i from Δ would require PFU of x_i.

A repair strategy for multi-fault scenarios could exploit PFU as follows. Choose a component x_i that appears in many diagnoses and then investigate and, if needed, repair x_i. If x_i was found to be correct in the inspection (due to PFU), all diagnoses Δ_j containing x_i can be discarded s.t. they get a probability of 0, and the probabilities $P(\Theta)$ for all diagnoses that are still possible (with search space $\Theta = 2^X$) should be updated accordingly. We repeat this until the symptoms have gone, there is no diagnosis left, or until there is no component left to repair. Obviously, estimating the costs C of this repair process is a bit

more complicated than for a single fault scenario. That is, we need to constantly update $P(2^X)$ after investigating some x_i, and we have to statistically approximate the costs incurred when estimating C.

If we do not have PFU (like when we just swap components without inspection), we could simply walk through the list of diagnoses Δ_i according to their probabilities and repair for each investigated Δ_i all components $x_j \in \Delta_i$. Extending our Eq. 6, we can approximate the average repair costs in this case as

$$C = \Sigma_{0<i\leq|L|}((\Sigma_{x_k\in\Delta_i} rep(x_k)) * (1 - \Sigma_{0<j<i} p(\Delta_j))), \tag{7}$$

with $rep(x_k)$ referring to the repair costs of component x_k. When following this process, one specific downside is that a certain component x_k might get repaired more than once. That is, if it is contained in more than one Δ_i that we had to investigate before arriving at the actual diagnosis (so that the system is finally repaired).

A *supposedly* better repair process would keep track of whether individual components have been repaired before, so as to avoid unnecessary repairs. However, this requires that a repaired component is not destroyed again during the repair process. That is, before all x_i in the actual diagnosis have been repaired. In practice, this can easily happen, when the interaction of faults leads to the immediate destruction of a repaired component x_j by some x_k that has not yet been repaired. A simple example would be that of a fuse. The fuse could be immediately destroyed again if the problem that triggered it has not yet been taken care of. Considering such dependencies, the more naive and also supposedly more costly process could even be better suited (and cheaper) in practice for many applications.

Please note that the metrics we considered in this subsection are simplistic variants of the economic metric discussed in Section 4.

2.3 Metrics tailoring to specific evaluation requirements

When researchers present a new algorithm, they often evaluate it in the context of a family of relevant algorithms and use specific metrics that allow them to show the validity and effectiveness of the proposed improvements over the state-of-the-art. This approach enables a clear and focused presentation of these improvements and allows the authors to adhere to the page limits of a conference at the same time. However, it comes with certain limitations.

Let us illustrate them with the example of RC-Tree [19]. RC-Tree is a diagnosis algorithm that implements a conflict-driven computation as first proposed by Reiter [25] as well as de Kleer and Williams [4] in their seminal papers. The computational concept behind RC-Tree is close to that of HS-DAG [10], but due to some additional reasoning in terms of a divide-and-conquer exploration of the search space, RC-Tree can avoid all redundancy in the search. Intuitively, any diagnosis candidate gets generated only via one tree-based exploration sequence (in contrast to HS-DAG that allows permutations), which results in a narrower search without losing important properties like completeness, soundness, being an anytime algorithm, or supporting an on-the-fly calculation of the conflicts. When evaluating RC-Tree, the authors proved (1) that it returns diagnoses adhering to Reiter's theory in a sound and complete manner (see Defs. 4 and 5 in Section 3.1), and they conducted (2) an empirical evaluation that isolated the performance advantages over HS-DAG as encountered in practice. The algorithm's run-time and the number of evaluated diagnosis candidates (which relate to tree nodes and thus memory consumption) served as metrics in those experiments. The resulting evaluation certainly meets one's expectations in terms of elucidating the authors' contributions. But it does not provide us with a holistic picture on how this algorithm

compares to the entire algorithmic landscape. That is, in comparison to completely orthogonal concepts – like subsymbolic algorithms that would approximate probability distributions (see Section 2.1).

The use of metrics that focus, *e.g.*, on repair costs (see Sec. 2.2) could provide a clearer picture, but those metrics are not suited for highlighting RC-Tree's algorithmic improvements over the state-of-the-art. Experts can draw on their expertise to maintain their own educated picture of the algorithmic landscape. For example, since RC-Tree returns the same diagnoses as HS-DAG (which has been around for about four decades), we can infer that the repair costs and other aspects can be considered to be the same and that there is only a difference in the computational costs. Consequently, this allows an expert to put RC-Tree into perspective. A non-expert would hardly have the knowledge to make such deductions. Metrics and evaluations that are more informative from a global perspective could thus help them in establishing a holistic picture as quickly as possible. Only such a holistic picture would allow a diagnostician or engineer to make an educated decision when faced with the challenge of having to select the most suitable diagnosis algorithm for a specific diagnosis problem.

2.4 Defining the baseline: On the ideal results of a diagnosis algorithm

When we assess the performance of a diagnosis algorithm, we need to think not only about sampling and metrics, but we also need to know how the ideal results should look like. In this subsection, we focus on the latter, and we hope to convince you that this supposedly simple task is not as straightforward to address as it seems.

In order to illustrate some of the issues, let us have a look at a simple example in the form of the circuit illustrated in Fig. 1. It is a simplified variant of the n-buffer example from Sec. 2.1 and contains. two inverters. Assume that we can observe the input in_1 and the output out_2 and let us consider three scenarios for an input value of $in_1 = \top/1/high$.

It is easy to see that the nominal output (such that both inverters work correctly) is $out_2 = \top$. If we observe this expected output in practice, we must recall though that this would be the case also for some fault scenarios. That is, observability limitations (like that can't observe out_1) might prohibit us to distinguish the nominal case from some fault scenarios. For our example, this would be the case for Fault Scenario 1 where Inverter 1 suffers from a stuck-at-zero fault so that out_1 is always $\bot$. The same is true for for Fault Scenario 2 where Inverter 2 suffers from a stuck-at-one fault. In both cases, the fault does not manifest in the observations, but this would require the input to be$in_1 = \bot/0/low$. Consequently, the nominal case, Fault Scenario 1 and Fault Scenario 2 form an ambiguity group (see Sec. 2.1). A consistency-based diagnosis algorithm like RC-Tree [19], HS-DAG [10], or GDE [4] would report from this ambiguity group only the empty diagnosis which represents nominal behavior. In fact, the underlying computation concept of those algorithms entails that when using a weak fault model, any superset of a diagnosis Δ is also an explanation. Thus they form sort of an ambiguity group with Δ by default and are, in turn, not explicitly

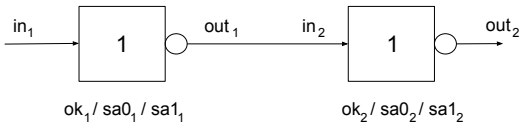


Figure 1 A simple electronic circuit with two inverters, taken from [21].

reported. From this example, we can easily see that we need to know *how to interpret a specific algorithm's results in terms of aspects like implicit ambiguity groups, and we need to decide how we would formalize this in the baseline.*

Let us now consider Fault Scenario 3 which illustrates a special case of an ambiguity group. In this fault scenario, both inverters malfunction so that they act as buffers and pass their input values along instead of inverting them. For this double fault, we can easily observe that there is no input that can reveal it. That is, unless we would improve the observability. Either via monitoring the intermediate signal $out_1 (= in_2)$, or by adding a connection from out_1 to some circuit part that is connected to a different observable output. In fact, we can observe that the system behaves *exactly* as intended on its observable interfaces not only for Fault Scenario 3, but for all I/O scenarios. In particular, the two faults cancel each other's negative effects on the system's output so that there *cannot* be any evidence of this double fault on the observable signals. Since the mutated system is I/O-conformant, we can refer to it as *equivalent mutant* [7].
From a designer's perspective, such an equivalent mutant is among equivalent design choices for implementing the desired I/O behavior. This presents us with quite interesting challenges from a diagnostic perspective. That is,*we need to explore the question of whether a diagnosis algorithm is supposed to report equivalent mutants as diagnoses or not, and, potentially, we need to identify the set of equivalent mutants.*

For elucidating further issues, let us consider two general concepts for defining a baseline, *i.e.*, (1) using the ground truth, and (2) defining/computing a golden set of diagnoses.

For the first option, we would use ground truth knowledge about the faults that are present in a system and would expect an algorithm to report it. Assume that we, *e.g.*, have a simulation-based setup like we discussed it in the Introduction. Since we know which faults we injected and when, we could simply consider this ground truth as gold standard for the diagnosis algorithm's results. This choice would be certainly intuitive and natural, but it comes with several problems, as we can illustrate using the above fault scenarios.
Consider Fault Scenario 3, where, no matter the input, a diagnosis algorithm is likely to completely fail if we do not consider implicit ambiguity groups. That is, since it will most probably report the empty diagnosis instead of the double fault.
But there are more issues. So, what if the faults have not manifested *yet* and there is potential subsequent behavior for nominal and faulty cases? That is, despite the fault being present we could have (a) for a stateless system that we might not *yet* have seen an input combination that manifests the fault(s) and (b) for a stateful system that we did not *yet* experience the delay that is necessary for the manifestation. While the baseline would indeed require the faults' detection, the question is on which basis an algorithm would justify that it reports a related diagnosis? Would we consider the algorithm then to be (a) clairvoyant (see Sec. 3), does it simply report (b) a statistic assessment with corresponding probabilities for all the possible continuations and their fault combinations/diagnoses, or (c) should the diagnosis actually be considered spurious – despite being defined in the baseline?

Reflecting on these issues, we might thus rather choose to define the baseline via computing a well-founded set of diagnoses that takes into account limitations in terms of observability and diagnosability. Although this approach would allow us to address some of the issues mentioned above, it comes with *increased computational costs* and causes its own problems. Let us consider first the diagnosis of a stateless system like the two-inverter circuit. If a system is stateless, its behavior depends only on the current input which means in a diagnostic context that we can diagnose each time step individually. The same holds then also for the computation of the baseline for which we can either use a proven algorithm or manual

analysis. The split into independent computations allows us to address a potential scalability issue that could otherwise arise with larger systems and long computation sequences and their traces. With the focus on individual time steps, the sizes of *SD* and *OBS* are minimized for each individual computation. Still, *we need either an algorithm that can compute the baseline, or we need to manually inspect the diagnosis problem(s).*
For a stateful system, the computational demands might be even higher. That is, we have to consider the entire history of observations and can't split the execution into smaller problems for each time step. The simple reason is that this history is reflected in an internal (potentially unobservable) state [18] that is relevant for the system's I/O relation. Like the authors did for [18, 9], we can still use one health state variable for the entire execution. And this allows us to constrain the exponential diagnosis search space to 2^X instead of $2^{X \times T}$ (for a weak fault model and a temporal horizon of length T). We might also not loose timing information when a fault manifests since this information might be present in the underlying computation (like in the conflicts when using RC-Tree).But the models for *SD* and *OBS* will still grow linearly with T.
It is important to note that these scalability issues are further exacerbated when we evaluate diagnosis algorithms that consider not only one but a set of executions [24] – like when a diagnostic algorithm examines *all* failed tests after executing a test suite [23].

For our final argument, assume that we are investigating a live scenario and that we managed to compute the gold standard of diagnoses for all the individual prefixes. As we suggested in the introduction, we now *have to decide how to penalize the various types of deviation from the baseline.* We have, for instance, that the diagnoses reported could be over- or under-approximations of diagnoses in the baseline. Or there could be a delay in detecting a diagnosis, which would require us to compare the diagnoses reported for time step t_i with the baseline for time step $t_j \neq t_i$. Accommodating these and a variety of other deviations entails another increase in the computational costs and it illustrates that we must not consider metrics and baselines in isolation. In Section 3.1, we will define the notions of near-soundness and near-completeness that will allow us to discuss this in more detail and to support automated reasoning in this context.

3 A more formal take on the problem at hand

In this section, we move our discussion to a more formal level. We start by developing some formal definitions in Section 3.1. Based on these definitions, in Section 3.2, we look at how we can answer the questions of whether *a specific diagnosis algorithm returned the correct result for some diagnosis problem, and if not, how good the results would be.* In particular, we propose to implement an *oracle* for providing a yes/no answer, and we design a *variety of metrics* for quantifying how far off an algorithm is from reporting the ideal results. These metrics can be used to derive an answer for the second (part of the) question, but also in the isolation of a yes/no answer to the first (part).

3.1 Some definitions

Note that the notation for some of our definitions deviates from what the reader might be familiar with from the literature and papers like [4, 25, 19]. The purpose of these deviations is to accommodate the results from as many diagnosis algorithms/approaches as possible, including results like diagnosis probability distributions that were computed under the consideration of strong fault models.

Assume that a system SD consists of components $x_i \in X$, and that we have some observations OBS about the behavior of SD[1]. As indicated in Section 2.1 and following the literature [4, 25], we can define from SD, X and OBS a diagnosis problem (SD, X, OBS) where we, *e.g.*, sampled over such diagnosis problems in Eq. 1. A diagnosis $\Delta \subseteq X$ for a diagnosis problem points us to a subset of faulty components that can explain OBS. To this end, a diagnosis assigns each component $x_i \in X$ a mode $m_j \in M_i$, where M_i is the finite set of modes in which a component x_i can be.
M_i must contain the nominal mode m_{nom} and at least one fault mode. This can be the general fault mode m_{gen}, but also one or more concrete fault modes m_k. The general fault mode m_{gen} implements a *weak fault model* [4, 25] that does not place any restrictions on the behavior of x_i in the faulty case. In contrast, concrete fault modes pose concrete constraints on the behavior of x_i also in the faulty case. Technically, there is no requirement to describe the exact behavior of a *strong fault model* but only some constraint. In practice, though, we tend to do so and define concrete faulty behavior – like a stuck-at-0 fault for a logic gate. As the respective authors outlined in [5, 25], exact strong fault models have advantages and disadvantages. For example, they indicate to a user what exactly happened. This information is certainly welcome during the inspection and repair of the system. When diagnosing specifications or design ideas, using a strong fault model allows us to even suggest repairs [18], which we can exploit in *generative model-based diagnosis* for design purposes as suggested in [17]. But the advantages are secured at the cost of an increased search space, since we now have not only two modes per component, but multiple ones. On an algorithmic level, we are then also limited by the chosen fault modes in our diagnostic search.
We consider nominal mode assignments in a diagnosis Δ to be optional, so that we require only those tuples (x_i, m_j) to be specified in Δ that assign a non-nominal mode $m_j \in M_i \setminus \{m_{nom}\}$ to some component x_i. All components $x_i \in X$ for which Δ does not contain an assignment tuple are assumed to be in nominal node. If we state that a diagnosis Δ is a subset of X, like the definitions in [4, 25] suggest, we refer to the observation that Δ defines a subset of components that are faulty.

▶ **Definition 1** (diagnosis problem, diagnosis, subset-minimal diagnosis, diagnosis candidate)**.** *Given a system SD, a set OBS of observations about SD's behavior, a set of components X, and for each $x_i \in X$ a set of modes M_i that x_i can take and where M_i contains x_i's nominal mode m_{nom} and at least one fault mode. A diagnosis Δ for a* diagnosis problem *(SD, X, OBS) is defined as a set of mode assignments (x_j, m_k), such that component x_j is assigned mode $m_k \in M_j$ and there are no two assignments (x_j, m_k) and (x_l, m_m) that refer to the same component (s.t. $x_j = x_l$). We allow nominal mode assignments to be optional in Δ, so that any $x_j \in X$ where Δ contains no mode assignment $(x_j, *)$ is assigned m_{nom} by default. A set of mode assignments, a.k.a.* diagnosis candidate *Δ becomes a diagnosis if, and only if the diagnosis algorithm assigns it a probability higher than zero (s.t. $p(\Delta) > 0$) such that it supposedly can explain OBS. A diagnosis Δ is subset-minimal, if and only if there is no diagnosis Δ' s.t. $\Delta' \subset \Delta$.*

Like in Section 2.1, we assume that the results of a diagnosis algorithm are given as a probability distribution $P(\Theta)$ over the diagnosis space Θ. The gold standard in terms of the expected results of an algorithm will be referred to as $Q(\Theta)$, where we discuss in Section 2.4 several concepts to define this baseline. If an algorithm algorithm does not report

[1] As usual, we will also formally refer to a system model with SD and it will be clear from the context whether we refer to the system in general or a model

probabilities but only a set of solutions, a naive concept to create a distribution is to assign all *diagnoses* the same probability and non-diagnoses a probability of zero. More elaborate approaches would consider the components' priors and a diagnosis Δ's cardinality when computing $p(\Delta)$, but let us continue this discussion later in this section.
Based on the individual M_i from Def. 1, the search space for diagnoses Θ varies in its size and structure. Let us first consider the cases where $\forall x_i \in X : M_i = \{m_{nom}, m_{gen}\}$, so that we implement a weak fault model. In this case, $P(\Theta)$ and $Q(\Theta)$ are distributions over the diagnosis space $\Theta = 2^X$. This space grows to M^X if we implement strong fault models such that M is a general set of modes for all $x \in X$. If the fault sets vary between individual components (which is likely to be the case in practice), we can use individual mode sets M_i and $\Theta = \Pi_{x_i \in X} M_i$ for a more precise characterization.

▶ **Definition 2** (diagnosis probability distribution)**.** *Given a diagnosis problem* (SD, X, OBS) *as of Def. 1, the diagnosis search space* Θ *is defined by the individual mode sets* M_i *for the components* $x_i \in X$ *as their cross product. A* diagnosis probability distribution $P(\Theta)$ *assigns each diagnosis candidate* $\Delta' \in \Theta$ *a probability* $0 \leq p(\Delta') \leq 1$ *s.t.* $\Sigma_{\Delta' \in \Theta} p(\Delta') = 1$*.*

▶ **Corollary 3.** *As per Def. 1, a diagnosis candidate* $\Delta' \in \Theta$ *from a probability distribution* $P(\Theta)$ *as of Def. 2 is a diagnosis iff its probability is higher than zero such that* $p(\Delta') > 0$*.*

Soundness and *completeness* are important properties of any diagnosis algorithm A, since they capture whether (1) the diagnoses reported by A are indeed diagnoses (in that they follow a formal definition like our Def. 1), and whether (2) the algorithm will return *all* diagnoses if it runs long enough. There's an abundance of reasons why algorithms could be incomplete or unsound. Obvious examples are approximating concepts like subsymbolic approaches, but there are also well-founded analytic concepts. For example, if a diagnosis algorithm is tasked to derive *one* minimum-cardinality diagnosis, we usually employ optimizations that result in an incomplete search but where we can still guarantee that at least one minimum-cardinality diagnosis survives and that A reports sound results [2, 6]. Another example would be that of computing all minimal conflicts between SD and OBS (see [4, 25] for the underlying theory), and where we subsequently employ an approximating algorithm like Staccato [1] for computing diagnoses as the minimal hitting sets of the conflicts.

Assessing traits like completeness and soundness should thus be an integral part of any approach that assesses the quality of a diagnosis algorithm and its results, especially if we do not have detailed knowledge of the algorithm (or its implementation) but have to evaluate it anyway. Having white-box access to an algorithm and its implementation provides us with a powerful and intuitive approach to do so, *i.e.*, by deriving formal proofs. In some contexts, such as competitions, we may have only limited knowledge about the diagnosis engine rather than detailed information about its implementation. But even if we would know the algorithm's details, we need to be aware that a proof would most likely target the *algorithm*'s computational concept and seldomly its *implementation*. In this paper, we are thus interested in assessing soundness and completeness in the context of a diagnosis algorithm's results, rather than proving these traits for the algorithm's computational concept.
In the context of $P(\Theta)$ and $Q(\Theta)$, intuitively, soundness refers to the question whether a diagnosis Δ reported in $P(\Theta)$ (s.t. $p(\Delta) > 0$) is also a diagnosis in $Q(\Theta)$. When evaluating completeness, we need to verify that all diagnoses in $Q(\Theta)$ are also diagnoses in $P(\Theta)$.

▶ **Definition 4** (soundness)**.** *Given the diagnosis search space* Θ*, a diagnosis* $\Delta \in \Theta$ *reported in* $P(\Theta)$ *is* sound*, iff it is indeed a diagnosis s.t.* Δ *is a diagnosis in* $Q(\Theta)$*. The results of a diagnosis algorithm are sound, iff all diagnoses* Δ_i *reported in* $P(\Theta)$ *are sound.*

▶ **Definition 5** (completeness). *Given the diagnosis search space Θ, the results of a diagnosis algorithm A are* complete*, iff there is no $\Delta_i \in \Theta$ that is a diagnosis in $Q(\Theta)$ but not a diagnosis in $P(\Theta)$.*

It is easy to see that the two definitions support any diagnosis search space, but they do not take into account how the algorithms encode their results. In particular, as we observed in Sec. 2.4, some algorithms mention some ambiguity groups *implicitly*. That is, when using a weak fault model, algorithms like [25, 4, 19] report only Δ_i for an ambiguity group defined by a subset-minimal diagnosis Δ_i and all its supersets. That is, since all of Δ_i's supersets would qualify as diagnosis according to our Def. 1 by default.
Continuing our discussion on how to generate $P(\Theta)$, we thus might have to enrich the reported set of diagnoses, like we suggested in Section 2.4. In the current case, we would derive all the missing probabilities for diagnoses that are entailed by the reported ones. All non-diagnoses are then finally assigned a probability of 0. Please note that other algorithms might call for a much more elaborate post-processing.

It is intuitive that Definitions 5 and 4 leave no room for deviations between $P(\Theta)$ and $Q(\Theta)$ when we look at the respective sets of diagnoses. When considering our discussion in Section 2.4, the choices we make for defining the gold standard $Q(\Theta)$ could, however, result in the situation that even the most precise algorithm could not achieve completeness and soundness. Potential delays in diagnosability could be one of these reasons, *i.e.*, when we consider the known ground truth about injected faults to define $Q(\Theta)$ and thus do not consider observability-related issues that would cause the faults to manifest on the observable signals (and thus in *OBS*) only after some delay (see Sec. 2.4).

To this end, let us introduce temporal and cardinality-oriented error bounds. Via those bounds, we can then define some maximum deviations in terms of temporal deviations or over- and under-approximations of a diagnosis. Within those bounds, we will consider the results to be *near-complete* and *near-sound*.

▶ **Definition 6** (near-completeness). *The results of a diagnosis algorithm A are* near-complete *with respect to bounds $\epsilon_C \in \mathbb{N}_0$ and $\epsilon_T \in \mathbb{N}_0$, iff for every diagnosis Δ_i in $Q(\Theta)$ for timestep t_j, there is some diagnosis Δ_k for timestep t_l in $P(\Theta)$ such that*

- *Δ_k is a sub- or superset of Δ_i,*
- *$||\Delta_i| - |\Delta_k|| \leq \epsilon_C$, and*
- *$|l - j| \leq \epsilon_T$.*

Near-completeness allows us to reason within a temporal scope for situations where we compute diagnoses for each individual time step t_i of a stateful system's computation (see our discussion in Sec. 2.4). For every diagnosis in $Q(\Theta)$ for time step t_i, we then search for matching diagnoses in $P(\Theta)$ for any time step $t_{j-\epsilon_T} \leq t_l \leq t_{j+\epsilon_T}$. This allows us to deal with delayed observability as well as with the potential clairvoyance (see Def. 14) of, *e.g.*, subsymbolic algorithms. If we do not want to consider temporal deviations, or for the usual case of diagnosing the entire computation after it finished, we just have to set ϵ_T to 0. Via parameter ϵ_C, we can control the acceptance of over- or under-approximations of diagnoses Δ_i from Q, such that P would contain only super- or subsets of Δ_i. ϵ_C allows us to define a limit on the difference in cardinality.

▶ **Corollary 7.** *The results of a diagnosis algorithm A are complete as of Def. 5 if and only if they are* near-complete *as of Def. 6 with respect to bounds $\epsilon_C = \epsilon_T = 0$.*

▶ **Definition 8** (strict near-completeness). *The results of a diagnosis algorithm A are strictly near-complete iff the results are incomplete, but where there are some bounds ϵ_C and ϵ_T for which they are near-complete.*

▶ **Corollary 9.** *Let the results of a diagnosis algorithm A be near-complete, and let $\epsilon_{C_{min}}$ be the minimum value for ϵ_C enabling near-completeness (ϵ_T being a free variable) and $\epsilon_{T_{min}}$ be the corresponding minimum value for ϵ_T (ϵ_C being a free variable). Then the results of A are* not *necessarily near-complete for bounds $\epsilon_{C_{min}}$ and $\epsilon_{T_{min}}$.*

Similar to considering completeness within temporal and cardinality-oriented bounds, let us consider also soundness within such well-defined bounds.

▶ **Definition 10** (near-soundness)**.** *A diagnosis Δ_k from $P(\Theta)$ for time step t_l is* near-sound *with respect to bounds $\epsilon_C \in \mathbb{N}_0$ and $\epsilon_T \in \mathbb{N}_0$ iff there is a diagnosis Δ_i for time-step t_j in $Q(\Theta)$ such that*

- *Δ_i is a sub-or superset of Δ_k,*
- *$||\Delta_i| - |\Delta_k|| \leq \epsilon_C$, and*
- *$|j - l| \leq \epsilon_T$.*

The results of an algorithm for time step t_l are near-sound, iff all of its diagnoses Δ_k from $P(\Theta)$ for time step t_l are near-sound.

▶ **Corollary 11.** *The results of a diagnosis algorithm A are sound as of Def. 4 if and only if all reported diagnoses are* near-sound *as of Def. 10 with respect to bounds $\epsilon_C = \epsilon_T = 0$.*

▶ **Definition 12** (strict near-soundness)**.** *The results of a diagnosis algorithm A are strictly near-sound iff there is at least one reported diagnosis Δ_k from $P(\Theta)$ for time-step t_l that is not sound as of Def. 4, and every such diagnosis is near-sound for some bounds ϵ_C and ϵ_T as of Def. 10.*

▶ **Corollary 13.** *Let the results of a diagnosis algoritm A be near-sound, and let $\epsilon_{C_{min}}$ be the minimum value for ϵ_C that enables near completeness (s.t. there is some ϵ_T) and $\epsilon_{T_{min}}$ be the corresponding minimum value for ϵ_T (s.t. there is some ϵ_C). Then the results of A are* not *necessarily near-sound for bounds $\epsilon_{C_{min}}$ and $\epsilon_{T_{min}}$.*

When we discussed near-completeness as of Def. 6, we briefly mentioned that an algorithm might be clairvoyant. Let us briefly formalize such clairvoyance and let us connect it to strictly near-sound algorithms.

▶ **Definition 14** (clairvoyance, clairvoyant diagnosis)**.** *A near-sound diagnosis Δ_k from $P(\Theta)$ reported by algorithm A for time step t_l is called a* clairvoyant diagnosis*, if and only if there is a diagnosis Δ_i in $Q(\Theta)$ for some time-step $t_{l<j\leq l+\epsilon_T}$ s.t. Δ_i is a sub- or superset of Δ_k with $||\Delta_i| - |\Delta_k|| < \epsilon_C$, but there is no such Δ_i for $t_{l-\epsilon_T \leq j \leq l}$.*

It is easy to see that a clairvoyant diagnosis means that the algorithm is by definition unsound, *i.e.*, since there was no matching diagnosis for the same time step.

▶ **Corollary 15.** *If the results of a diagnosis algorithm A contain a clairvoyant diagnosis Δ, then the algorithm's results are not sound. The results are strictly near-sound iff there exist appropriate bounds ϵ_T and ϵ_C such that the results are near-sound with respect to those bounds.*

It is evident that the notions of near-completeness and near-soundness allow us to classify and compare some algorithms' performance for a diagnosis problem via inspecting ϵ_T and ϵ_C. That is, via their respective minimum parameters ϵ_T and ϵ_C that are required for achieving near-soundness and/or near-completeness. The lower the minimum values for the parameters, the better the algorithm. Via ϵ_T and ϵ_C we can thus establish partial orders in the sense that

when we fix one of these parameters, the minimum values for the other parameter defines a natural order regarding their performance. For obtaining a total order, we would need a weighted metric considering $\epsilon_{T_{min}}$ *and* $\epsilon_{C_{min}}$, or we could take into account the entire set of all (minimal) combinations. For a representative verdict, some (weighted) average over a set of samples might need to be computed, implementing the discussion offered in Section 2.1.

Please note that the values for parameters ϵ_T and ϵ_C depend on each other, as can be seen from Corollaries 9 and 13. A consequence that we can easily observe is that we might need values for ϵ_T that are higher than $\epsilon_{T_{min}}$ for enabling near-completeness and/or near-soundness for some specific ϵ_C (and vice versa).

3.2 The challenge: of oracles and metrics

We motivated our work in the Introduction with the complexity of finding correct and informative answers to the natural question of whether *a specific diagnosis algorithm returned the correct result for some diagnosis problem, and if not, how good the results are.* As we hope to have convinced you with our discussion in Section 2, providing the right answers is a complex task that requires us to take into account a wide variety of aspects.

Let us now discuss potential solutions to the problem at hand. In principle, we can observe that any solution needs to tackle two distinct problems:

- Task 1: We need to implement an *oracle* that judges whether the results are OK or not.
- Task 2: We need to quantify the quality of the results with a corresponding measure.

T1 and T2 *can* be addressed and answered independently, but we can also exploit synergies and tackle both tasks together. So we could use a quantitative metric for T2 and implement the oracle O for T1 by a qualitative interpretation of the value obtained by the metric.

An intuitive and straightforward solution for the latter would be to consider $P(\Theta)$ and $Q(\Theta)$ and to sum up the absolute value of the differences between $p(\Delta_i)$ and $q(\Delta_i)$ when iterating through the diagnosis space Θ. We could adopt KL-convergence as of Eq. 5 in Section 2.1, we could sum up the squared absolute value such as to penalize larger differences, and we could adopt a large a variety of similar ideas with individual twists. That is, as long as we ensure that the final value is 0 iff there is no deviation at all, we have a solution with the desired property. Obviously, this concept is not very robust against the influence of noisy computations. If we add some error bound $\epsilon \in \mathbb{R}^+$ to mitigate some of the effects (see Eq. 8 for such an oracle O), we would still not be able to address the disadvantage that the metric (the sum in Eq. 8) does not distinguish between problems with different severity levels.

$$O(P, Q, \epsilon) = \top \textbf{ iff } \Sigma_{\Delta_i \in \Theta} |p(\Delta_i) - q(\Delta_i)| \leq \epsilon \textbf{ and } \bot \textbf{ otherwise} \tag{8}$$

Let us elucidate the issue by considering the two following cases. For Case 1, assume that there is a very small deviation $\delta_i = |p(\Delta_i) - q(\Delta_i)|$ in the probabilities for some $\Delta_i : (p(\Delta_i) > 0 \ \wedge q(\Delta_i) > 0$. Case 1 refers thus to a scenario in which Δ_i is a diagnosis in both $P(\Theta)$ *and* $Q(\Theta)$, but in which we observe a slight deviation between $p(\Delta_i)$ and $q(\Delta_i)$. In the context of a repair procedure, this difference might cause a change in the rank of Δ_i, which could in turn affect repair cost estimates (see Section 2.2). But we can easily see that A neither missed to report diagnosis Δ_i in $Q(\Theta)$, nor is Δ_i a spurious diagnosis. Mathematically, we thus observe that the completeness and soundness of the results is not negatively affected by the difference in probabilities. Now let us look at Case 2, in which the probability for the same Δ_i is 0 for either $P(\Theta)$ *or* $Q(\Theta)$ and is δ_i for the other. In Case 2, while the difference is indeed the same as in Case 1, we can easily observe that the results $P(\Theta)$ are either incomplete or unsound, *i.e.*, since A either failed to report a diagnosis (if

$p(\Delta_i) = 0$), or it reported a spurious and thus unsound diagnosis (if $q(\Delta_i = 0)$). The metric can, however, not distinguish Case 2 from Case 1 so that Δ_i contributes in either case with the same δ_i to the computation.

An intuitive way to address the issue would be to employ a combination of such a simple quantitative metric and a formal assessment of completeness and soundness. Following up on our discussions in Sections 2.4 and 3.1, this might also call for inspecting near-soundness and near-completeness as of Definitions 10 and 6 instead, since this would allow us, *e.g.*, to accommodate delays in the diagnosability of some faults.

In combination with this formal assessment, we could replace the metric in Eq. 8 with a much simpler one. First, we rank the diagnoses for $P(\Theta)$ and $Q(\Theta)$ respectively, and then subsequently count the Δ_is for which there is a difference in ranking (taking into account ambiguity groups). This simplification would make the quantitative measure more robust to noise and slight deviations in the probabilities that do not change the ranking, while it already takes the effects on a repair process into account. There is an abundance of similar metrics, and in Section 4 we discuss with the economic metric a much more elaborate approach to quantify effects on the repair costs, and thus a metric that is closer in line with our repair-cost estimate discussion from Section 2.2.

It is easy to see that the combination of a formal assessment with some quantitative ranking could be realized in a *combined* metric that allows us to tackle T1 and T2 together, but also in individual solutions for T1 and T2. We could use the assessment of soundness and completeness for addressing T1, and employ a quantitative metric for T2. Approaches for T1 following the concept described above require us to know $Q(\Theta)$. In reality, this assumption is indeed a strong one and we may not be able to fulfill it in practice. Let us thus explore whether we might be able to assess completeness and soundness without knowing $Q(\Theta)$.

So, we can verify the soundness of a single diagnosis Δ_k, for example, by checking if the symptoms disappear after repairing all (necessary) x_i as suggested by Δ_k. Connecting to our discussions around PFU in Sec. 2.2), such a Δ_k is indeed sound. Furthermore, considering Def. 1, we do not need to verify whether Δ is subset-minimal, since it is not required by our definition. For these two reasons, in general, Δ_k should also be part of $Q(\Theta)$). That is, unless we break the hidden assumption that we consider the ambiguity groups of diagnoses and their supersets in a sensible way such that $Q(\Theta)$ would miss to contain also the relevant supersets of some diagnosis as diagnoses.
Please note that if we would desire subset-minimality in the diagnoses for a weak fault model computation, it would not suffice to simply check whether $P(\Theta)$ contains another diagnosis $\Delta_l : \Delta_l \subset \Delta_k$ – like some diagnosis algorithms do. That is, this check works out for algorithms like RC-Tree [18] since they are sound *and* complete. But while completeness is ensured in those algorithm's construction, it is not necessarily the case in our context.

Verifying completeness is also sometimes possible without knowing the expected results. For example, in cases where we have access to a precise system model that is suitable for working with a SAT, SMT or constraint solver. Assuming that we have, *e.g.*, a behavioral model in some temporal language [18, 9] or a simple combinational circuit [4], we first compile *SD* and *OBS* into a SAT problem as suggested in those papers (see also [13]), or into an SMT problem/constraint problem [16]. We then add blocking clauses for all diagnoses from $P(\Theta)$ and finally ask the solver to search *exhaustively* for a new diagnosis [13, 15]. If one is found, the results $P(\Theta)$ are incomplete, and otherwise they are complete.

We hope that we could convince you that there are many ways to tackle the problem at hand, despite all the challenges that we have to face. To illustrate this, we, indeed, discussed several options to implement the desired oracle (T1), as well as to quantify the quality of the results reported by a diagnosis algorithm (T2).

4 The Economic Metric

The intuition behind economic metrics for diagnostic algorithms is to evaluate their results in the context of their use. Here we will focus on the task of a repair person who's task it is to return the system to its full functioning. (Many other analogous diagnostic metrics are possible.) If diagnosis is completely accurate the diagnostician can simply replace the components listed in the diagnosis. But if the diagnosis is not accurate the diagnostician may replace components which are unfaulted, and in addition incur extra diagnostic expense to restore the system to complete functioning. Thus they incur two types of wasted effort. The report of DXC 2010 [8] outlined one approach which we first summarize and then augment according to the ideas outlined earlier in this paper.

The diagnostic algorithm returns a set of diagnoses: $\Omega = \{\omega_1, \ldots, \omega_k\}$. In addition it provides a weight $V(\omega)$ for each diagnosis. The weights typically represent probabilities. We assume

$$\sum_{\omega \in \Omega} V(\omega) = 1$$

The normalized utility including both types of wasted effort is (see [8] for details):

$$m_{utl}(\omega, \omega^*) = 1 - \frac{n(N+1)}{f(n+1)} - \frac{\bar{n}(\bar{N}+1)}{f(\bar{n}+1)}$$

where f is the count of all components, ω is a diagnosis, N is the number of healthy components, n is the number of false negatives ($|\omega^* - \omega|$), ω^* is the inserted fault, $\bar{n}$ is the number of false positives ($|\omega - \omega^*|$). From this definition we can see that $f = N + \bar{N}$ and $n + \bar{n} = |\omega^* \triangle \omega|$ where $\triangle$ indicates symmetric difference. m_{utl} intuitively has the desired properties. If there are no false positives or false negatives, $m_{utl} = 1$. A system with 10 components for which the diagnoser finds the incorrect single fault, $m_{utl} = 0.4$. If the diagnoser returns multiple diagnoses, the utility for the diagnoser on that task is:

$$M_{utl}(\omega^*) = \sum_{\omega \in \Omega} V(\omega) m_{utl}(\omega^*, \omega)$$

This was the exact metric used in prior DXC competitions. However, it does not accommodate ambiguity groups. Therefore, to obtain a decent score at all, the diagnosers in prior competitions needed to return as many elements of the ambiguity group each with its own $V(\omega)$. The final scores were thus much lower than expected. This was an unfortunate state of affairs as at the point this was discovered the metrics had been fixed for the competitors.

The full discussion of the utility metric in the presence of ambiguity groups is outside the scope of this paper. Generalizing the approach we used for the sampling approach we obtain:

$$S(A) = \frac{\sum_{s \in allsamples} p(G(s)) M_{utl}(s)}{\sum_{s \in allsamples} p(G(s))} \tag{9}$$

5 Conclusions

We hope that we have convinced you that choosing a metric for the evaluation of a diagnostic algorithm is very complex. In addition, we argue that the best metrics for evaluating the algorithms' performance could be those that are based on the costs incurred by the diagnostic algorithm in its context of use, like the economic metric. In particular, we observe that this

family of metrics provides us with an assessment that is agnostic to the algorithm itself and connects directly to how we perceive its performance by considering the practical effects of the derived results.

A complementing way to assess the results of a diagnostic algorithm is to look at their completeness and soundness. In practice, we would want to consider some error bounds in this context, as suggested by our notions of near-completeness and near-soundness. While we discussed the intuitiveness of computations that take advantage of knowing the gold standard, in specific circumstances, we can implement approaches that are able to circumvent the strong requirement of having to know the ideal results.

Following up on our discussions of issues that have to be addressed and our suggestions for concepts to solve the problems, future work will have to empirically evaluate available and new metrics to provide us with a clear picture of the algorithmic landscape.

References

1 R. Abreu and A. J. C. van Gemund. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *8th Symposium on Abstraction, Reformulation, and Approximation, SARA*, 2009. URL: `http://www.aaai.org/ocs/index.php/SARA/SARA09/paper/view/834`.

2 J. Biteus, M. Nyberg, and E. Frisk. An algorithm for computing the diagnoses with minimal cardinality in a distributed system. *Engineering Applications of Artificial Intelligence*, 21(2): 269–276, 2008. `doi:10.1016/j.engappai.2007.03.006`.

3 A. Boussif and M. Ghazel. Diagnosability analysis of input/output discrete-event systems using model-checking. *IFAC-PapersOnLine*, 48(7):71–78, 2015. `doi:10.1016/j.ifacol.2015.06.475`.

4 J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987. `doi:10.1016/0004-3702(87)90063-4`.

5 J. de Kleer and B. C. Williams. Diagnosis with Behavioral Modes. In *11th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1324–1330, 1989.

6 Johan de Kleer. Hitting set algorithms for model-based diagnosis. In *22nd International Workshop on Principles of Diagnosis (DX'11)*, 2011.

7 P. Delgado-Pérez and F. Chicano. An experimental and practical study on the equivalent mutant connection: An evolutionary approach. *Information and Software Technology*, 124:106317, 2020. `doi:10.1016/j.infsof.2020.106317`.

8 A. Feldman, T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, J. de Kleer, L. Kuhn, and A. van Gemund. Empirical evaluation of diagnostic algorithm performance using a generic framework. *International Journal of Prognostics and Health Management*, pages 1–28, 2010.

9 A. Feldman, I. Pill, F. Wotawa, I. Matei, and J. de Kleer. Efficient model-based diagnosis of sequential circuits. In *34th AAAI Conference on Artificial Intelligence (AAAI'20)*, pages 2814–2821. AAAI Press, 2020. `doi:10.1609/AAAI.V34I03.5670`.

10 R. Greiner, B. A. Smith, and R. W. Wilkerson. A Correction to the Algorithm in Reiter's Theory of Diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989. `doi:10.1016/0004-3702(89)90079-9`.

11 A. Hou. A theory of measurement in diagnosis from first principles. *Artificial Intelligence*, 65(2):281–328, February 1994. `doi:10.1016/0004-3702(94)90019-1`.

12 I. Matei, M. Zhenirovskyy, J. de Kleer, and A. Feldman. Classification-based Diagnosis Using Synthetic Data from Uncertain Models. *Annual Conference of the PHM Society*, 10(1), 2018.

13 A. Metodi, R. Stern, M. Kalech, and M. Codish. Compiling Model-Based Diagnosis to Boolean Satisfaction. In *26th AAAI Conference on Artificial Intelligence*, pages 793–799, 2012.

14 E. Muškardin, I. Pill, and F. Wotawa. CatIO - A Framework for Model-Based Diagnosis of Cyber-Physical Systems. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, and Z. W. Raś, editors, *Foundations of Intelligent Systems*, pages 267–276, 2020.

15 I. Nica, I. Pill, T. Quaritsch, and F. Wotawa. The Route to Success - A Performance Comparison of Diagnosis Algorithms. In *23rd International Joint Conference on Artificial Intelligence*, pages 1039–1045, 2013.

16 I.-D. Nica and F. Wotawa. ConDiag - Computing minimal diagnoses using a constraint solver. In *23rd International Workshop on Principles of Diagnosis*, 2012.

17 I. Pill and J. de Kleer. Challenges for Model-Based Diagnosis. In *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*, volume 125 of *Open Access Series in Informatics (OASIcs)*, pages 6:1–6:20, 2024. doi:10.4230/OASIcs.DX.2024.6.

18 I. Pill and T. Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *23rd Int. Joint Conf. on Artificial Intelligence*, pages 1053–1059, 2013.

19 I. Pill and T. Quaritsch. RC-Tree: A variant avoiding all the redundancy in Reiter's minimal hitting set algorithm. In *IEEE Int. Symp. on Software Reliability Engineering Workshops (ISSREW)*, pages 78–84, 2015.

20 I. Pill, I. Rubil, F. Wotawa, and M. Nica. SIMULTATE: A Toolset for Fault Injection and Mutation Testing of Simulink Models. In *IEEE 9th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 168–173, 2016.

21 I. Pill and F. Wotawa. Spectrum-Based Fault Localization for Logic-Based Reasoning . In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 192–199, 2018. doi:10.1109/ISSREW.2018.00006.

22 I. Pill and F. Wotawa. On Using an I/O Model for Creating an Abductive Diagnosis Model via Combinatorial Exploration, Fault Injection, and Simulation. In *29th International Workshop on Principles of Diagnosis (DX'18)*, 2018.

23 I. Pill and F. Wotawa. Exploiting observations from combinatorial testing for diagnostic reasoning. In *30th Int. Workshop on Principles of Diagnosis*, 2019.

24 I. Pill and F. Wotawa. Computing Multi-Scenario Diagnoses. In *31st International Workshop on Principles of Diagnosis, DX ; Conference date: 26-09-2020*, 2020. URL: http://dx-2020.org/.

25 R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

26 L. Ye, P. Dague, D. Longuet, L. B. Briones, and A. Madalinski. Fault manifestability verification for discrete event systems. In *22nd European Conf. on Artificial Intelligence*, pages 1718–1719, 2016. doi:10.3233/978-1-61499-672-9-1718.

Towards Predictive Maintenance in an Aluminum Die-Casting Process Using Deep Learning Clustering and Dimensionality Reduction

Miguel Cubero
University of Valladolid, Spain

Luis Ignacio Jiménez
University of Valladolid, Spain

Daniel López
HORSE Powertrain, Valladolid, Spain

Belarmino Pulido[1]
University of Valladolid, Spain

Carlos Alonso-González
University of Valladolid, Spain

Abstract

In the manufacturing industry, predictive maintenance requires the estimation of the health status of key subsystems or components. In this study, we will look for degradation patterns in the piston of an injection machine used in an aluminum die casting process operating in an automobile factory in Valladolid (Spain). The injection machine produces a new engine block every 90 seconds and each injection device provides 2000 measurements of various physical variables. This study faced the challenge of finding piston head degradation patterns for an injection machine in the factory, using time series data obtained from the controller, as a preliminary step to estimate the remaining useful life (RUL) of the piston head. The proposed solution used advanced deep learning clustering techniques to generate an index related with the progression of the degradation of the components. The results indicated that degradation patterns can be identified. Later on, using an exponential function an approximation of the RUL can be provided to the plant operator to achieve an ordered piston replacement.

2012 ACM Subject Classification Computing methodologies → Cluster analysis; Computing methodologies → Dimensionality reduction and manifold learning

Keywords and phrases Prognostics, Deep Learning, Clustering, UMAP, LOWESS regression

Digital Object Identifier 10.4230/OASIcs.DX.2025.6

Funding This work has been partially supported by Spanish Ministerio de Ciencia e Innovación under Grant PID2021-126659OB-I00.
Miguel Cubero: M. Cubero's work has been supported by the 2024 Investigo Programme from the Spanish Ministerio de Trabajo y Economia social using EU Next Generation Funds.

Acknowledgements Authors want to thank our former students David Garcia de Vicuña López de Ciordia and Daniel Veganzones for their help in early stages of this work; also, authors want to acknowledge the people from the Informatics and Injection Sections of the factory for granting access to the time series dataset, and for the help and support provided in understanding the injection process.

[1] Corresponding author

1 Introduction

Predictive manufacturing is one of the basic pillars of smart factories under the Industry 4.0 paradigm. This task is essential to improve industry competitiveness [23] and intelligent systems are required to obtain "self-awareness", "self-predicting", "self-maintaining", and "self-learning", which are inherent capabilities of predictive manufacturing [16]. Health Management Systems (HMS) provide the conceptual framework to build these intelligent systems, which must include all the necessary (and cooperating) tasks to get those capabilities: health state estimation, monitoring, fault detection and diagnosis, prognostics, and predictive maintenance. Prognostics and Health Management (PHM) is the technology commonly used to develop HMS. Such technology aims to detect incipient faults, perform fault diagnosis, and prognostics of failure [3]. Different authors have identified different tasks as essential to develop PHM solutions [3, 1], being the most common: Human-Machine Interfaces, data acquisition, detection, diagnostics, and prognosis modules. In this work we will focus mainly on fault prognostics and predictive maintenance, aiming to extend the useful life of the system, and helping in the optimization of maintenance activities [20].

There are two main approaches to solve the PHM problem: model-based and data-driven [4, 24]. Model-based prognostics [8] use physics-based models that model physical phenomena in order to predict how failure in a system or in its components will evolve. The task has two steps: damage estimation, and damage prediction, which projects the current health state forward in time to determine the End of Life (EOL), and the Remaining Useful Life (RUL) of the system or component. Meanwhile, data-driven prognostics use available data to build a black-box model for predicting the fault growth [10, 25]. In order to extract features from the data that can be related to fault growth [19] different techniques can be used, such as regression (e.g., Gaussian process regression), mapping (e.g., neural networks), or statistics (e.g., relevance vector machines).

In the context of automotive industry, several subsystems or some of their components are so complex that it is not possible to obtain accurate or usable first-principles models. However, in smart factories, with their Cyber-Physical systems (CPSs), large amounts of data are available, specially from a large variety of sensors or automata. The big data obtained from those CPSs can be used to produce data-driven models for HMS purposes; but such kind of models for large parts of a factory are hardly feasible. Instead, using a *divide and conquer* strategy, smaller models for specific parts of the process or for specific high-level tasks in the HMS (such as fault diagnosis or prognostics) can be obtained. In discrete manufacturing, this is not a major problem because different stages can be clearly isolated.

More specifically, in the domain of automotive manufacturing, several stages can be clearly identified: the production of components (such as engines, gearboxes, etc.), wielding, painting, and assembly. In addition, we can also identify several discrete processes within each stage. This study focuses on one of these processes, required for producing car engines: the aluminum die-casting process, which produces engine blocks. This process is so complex that it is divided into different phases, one of them being the injection of aluminum to produce each engine block. A new aluminum engine block is produced in less than two minutes using an injection machine, whose components also suffer from wear, especially moving parts such as pistons, which are subjected to extreme temperature and pressure conditions. The automatic controller, embedded in each injection machine, measures several physical variables during each aluminum injection. These process measurements for each engine block are stored in the real time database as time series, together with some static data related to the whole injection process, which are stored for maintenance purposes.

In this context, the research questions in this work were: first, is it possible to find degradation patterns for pistons involved in the aluminum die-cast injection process, using only the time series related to physical variables provided by the controller? Second, can we perform an estimation of the RUL for the piston in the injection machine? To answer the first question the calculation of a Fuzzy Progression Index (FPI) is proposed; this index shows the evolution of the piston head wear in sequential stages, combining Autoencoders, dimensionality reduction, and fuzzy clustering. The answer to the second question, the FPI values were used to estimate the current wear state of the piston head, and a projection of its remaining life span.

This study has been done using real data from an automotive factory in Valladolid, Spain; more specifically from one of its injection stations during a three months period in 2023 where abnormal degradation patterns were suspected. From a factory management point of view, finding those degradation patterns would help the predictive maintenance of the system, thus reducing the downtime of the process due to pistons failures.

This manuscript is organized as follows: next section will provide a description of the real case study used in this work. Later, background on the main techniques used in this work will be introduced. Section 4 is focused on the proposal for the estimation of the wear condition of the piston heads, while Section 5 will explain how to obtain an estimation of the RUL for each piston head. In both sections we will provide the results on the case study. Finally, some conclusions will be drawn.

2 Case study: the aluminum die-casting process in an automotive factory in Valladolid, Spain

2.1 The injection process

Car manufacturing has several stages: *Stamping* (which creates the vehicle basic outer shell out of large rolls of sheet metal), *Body Shop* (which is devoted to welding and the assembly of the stamped panels from the previous stage), *Painting* of the car body, *Assembly Line* (where wiring, electronics, windows, seats, etc. are assembled), *Powertrain Assembly* (where engines and transmissions are installed), *Quality Control and Testing*, and finally *Logistics and Delivery*. In addition, all the components required in these stages need to be produced, such as engine blocks or gearboxes, before they can be assembled. Usually there are whole factories devoted to the production of an specific element, such as the engine block.

This work is focused on the die-casting process, where injection machines produce engine blocks from aluminum ingots, as can be seen in Figure 1. This work aims to build an intelligent system to support quality control for the die-casting process, by means of a degradation model, which can be used to warn the plant operator when the RUL of the system is close. After the die-casting process, the produced engine blocks are tested in the factory before they are delivered to the *Powertrain Assembly* stage.

Aluminum die-casting has several stages itself, which can be seen in Figure 2: the aluminum is melted in a *Melting tower*, and it goes directly to the *Casting* stage; afterwards, the engine blocks pass a *Visual inspection*; later on, each engine receives a *Thermal treatment* to avoid leakages, and the exceeding aluminum is removed in the *Machining* stage. Finally, to find potential liquid or gas leakages, each engine passes through the *Leakage test*. Engine blocks that do not pass this test are sent backwards to the *Melting tower*.

Filling the mold is a process which also has several stages: first, the aluminum is fed into the piston chamber. Second, there is a slow-speed phase, where the piston moves slowly to almost completely fill the mold. Third, there is the low pressure, high-speed phase to

Figure 1 Aluminum die-casting machine.

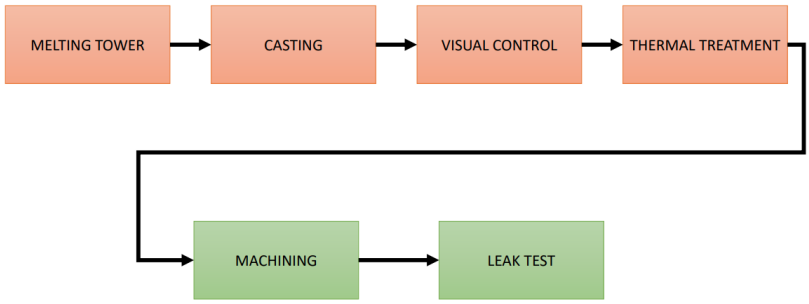


Figure 2 Die-casting process diagram.

completely fill the mold. Fourth, compaction stage uses high pressure to remove air or any other gas from the mold. Later on, a vaccum valve is used to seal the mold. During all these phases, the piston speed and pressure must be carefully controlled, because it is critical to produce fine engine blocks. Finally, once the mold is open again, a layer of an oil-based solution is sprayed on both the engine block, and the mold. Then, there is a cooling stage.

In this study, the information used belongs to three of the four initial stages: slow-speed, high-speed, and compaction. All the data used from these stages are provided by the injection machine.

2.2 The dataset

Each injection lasts less than 90 seconds, and in that span each injection device produces 2000 data for at least five measured signals for each piston: the space traveled, the current speed and its desired set point, and finally, the pressure exerted by the piston on the mold and its desired set point. Consequently, for each injection process, we have five time series made up of 2000 points each[2]. In addition, the injection machine provides a text file summarizing the injection process, and accordingly it labels each new engine block. If the generated label is not **OK**, then the engine block is rejected, but most of the generated engines are labeled as **OK**.

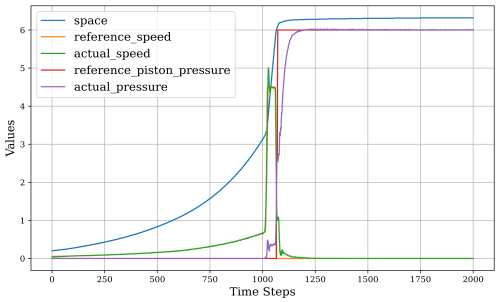
[2] Details about measurement units and scales are intentionally omitted due to confidentiality issues.

Figure 3a shows the evolution of the five measurements recorded for the relevant parts of one injection: the end of the slow speed, together with the fast speed, and the compaction stages, as described in Section 2.1. However, depending on the settings fixed by each plant operator, the time series had different parameters regarding the initial time point to be measured, and the sampling time for the data series. Hence, we were forced to subsample the original 2000 points to have the same number of points, with reference to the same time instant, for all the injections. Consequently, the time series were reduced to 158 points comprising the end of the slow-speed and the whole fast-speed stages for all the engine blocks. After analyzing those data, the size of the time series was later reduced to 56 points, available from the beginning of the fast-speed stage in the injection. Figure 3b shows the 56 points for the same injection as in Figure 3a.

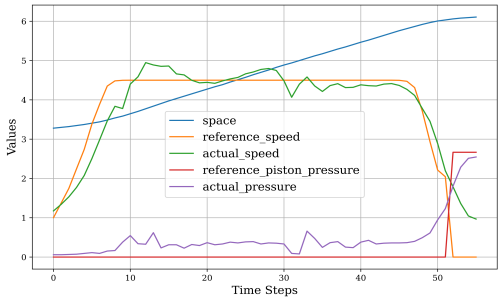
After training several Deep Learning configurations [7] only 20 time steps were considered as relevant to distinguish different behaviours. From these reduced data, only the real speed ($v \in \mathbb{R}$) and the real pressure ($p \in \mathbb{R}$), measurements were considered relevant. Additional features, related to the temporal evolution of these two measurements were included. Each feature will provide an additional 20 points time series for both v and p.

- the standard deviation (five points sliding window), $\sigma \in \mathbb{R}$,
- relative speed: $\frac{x(t+1)-x(t)}{x(t)}$, $rs \in \mathbb{R}$,
- curvature: $\frac{x(t+1)-2x(t)+x(t-1)}{(1+(x(t+1)-x(t))^2)^{3/2}}$, $c \in \mathbb{R}$,
- third order difference: $x(t) - 3x(t-1) + 3x(t-2) - x(t-3)$, $tod \in \mathbb{R}$,
- and lags of order 1 to 5: $x(t) - x(t-k)$, $k = 1, 2, 3, 4, 5$, $lag_k \in \mathbb{R}$.

As a result, the dataset is made up of 20 time series, made up of 20 time steps each, for each engine block: $x_i = \{v_i, \sigma_1, rs_1, c_1, tod_1, lag_{1_1}, lag_{1_2}, lag_{1_3}, lag_{1_4}, lag_{1_5}, p_i, \sigma_2, rs_2, c_2, tod_2, lag_{2_1}, lag_{2_2}, lag_{2_3}, lag_{2_4}, lag_{2_5}\}$.



(a) Five time series covering relevant phases of the injection process for a single engine block.



(b) Time series for the fast speed phase of the injection process for a single engine block.

Figure 3 Comparison of time series representations across different phases of the injection process for a single engine block.

The dataset comprised data about engine blocks produced over a three-months span (mid-January to mid-April, 2023) by a single injection machine, and labeled as **OK** by the device. 20 pistons were used during that period of time, but the factory reported that four of them did not meet the expected lifespan standard, and were replaced due to unforeseen issues (the remaining 16 exhibited normal lifespans). A filtered dataset was established based on these 16 pistons, resulting in a total of 29158 engine blocks. The generated dataset has dimensions $N \times P \times F$, with N representing the 29158 engine blocks, P indicating the time points collected for each time series, 20, and F representing the number of time series used for each block, 20.

3 Background: unsupervised machine and deep learning methods

Raw data is made up of five time series for each engine block, without labels on the different stages of the life cycle of the piston to perform classified supervision, or numerical information on the RUL of each engine block to train a regressor. Hence, it was necessary to look for clusters on the time series previously described (the filtered speed and pressure, plus the nine additional features, such as relative speed, curvature, etc., for each one of them).

Traditional time series clustering techniques can be used to find clusters for different behaviors, partitioning the original data series using different distance or dissimilarity measurements among the series, or features extracted from them. A detailed review of classical techniques for time series clustering can be found in the work by Maharaj et al. [14].

Several classic methods were tested on the current dataset with no satisfactory results [9], being unable to find significant difference between nominal and faulty behavior. Moreover, using supervised deep-learning and information about the early and final stages of the piston life cycle it was possible to separate both stages with more than 97% of accuracy [17], but it was not possible to isolate other degradation stages with more than 65% of accuracy. Consequently, different surveys on deep learning clustering techniques were explored [22, 2, 13, 12], where different methods are proposed; the most common being the combination of autoencoder followed by clustering of the latent space, and transformer-based embeddings also followed by clustering. In the remainder of this section the methods and techniques selected for this work will be summarized. It was decided to use the most promising one for time-series clustering following [12].

3.1 Autoencoder

An autoencoder is a deep neural network designed to learn an efficient representation of the input data, $\mathbf{X}$, by means of an *Encoder*, $E(\mathbf{X})$, that extracts the inputs features to a latent representation, $\mathbf{Z}$, and then uses a *Decoder*, D, which reconstructs $\hat{\mathbf{X}}$ from the encoding $\mathbf{Z}$, while minimizing a loss function (L), which measures the difference between the input ($\mathbf{X}$) and the reconstructed output ($\hat{\mathbf{X}}$). This can be expressed as follows:

$$E : \mathbf{Z} = E(\mathbf{X}) = f(W_e\mathbf{X} + b_e) \quad (1)$$

$$D : \hat{\mathbf{X}} = D(\mathbf{Z}) = g(W_d\mathbf{Z} + b_d) \quad (2)$$

$$\mathcal{L} = ||\mathbf{X} - \hat{\mathbf{X}}||^2 \quad (3)$$

The behavior of the autoencoder can be summarized as:

$$\hat{\mathbf{X}} = D(E(\mathbf{X})) \quad (4)$$

where f and g are activation functions, W_e, W_d are weight matrices and b_e, b_d are the biases.

A Dilated Convolutional Neural Network (DCNN) architecture [11] is employed to implement the autoencoder model for time-series clustering. This architecture leverages variations in the dilation parameter, which are determined by the length of the input time series. Specifically, the dilation increases exponentially: at a rate of 4 for series shorter than 50 data points, and at a rate of 2 for longer series.

3.2 Dimensionality reduction

Uniform Manifold Approximation and Projection (UMAP) [15] is a non-linear stochastic method based on graphs for dimensionality reduction. The algorithm is divided into two main steps. First, it constructs a fuzzy graph between the data points: it computes a k-nearest

neighbor graph and then defines the probability that two points are connected. Next, it creates the low-dimensional embedding by minimizing a cross-entropy loss, which pulls points together or pushes them apart depending on their distances in the high-dimensional space. This optimization is performed using stochastic gradient descent.

Let C be the set of all connections in the graph constructed from the high-dimensional space. Let $w_h(c)$ and $w_l(c)$ denote the weights of a connection c in the high- and low-dimensional spaces, respectively. The cross-entropy is then computed as:

$$\sum_{c \in C} w_h(c) \log(\frac{w_h(c)}{w_l(c)}) + (1 - w_h(c)) \log(\frac{1 - w_h(c)}{1 - w_l(c)}) \tag{5}$$

The objective is to minimize this function. The first term is minimized by increasing $w_l(c)$, meaning the points are close together in the low-dimensional space. The second term is minimized by decreasing $w_l(c)$, which pushes the corresponding points farther apart.

One major advantage of UMAP over other dimensionality reduction methods, such as t-SNE [21], is that, by saving the model parameters, we can obtain the latent representation in the low dimension space for each new block, without recomputing the UMAP model on the whole training set.

UMAP includes several hyperparameters, such as the number of neighbors, the distance metric used to construct the graph, the minimum allowed distance among points in the embedding, and the target dimensionality.

In this work, UMAP will be applied to the latent space $\mathbf{Z}$ from the autoencoder, to reduce its initial dimensionality, which is 50.

3.3 Clustering method

Fuzzy c-means (FCM) [18, 5] is a clustering technique that allows the probabilistic assignment of an element to a set of clusters based on a minimization function. While there are several proposals for optimization criteria, the most popular to date is associated with the least square error function (6).

$$J(W, C) = \sum_{k=1}^{N} \sum_{i=1}^{C} w_{ik}^{m} \|x_i - c_j\|^2 \tag{6}$$

Where N is the number of samples; C is the number of clusters, w_{ik} is the degree to which element x_i belongs to the cluster c_j, m is the fuzzy factor, strictly greater than one, and $\|x_i - c_j\|^2$ is understood as the euclidean distance that separates the element x_i from the centroid of the cluster c_j. Optimization is subject to the constraint $\sum_{i=1}^{C} w_{ik} = 1 \quad \forall k \in [1, ..., N]$ and solved using Lagrange multipliers. It is worth mentioning that if m tends to one, the optimal partition is increasingly closer to an exclusive partition (K-means), and when it tends to infinity the optimal partition approaches a matrix with all its values are equal to $1/C$. Usually m takes values in the range between $[1 - 30]$.

In this work fuzzy c-means will be applied to the reduced space obtained after applying UMAP to the autoencoder latent space.

3.4 LOWESS

In order to plot the evolution of the cluster assignment obtained after applying fuzzy c-means for each new engine, which will be a really noisy value, LOWESS was selected to smooth the visualization.

LOWESS, which stands for Locally Weighted Scatterplot Smoothing [6], is a robust locally weighted regression method for smoothing scatterplots. Local regression is a nonparametric approach for estimating a regression function or a surface.

Let's say that we have a collection of points $(x_i, y_i), i = 1, \ldots, n$, in a two dimensional space, in which the fitted value at x_k is the value of a polynomial fit to the data using weighted least squares, where the weight for (x_i, y_i) is larger if x_i is closer to x_k and is smaller if it is not. This fitting procedure is used to deal with deviant points distorting the smoothed points.

In this work LOWESS was applied to the projection on a two, or three, dimensional space for the UMAP output after applying fuzzy c-means.

4 Estimating the wear condition of the piston head with a Fuzzy Progression Index and results on the case study

This section presents a methodology capable of estimating the degree of wear of a piston head throughout its life cycle. For this purpose, a fuzzy index, Fuzzy Progression Index, FPI, was proposed. FPI represents the wear state of the piston head. Information on the piston wear state is not available in the training data, and there are only data related to the number of injections performed by the piston in its life cycle. For these reasons, advanced clustering techniques were chosen to obtain the FPI of a piston throughout its life cycle. This section is devoted to presenting the proposed methodology from the raw data, indicating the processing steps, and showing the experimental results obtained on the training set. The next section will present the methodology to estimate the RUL of a piston from its FPI.

4.1 Experimental setup

The methodology for obtaining the FPI that shows the evolution of piston head wear consists of three sequential stages, which are represented in Figure 4.

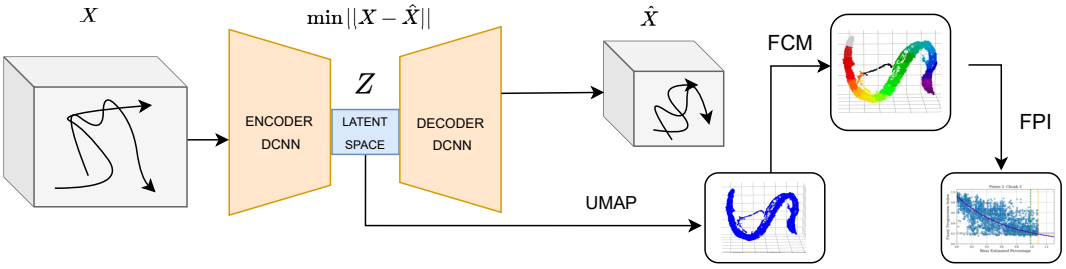


Figure 4 Proposed methodology to estimate the piston wear state by means of FPI from the data.

In the first stage, a DCNN-based autoencoder architecture is used to construct a latent representation of the input data. The autoencoder, using dilated convolutions, is trained to minimize the reconstruction error, thereby ensuring that the latent space effectively captures the most relevant features of the original high-dimensional dataset, including local and contextual patterns that might otherwise be lost with standard convolutional layers.

In the second stage, the dimensionality of the latent space generated by the autoencoder is further reduced using Uniform Manifold Approximation and Projection, which provides a compact and computationally tractable representation of the data while preserving its

underlying topological structure. In this study, the UMAP projection will generate a three-dimensional region of high point density. The projection of the head piston for each newly generated engine block shifts from one end of the region to the other, as the number of injected blocks increases. Unfortunately, the displacement within the region is noisy, and there is no simple correlation between the location of the projection in one region and the degree of wear.

Finally, in the third stage, due to the noisy evolution of the piston head projection, fuzzy clustering is performed on the reduced latent space using the FCM algorithm. This step groups the engine blocks into distinct overlapping clusters. The trend of assignment of each component to different clusters, based on the probabilities provided by the FCM, can be used to define the Fuzzy Progression Index as follows:

$$FPI = \frac{1}{C} \sum_{i=1}^{C} i \cdot w_i \tag{7}$$

Where C is the number of clusters, $i = 1$ is the cluster associated with the initial state, $i = C$ is the cluster representing the final state, and w_i is the degree of membership to cluster i.

FPI is correlated to the remaining percentage of life of the piston head, but unfortunately it also presents a high variance. Therefore, LOWESS [6] method will be used to smooth the data in order to obtain the trend of the FPI. The resulting curve serves as an indicator of the piston life trend. Using the probabilistic nature of FCM assignments, the approach captures the gradual transition between health states.

Table 1 summarizes the parameter configurations explored for the DCNN, UMAP, and FCM techniques. For the FCM method, the values listed represent the various configurations tested during the optimization process, which aims to improve the accuracy and stability of the cluster assignments across the component data. By systematically adjusting these parameters, the model seeks to enhance the clarity of the clustering structure, ensuring that the transitions between health states are captured more effectively. This cluster representation can be used to obtain a more robust representation of the component's behavior over time, providing a reliable foundation for subsequent analyses and decision-making processes. This idea will be explained in Section 5.

4.2 Results for the FPI computation and cluster assignment on the case study

To validate the proposed approach, the dataset was partitioned into training and test subsets, following an 80-20% split, to ensure a fair comparison. The reader should notice that the split was made at the piston level, i.e. we used for training all the measurements from 80% of the available pistons, and 20% of the pistons were used for test, rather than doing a random 80-20% split of available measurements for each piston from the full set of pistons data. As a result, a $[13 - 3]$ piston split forms the basis for this work.

Several model configurations were explored for the training set. The upper part of Figure 5a shows the resulting structure of the latent space generated by the autoencoder in three dimensions for the training set; this structure can be interpreted as the characteristic life cycle of a piston.

Once the structure was fixed, various numbers of clusters were explored, ranging from five to twenty-five. This range was selected based on preliminary analyses, which indicated that fewer than five clusters failed to capture the necessary complexity of the system degradation patterns, while more than twenty-five clusters led to over-fitting and decreased generalization

Table 1 Hyperparameters and values explored.

Autoencoder	
Input sequence length	20
Latent space dimension	50
Reconstruction loss functions	MSE
DCNN parameters	
Dilation	1, 4, 16
Kernel size	3
Number of filters	40
Number of conv. layers per dilation value	2
Number of filters in the last convolution	320
UMAP	
Number of output dimensions	3
FCM	
Number of clusters	5–25
Fraction of the data used when estimating each y value	0.3, 0.5, 0.7, 0.9
Fuzzy factor	2, 2.5

performance. From these experiments, the optimal number of clusters was determined to be 23, as this configuration achieved the best performance in the FCM method of FPI optimization. The criterion for optimizing the number of clusters was to minimize the prediction error at 15% of the remaining piston head life, along with the variance of these estimates in the training set. The reader should notice that, in the absence of failure models, the service life is estimated in terms of the percentage of the piston lifespan. In this case study, pistons from different manufacturers were used. Although all of them showed similar wear patterns over their service life, the expected lifespan is different among manufacturers. To obtain an estimate of the RUL in temporal terms, it would be necessary to know the average service life provided by each manufacturer for each type of piston.

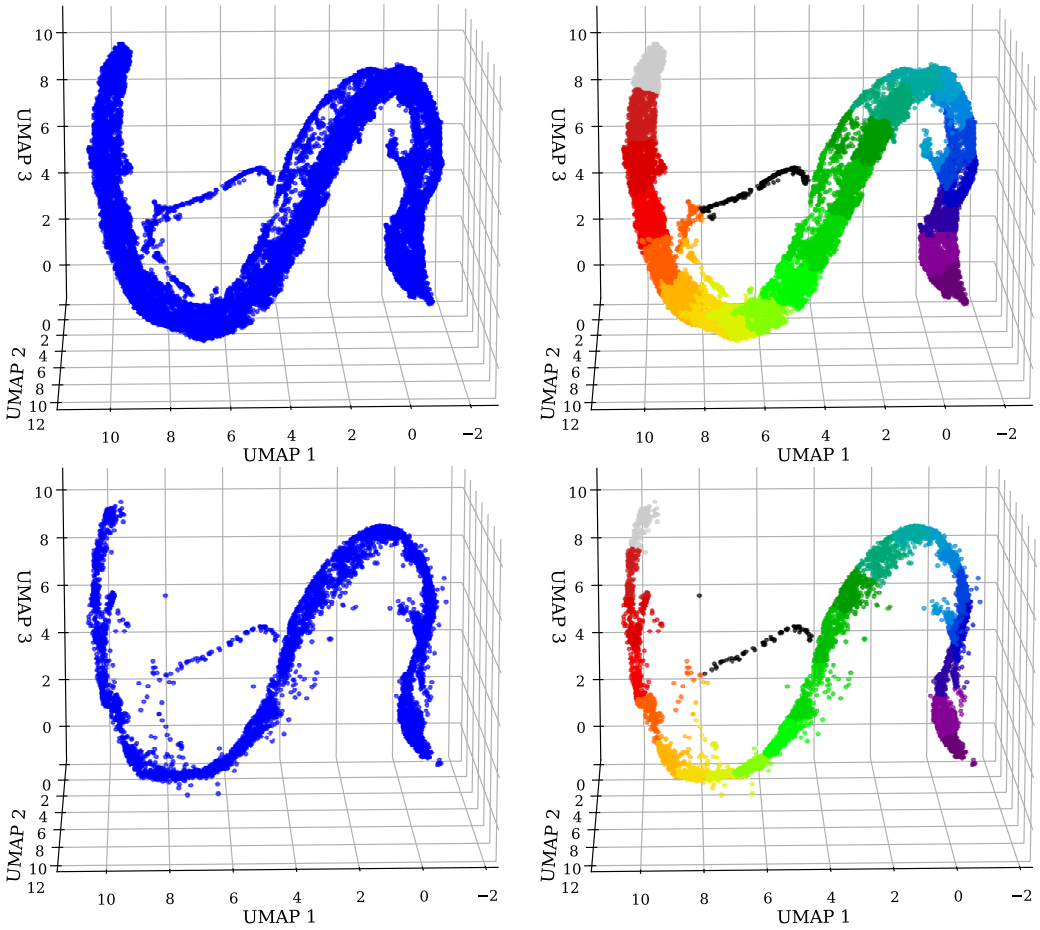
The evolution of the piston head degradation over the piston life cycle for the training set is shown in the upper part of Figure 5b, where different colors have been assigned to the clusters, temporally ordered: with the gray color representing the initial cluster, and the violet color representing the final cluster. As it can be seen in the lower part of both Figures 5a and 5b, the results are very similar for the test dataset.

Using that fuzzy cluster results, the FPI values were generated for each engine block. Figure 6 shows the evolution of these FPI values (represented as points) generated by six pistons of the training set, and the LOWESS regression prediction for the pistons EOL.

5 A proposal for RUL estimation of the piston head

The values of LOWESS regression for each piston FPI have been valuable in determining the optimal number of clusters. However, due to its local nature, LOWESS regression is not a useful tool for predicting RUL outside of its training range, and it does not perform a reliable extrapolation outside of that training range. This was a major handicap for its online use, as the fitting is performed with the available parts, not with the complete curve.

To enhance the predictive capability of the previously calculated life trend, an additional step is proposed, consisting of the use of an exponential approximation. By fitting an exponential curve to the observed trend, using the number of clusters selected in the



(a) Three-dimensional latent space inferred by the DCNN after applying UMAP over the training (top) and test (bottom) sets.

(b) Distribution of clusters generated by *fuzzy c-means* over the latent space inferred by the DCNN after applying UMAP on the training (top) and test (bottom) sets.

Figure 5 Comparison between the raw latent space (left) and the clustered latent space (right) using UMAP representations for both training and test sets.

experiments described above, the approach aims to capture the underlying degradation dynamics in a smoother way. This method provides a compact mathematical representation of the FPI evolution, and facilitates the projection of future behavior.

5.1 A data-based model to predict the RUL from the Fuzzy Progression Index

To obtain an exponential data-based model, an exponential function for each piston in the training set was individually fit, and a threshold of this exponential function at the end of the lifetime was obtained. From these thresholds, the mean threshold and its variance were calculated. Next, the mean exponential function of the training set was computed. This was the function used as a model of the FPI in the testing phase. Subsequently, when evaluating the test set, a correction was applied to this mean exponential trend by incorporating the already observed residual values for each individually generated part and piston. This adjustment aims to refine the prediction by considering the previous piston

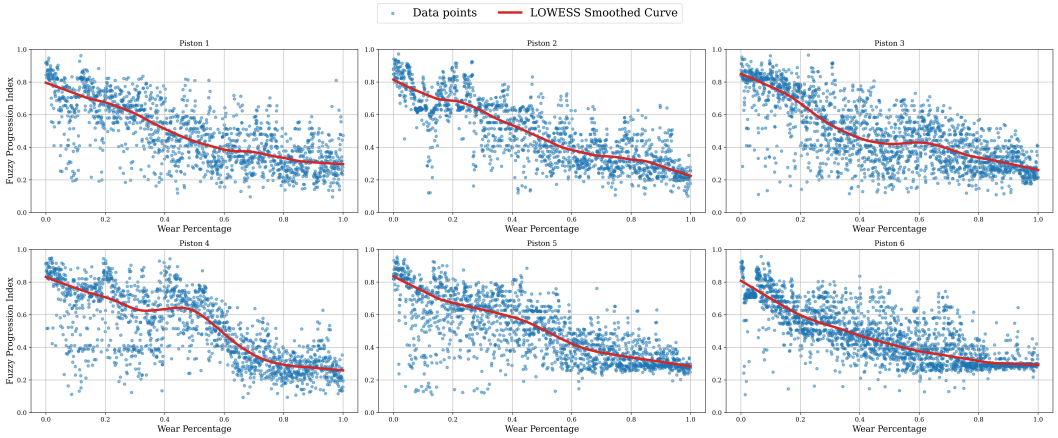


Figure 6 Sample of the trend calculated with LOWESS for six pistons from the training set.

behavior, which, in general, shows deviations that were not captured during the initial adjustment process. This adjustment improves the accuracy of the RUL estimation of the model through unobserved data.

5.2 Experimental results in the case study

Figure 7 presents the progressive estimation of the percentage of remaining parts for each of the test pistons at various points throughout their life cycle. The FPI value of each produced part is shown as a blue point, while the average trend for the exponential function derived from the training set is shown as a dotted red line. The dark blue line represents the corrected trend, obtained by adjusting the average exponential curve based on the residual values associated with each part for the current piston. Additionally, the orange and green vertical lines indicate the estimated range marking the end of the life cycle; specifically, these points correspond to the intersection of the corrected trend and the threshold value, considering both plus and minus one standard deviation. This visualization provides a clear and interpretable summary of the model's performance, highlighting the variability inherent in the predictions and offering a practical estimation window for the remaining useful life of the components.

In particular, it is observed that each piston in the test set falls into one of the three possible estimation outcomes: underestimation, where the model predicts less remaining life than is actually present; overestimation, where the predicted remaining life exceeds the true value; and accurate prediction, where the estimation closely matches the actual remaining life. This variability highlights both the strengths and limitations of the proposed methodology, emphasizing the importance of understanding the conditions under which the model reliably performs versus when it tends to deviate. Such insights are critical for refining the approach and ensuring more consistent predictive performance across diverse operating scenarios.

In Table 2, the predicted threshold values for each chunk are presented, with the rows shaded in grey specifically highlighting the chunks that correspond to the actual EOL cycle for each piston in the test set.

Figure 7 and table 2 shows that only the third piston EOL is included in the estimated interval, while piston 1 lasts an additional 18% on the predicted Upper Threshold, and piston 2 lasts about 9% under the Lower Threshold. However, it is not easy to interpret these results as under or over estimations for the RUL. The data used for training and testing

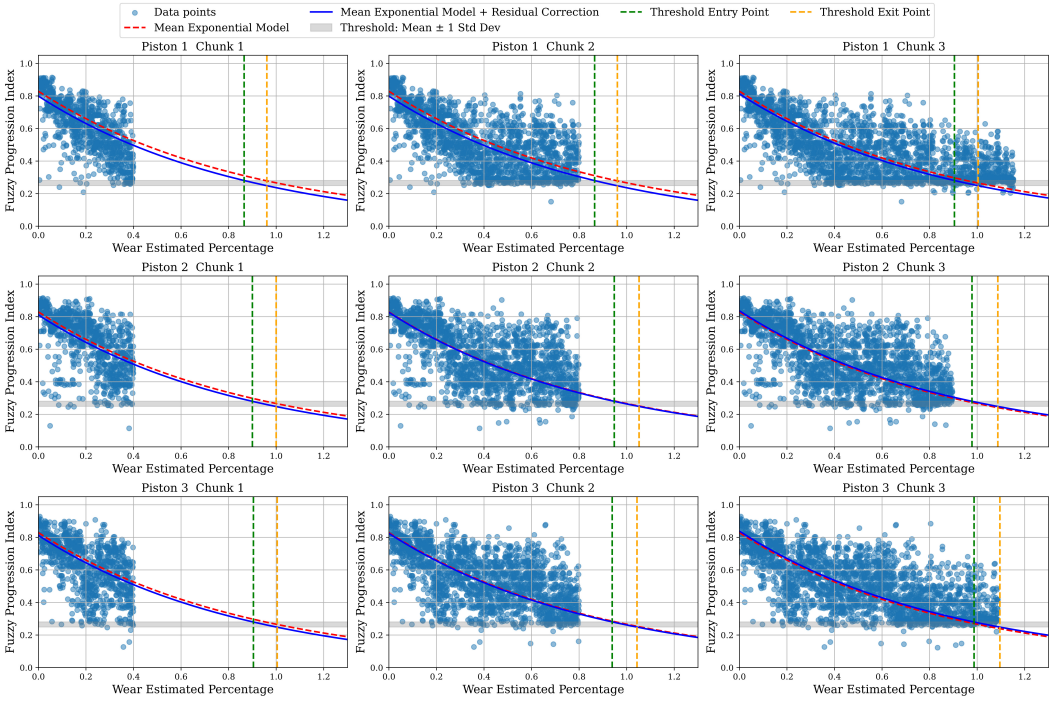


Figure 7 Progressive representation of the RUL estimate using the exponential approximation of the life cycle trend line of the pistons in the test set.

Table 2 Threshold Input and Output Values per Piston and Chunk.

Piston	Chunk	Input Threshold	Output Threshold
1	1	0.865	0.961
1	2	0.865	0.961
1	3	0.904	1.004
2	1	0.900	1.000
2	2	0.948	1.052
2	3	0.978	1.087
3	1	0.904	1.004
3	2	0.939	1.043
3	3	0.987	1.096

were not obtained in a controlled environment where the maximum acceptable RUL was pursued. These data come from standard factory operation where pistons are changed on a preventive maintenance agenda, that requires changing the piston head after a fixed number of operations. Moreover, the number of performed operations usually changes due to other operational criteria, such as the convenience of performing a programmed stop to change the piston head, or the detection of anomalies after the injection stage.

Even with these caveats, the proposed approach is considered as a potential valuable tool for plant operators and managers, to perform predictive maintenance of the piston head.

6 Discussion and Conclusions

This work has developed a tool to estimate the Remaining Useful Life of the piston head in an injection machine.

The problem of estimating the RUL has been divided into two stages. First, to find a useful wear index, the Fuzzy Progression Indicator. Second, to use the actual index of each piston to predict the RUL.

The Fuzzy Progression Indicator is a first step for estimating the EOL and the RUL, thus being the bases for a predictive maintenance policy. This work has proposed to estimate the FPI in a three-stage data-driven methodology, that has been validated by experimental results. The first stage of the methodology builds a latent representation from the available data using a state-of-the-art autoencoder architecture, DCNN. The second stage requires a reduction of the latent space dimension. UMAP has been shown to be able to find a three-dimensional projection related to the piston wear and tear state. The third stage relies on fuzzy c-means to cluster the projected space, looking for overlapping regions that can be related to different wear states. The membership vector of a projected point to the set of clusters allowed to obtain an index that ideally moves from the value of $1/C$, for a new piston, with C the total number of clusters, to a value of 1, for a totally worn piston. However, this index shows high variance and cannot be directly used to estimate the RUL.

To estimate the RUL, this work proposed to fit an exponential average function to the data of the training set. This function smooths out the highly variable FPI while still providing an acceptable forecast of the RUL. The prediction of the exponential function is corrected with the residuals of the actual index for the piston, which takes care, to some degree, of the deviation observed in a new piston from the expected behavior.

Currently, piston maintenance is preventive, and the piston is replaced after a pre-set number of injection operations, except when the piston wears out or fails early. The capability to predict the Remaining Useful Life of the piston introduces two significant operational benefits. Firstly, if piston degradation is detected prior to reaching the currently defined injection cycle threshold, maintenance activities can be proactively scheduled. This action would avoid unplanned downtime, which according to plant personnel, typically results in at least 45 minutes of machine unavailability in the event of an unexpected piston replacement. In contrast, planned maintenance can be executed within approximately 15 minutes. This results in a net gain of 30 minutes of productive machine time. Secondly, by extending piston usage beyond the conservative pre-set injection limit, based on actual condition rather than fixed intervals, spare part consumption can be optimized. This approach contributes to a reduction in the lifecycle cost of the component by decreasing its amortization per injection cycle.

Although further research is still needed, the current results are considered acceptable by plant managers in the sense that a plant preventive maintenance policy based on this tool can be tested.

In the immediate future we plan to validate the approach with a new batch of production data, which will include hundreds of life cycles to train and test. We will then implement a formal scheme to manage uncertainty, potentially considering unscented Kalman filter or particle filtering techniques.

References

1 Gerardo Acosta, Carlos Alonso González, and Belarmino Pulido. Basic tasks for knowledge-based supervision in process control. *Engineering Applications of Artificial Intelligence*, 14(4):441–455, 2001.

2 Nagdev Amruthnath and Tarun Gupta. A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance. In *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 355–361, 2018. doi:10.1109/IEA.2018.8387124.

3 Vepa Atamuradov, Kamal Medjaher, Pierre Dersin, Benjamin Lamoureux, and Noureddine Zerhouni. Prognostics and health management for maintenance practitioners-review, implementation and tools evaluation. *International Journal of Prognostics and Health Management*, 8(3):1–31, 2017.

4 Piero Baraldi, Francesco Cadini, Francesca Mangili, and Enrico Zio. Model-based and data-driven prognostics under different available information. *Probabilistic Engineering Mechanics*, 32:66–79, 2013.

5 James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 2013.

6 William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979. doi:10.1080/01621459.1979.10481038.

7 Miguel Cubero, Diego Garcia-Alvarez, Luis Ignacio Jiménez, Daniel López Gómez, Belarmino Pulido, and Carlos Alonso González. Deep-learning clustering to assess the health state in a die-casting process in the automotive industry. In *6th "International Conference on Control and Fault-Tolerant Systems (SYSTOL)"*, 2025.

8 Matthew J Daigle and Kai Goebel. A model-based prognostics approach applied to pneumatic valves. *International Journal of Prognostics and Health Management Volume 2 (color)*, 84, 2011.

9 David Garcia de Vicuña Lopez de Ciordia. *Application of time-series clustering for Factory 4.0 (in Spanish)*. Master's thesis, Escuela de Ingenieria Informatica de Valladolid, Universidad de Valladolid, July 2024. Supervisors: B. Pulido, C. Alonso-Gonzalez.

10 Alberto Diez-Olivan, Javier Del Ser, Diego Galar, and Basilio Sierra. Data fusion and machine learning for industrial prognosis: Trends and perspectives towards industry 4.0. *Information Fusion*, 50:92–111, 2019. doi:10.1016/J.INFFUS.2018.10.005.

11 J. Y. Franceschi, A. Dieuleveut, and M. Jaggi. Unsupervised scalable representation learning for multivariate time series. *Advances in Neural Information Processing Systems*, 32, 2019.

12 B. Lafabregue, J. Weber, P. Gançarski, and G. Forestier. End-to-end deep representation learning for time series clustering: a comparative study. *Data Mining and Knowledge Discovery*, 36(1):29–81, 2022. doi:10.1007/S10618-021-00796-Y.

13 Han Li, Wei Zhao, Yuxi Zhang, and Enrico Zio. Remaining useful life prediction using multi-scale deep convolutional neural network. *Applied Soft Computing*, 89:106113, 2020. doi:10.1016/J.ASOC.2020.106113.

14 Elizabeth Ann Maharaj, Pierpaolo D'Urso, and Jorge Caiado. *Time series clustering and classification*. Chapman and Hall/CRC, 2019.

15 Leland McInnes, John Healy, Nathaniel Saul, and Lucas Großberger. UMAP: uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. doi:10.21105/JOSS.00861.

16 Bojana Nikolic, Jelena Ignjatic, Suzic Nikola, Branislav Stevanov, Aleksandar Rikalovic, et al. Predictive manufacturing systems in industry 4.0: Trends, benefits and challenges. In *Proceedings of 28th DAAAM International Symposium on Intelligent Manufacturing and Automation*, pages 796–802. DAAAM International, Vienna, Austria, 2017.

17 Anibal Hernando Novo. *Health state estimation for an injection machine using deep-learning (in Spanish)*. BSc thesis, Escuela de Ingenieria Informatica de Valladolid, Universidad de Valladolid, July 2025. Supervisors: D. Garcia, B. Pulido.

18 Enrique H Ruspini. Numerical methods for fuzzy clustering. *Information Sciences*, 2(3):319–350, 1970. doi:10.1016/S0020-0255(70)80056-1.

19 Girish Kumar Singh et al. Induction machine drive condition monitoring and diagnostic research – A survey. *Electric Power Systems Research*, 64(2):145–158, 2003.

20 George J Vachtsevanos, Frank Lewis, Michael Roemer, Andrew Hess, and Biqing Wu. *Intelligent fault diagnosis and prognosis for engineering systems*, volume 456. Wiley Online Library, 2006.

21 Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11):2579–2605, 2008.

22 Z. Wang, W. Yan, and T. Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

23 Shu-Xin Zhou. The practical applications of industry 4.0 technology to a new plant for both manufacturing technique and manufacturing process in new product introduction. *IEEE Access*, 9:149218–149226, 2021. doi:10.1109/ACCESS.2021.3124373.

24 Enrico Zio. Prognostics and health management of industrial equipment. *Diagnostics and prognostics of engineering systems: methods and techniques*, pages 333–356, 2013.

25 Enrico Zio. Chapter 8 – Data-driven prognostics and health management (PHM) for predictive maintenance of industrial components and systems. In Curtis Lee Smith, Katya Le Blanc, and Diego Mandelli, editors, *Risk-Informed Methods and Applications in Nuclear and Energy Engineering*, pages 113–137. Academic Press, 2024.

One-Shot Learning in Hybrid System Identification: A New Modular Paradigm

Swantje Plambeck
Hamburg University of Technology, Germany

Maximilian Schmidt
Hamburg University of Technology, Germany

Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse, CNRS, France

Goerschwin Fey
Hamburg University of Technology, Germany

Abstract

Identification of hybrid systems requires learning models that capture both discrete transitions and continuous dynamics from observational data. Traditional approaches follow a stepwise process, separating trace segmentation and mode-specific regression, which often leads to inconsistencies due to unmodeled interdependencies. In this paper, we propose a new iterative learning paradigm that jointly optimizes segmentation and flow function identification. The method incrementally constructs a hybrid model by evaluating and expanding candidate flow functions over observed traces, introducing new modes only when existing ones fail to explain the data. The approach is modular and agnostic to the choice of the regression technique, allowing the identification of hybrid systems with varying levels of complexity. Empirical results on benchmark examples demonstrate that the proposed method produces more compact models compared to traditional techniques, while supporting flexible integration of different regression methods. By favoring fewer, more generalizable modes, the resulting models are not only likely to reduce complexity but also simplify diagnostic reasoning, improve fault isolation, and enhance robustness by avoiding overfitting to spurious mode changes.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Computing methodologies → Symbolic and algebraic algorithms; Computing methodologies → Learning paradigms; Computing methodologies → Modeling methodologies

Keywords and phrases Hybrid System, Model Learning, Symbolic Regression

Digital Object Identifier 10.4230/OASIcs.DX.2025.7

Supplementary Material
Software (Source Code): `https://github.com/TUHH-IES/SymbolicRegression4HA` [13]
archived at `swh:1:dir:00ff8c08db289ab78af743dd7ce6b71dceccb37c`

Funding This work is funded by BMBF project AGenC no. 01IS22047A.

1 Introduction

Hybrid systems integrate continuous-time dynamics with discrete transitions and serve as a fundamental modeling paradigm for a broad class of cyber-physical, manufacturing, and embedded systems [1, 8]. The identification of hybrid systems from data is particularly challenging due to the inherent coupling between discrete mode transitions and continuous behavior. However, accurate and compact abstract models are essential for tasks such as system verification, control synthesis, and fault diagnosis. Compactness is especially beneficial for diagnosis, as it can streamline fault isolation, lower the computational load of diagnostic reasoning, and mitigate the risk of ambiguity or overfitting to spurious or non-informative mode changes.

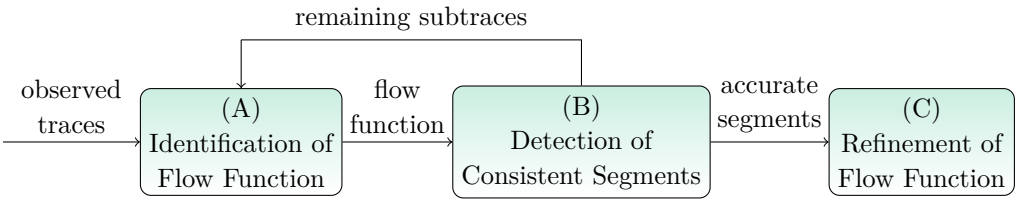


Figure 1 Illustration of the Proposed Learning Paradigm.

A hybrid system comprises a finite set of discrete modes, each associated with a distinct continuous flow function. Transitions between modes are governed by discrete events, defined through guard conditions, which depend on the state of the system or external inputs [3].

Data-driven identification aims to reconstruct the underlying hybrid model from observed traces of the system. Traditional approaches to hybrid system identification typically adopt a stepwise procedure [14]. First, the observed traces are segmented and grouped into regions presumed to be generated by the same dynamic mode. Subsequently, regression techniques are applied to fit a continuous flow function to each region representing the grouped segments. However, this decoupled approach suffers from several limitations. Independent optimization of the segmentation and regression often leads to inconsistencies, as the learned components do not account for their mutual dependencies. In particular, inaccurate segmentation can degrade the quality of the estimated dynamics, resulting in inaccurate models and poor performance.

In our previous work, we propose a learning approach using symbolic regression [15, 16]. This approach identifies segments directly based on changes in the system dynamics. For this purpose, symbolic regression is used, which allows adapting to given dynamics in a flexible way by iterating on a flow function. Now, we present a new procedure that elaborates on this idea comprising the overall learning paradigm of hybrid system identification. A major advantage of our previous work is the detection of transition points based on a first estimate of the flow function for the system dynamics. This encourages the development of a new learning paradigm that, instead of resetting an identification, leverages a flow function identified on a given segment to segment the remaining observed traces in one shot by finding the consistent segments. In contrast to the previous work, this paradigm is independent of the regression method used to identify the flow functions, i.e., the method does not restrict to symbolic regression as presented in Plambeck et al. [15, 16].

We present a discussion on the learning process for hybrid systems facing interdependencies in the learning process. With that, we provide a formalization of the underlying optimization problem of data-driven identification for hybrid systems and revisit the traditional idea of stepwise learning in this context. In addition, we propose an iterative learning paradigm for hybrid systems that takes account the original optimization problem. Figure 1 illustrates the proposed procedure. The approach begins by identifying a continuous flow function from an initial segment of a trace (A). The flow function from Step (A) is then evaluated against the remaining observed traces (B). Segments where the flow function exhibits sufficient predictive accuracy are consistent with respect to the flow function and they are grouped into a mode (C), while the remaining subtraces are treated as candidates for further mode identification (A). This procedure is repeated until the complete trace is adequately explained by one of the modes in the model. Step (C) finally refines the found flow function on all consistent segments of the dynamic mode. To determine an appropriate initial segment, we

employ a windowing strategy that increases the length of the segment until the error of the prediction by the corresponding flow function is above a specified threshold. This strategy maximizes segment size under a fixed accuracy constraint.

The proposed method offers several advantages over traditional techniques. By jointly optimizing segmentation and model identification in an iterative loop, the algorithm exploits the interdependencies among the segments of traces and flow functions to guide the learning process. The model is constructed incrementally, with new modes introduced only when existing ones fail to capture the observed dynamics. This results in compact models characterized by a minimal number of modes as well as large and coherent segments.

Our empirical results demonstrate that our approach has a flexible and modular design for the integration of regression methods in the identification of flow functions. In addition, our learning strategy identifies models with a smaller number of modes compared to a traditional learning strategy, which is very well illustrated on a bouncing ball example.

The remainder of the paper is structured as follows: Section 2 reviews related work on hybrid system identification. In Section 3, we introduce the necessary preliminary concepts and notation. In Section 4, we present our learning approach in detail. Section 5 presents an empirical evaluation of the proposed approach. We conclude the paper in Section 6.

2 Related Work

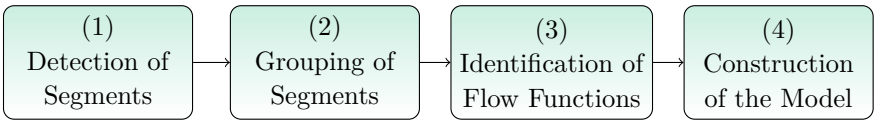


Figure 2 Traditional Approach to Hybrid Systems Identification.

Most existing approaches to hybrid system identification adopt a sequential pipeline, as illustrated in Figure 2. In contrast to our approach, these methods defer mode identification – i.e., the estimation of flow functions – after the segmentation of traces and the grouping of segments. This introduces two intermediate representations (segments and segment groups) that are constructed without directly considering the final goal of mode identification.

In this traditional process, the first step is the detection of transition points (1), which partition the observed traces into segments. For the detection of transition points, existing methods use a similarity measure, based on the distance between samples [2, 19], in the frequency domain [11], or based on slices of observations [21]. However, these methods often overlook the fact that complex trajectories generated by the same dynamics can occupy different regions of the state space. In contrast, our approach avoids explicit transition detection and instead identifies segments based on the consistency with a belief of the underlying flow function.

The second step (2) involves grouping the segments that are presumed to belong to the same mode. Common techniques for this step include clustering [2] and similarity-based grouping [18]. Ly et al. [9] propose a clustering approach based on symbolic regression to infer the structure of modes.

Finally, in step (3), flow functions are then identified for each group of segments. This is typically done using regression techniques, including linear regression [21], polynomial regression [18], or neural networks [11]. Often, existing methods restrict to a specific class of functions, e.g., linear differential equations [21] or polynomials [18], or types of regression models such as neural networks [11]. Several approaches are further limited to a single

regression method, such as linear regression or symbolic expressions [9]. Our method, on the contrary, is agnostic to the choice of the regressor and can incorporate various regression models, as demonstrated in our evaluation.

The final step (4) is the construction of the complete hybrid system. This is done by combining the identified flow functions into modes and forming transitions between them. Transition conditions are typically learned in a separate step [18] or identified during the construction of the hybrid system [11, 5].

3 Preliminaries

This section introduces the foundational concepts used throughout the paper. We start with the definition of hybrid systems, which are the target of our learning approach. We then introduce the system under learning and the traces, that we observe on this system.

3.1 Hybrid Systems

Hybrid systems combine continuous dynamics with discrete transitions, representing complex behaviors that are common in many real-world systems.

▶ **Definition 1** (Hybrid System [3]). *A Hybrid System* Γ *is defined by a 5-tuple* $(X, Q, \mathcal{F}, \Sigma, \mathcal{T})$ *where:*

- $X = \{x_1, x_2, \ldots, x_n\} = I \cup S$ *is the set of system variables, consisting of input variables* I *and state variables* S*.*
- Q *is the finite set of discrete modes.*
- $\mathcal{F}$ *is the set of flow functions. Each flow function* $f_q \in \mathcal{F}$ *defines the continuous flow, i.e., dynamics within mode* $q \in Q$ *by specifying the outputs as a function of the current state and inputs, i.e.,* $O = f_q(S, I)$*, where* $O \subseteq S$ *are the output variables of the system.*
- Σ *is a set of events. Each event is guarded by a condition* $c : X \to \{true, false\}$ *on the variables in* X*. An event is active if the condition evaluates to true.*
- $\mathcal{T} : Q \times \Sigma \to Q$ *defines transitions between modes* Q*. A transition is triggered if the corresponding event* $\sigma \in \Sigma$ *is active.*

This definition combines continuous behavior with discrete transitions. The continuous behavior is governed by mode-specific flow functions, whereas transitions between modes are triggered by the guard conditions on system variables, including external control signals.

Throughout this work, we assume that flow functions are time-invariant, i.e., the functions in $\mathcal{F}$ depend only on the current state and input values, not explicitly on the time. However, time-dependent behavior can be modeled by introducing time-shifted versions of state variables as an additional input to the flow function.

3.2 System & Observations

The system under learning is observed through a finite collection of execution traces $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_m\}$. Each trace $\mathcal{O}_j$ consists of a time series of sampled variable vectors from X. Formally, $\mathcal{O}_j[i] = [x_1, x_2, \ldots, x_n]$ denotes the ith sample in trace $\mathcal{O}_j$, where $j = 1, \ldots, m$ and $i = 1, \ldots, k_j$ with k_j being the length of the jth trace.

The goal is to identify a hybrid system Γ that explains the observed traces, capturing both the continuous dynamics and the discrete mode transitions of the underlying system behavior.

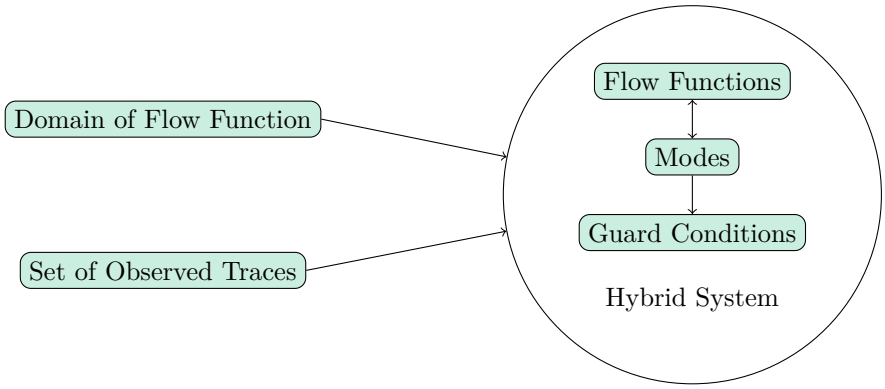


Figure 3 Illustration for the Problem of Hybrid System Identification.

4 A New Paradigm for Hybrid System Identification

Identifying hybrid systems from data is challenging due to the inherently complex structure. A complete model must capture discrete modes, continuous dynamics (flow functions), and the conditional transitions that govern mode switching. We analyze the goal of hybrid system identification and propose a formalization of the learning problem. Afterward, we discuss the limitations of traditional approaches to hybrid system identification with respect to the original learning problem. Finally, we present our new paradigm for hybrid system identification that integrates the learning of flow functions and segmentation into a unified framework.

4.1 Problem Definition and Objectives for Hybrid System Identification

The learning process for hybrid system identification starts with a set of observed traces. Additionally, a domain Φ of candidate flow functions is given at the beginning of the learning process. The set of candidate flow functions Φ contains all flow functions that can be used to model the continuous dynamics of the system and is, e.g., defined by the application. The goal is to identify a hybrid system Γ that explains the observed traces using a minimal number of flow functions. As illustrated in Figure 3, the task is framed as a multi-objective optimization problem: minimizing the model complexity (e.g., the number of modes or flow functions) while maximizing the accuracy of the learned system on the observed data. We define a required accuracy by a threshold ϵ for the deviation between the observed traces and the identified hybrid model. This reduces the learning problem to a minimization of the number of flow functions under this accuracy constraint:

$$\min_{\substack{\mathcal{F}\subset\Phi,\\ \mathcal{S}\in\mathcal{P}(\mathcal{O})}} |\mathcal{F}| \quad \text{s.t.} \quad E(\mathcal{F},\mathcal{S}) \leq \epsilon, \tag{1}$$

where E denotes an error function that evaluates the deviation between the behavior found from the flow functions $\mathcal{F}$ on the segments $\mathcal{S}$ composing a segmentation from the set $\mathcal{P}(\mathcal{O})$ of all possible segmentations of the observed traces $\mathcal{O}$. In this formalization, a minimal exact solution, i.e., $\epsilon = 0$ exists only if the observed traces are free of noise and the domain of candidate flow functions Φ covers the original system dynamics.

4.2 Traditional Identification of Hybrid Systems

The traditional approach to learning hybrid systems uses a sequential procedure as shown in Figure 2:

1. Transition detection: Identify potential transition points in each trace.
2. Segment grouping: Cluster segments between transitions into groups that are assumed to represent the same mode.
3. Flow function identification: Learn one flow function per segment group.
4. Model construction: Construct the hybrid system by combining the learned functions and define transitions.

In this approach, the individual steps are independent of each other and can be formulated as follows. First, the set $\mathbf{t}$ of transition points is identified as

$$\mathbf{t} = \{(j, i) \mid \varphi(\mathcal{O}_j[i]) \text{ is true}\}, \tag{2}$$

where φ is a predicate for transition detection. Every transition point is represented by a tuple (j, i) with j identifying the trace O_j and i giving an index i in this trace. Using the transition points, all traces are cut into segments $\mathcal{S}$:

$$\mathcal{S} = \bigcup_{j \in [1,\dots,m]} \{\mathcal{O}_j[i_k : i_{k+1}] \; \forall \; [(j, i_k), (j, i_{k+1})] \in \mathbf{t}\}, \tag{3}$$

where $\mathbf{t}$ is sorted such that $i_0 < i_1 < \cdots < i_{k_j}$. $\mathcal{S}$ is one possible segmentation of the traces, i.e., $S \in \mathcal{P}(\mathcal{O})$, based on the identified transition points. Segments form a group g, if they are sufficiently similar, i.e.,

$$g = \bigcup \{\hat{\mathcal{S}} \subseteq \mathcal{S} \mid \sigma(\hat{\mathcal{S}}) \geq \tau\}, \tag{4}$$

where σ is a similarity function used to compare segments and τ is a similarity threshold. Note that similar dynamics do not necessarily create similar traces, which is one issue that our approach addresses. Every segment is assigned to a single group only, i.e., $g_i \cap g_j = \emptyset \; \forall i \neq j$. The groups form the set

$$G = \{g_1, \dots, g_{|\mathcal{F}|}\}, \tag{5}$$

such that every group contains the learning data for one flow function. The set of flow functions $\mathcal{F}$ is found as follows:

$$\mathcal{F} = \{\arg\min_{f \in \Phi} e(f, g) \; \forall g \in G\}, \tag{6}$$

where $e(f, g)$ is the approximation error of function f on group g, e.g., the mean squared error between the output of the flow function and the observed output on the learning data.

The process defined by Equation (2) to Equation (6) is problematic as the identification of flow functions is postponed to the last step. Segments and groups form intermediate learning results that are not informed by the final goal of identifying accurate and compact flow functions, which is not inline with the original learning problem in Equation (1). This separation of concerns introduces strong interdependencies: transition points are only meaningful in light of changes in dynamics and accurate flow functions can only be learned from meaningful segments.

4.3 Proposed Identification Process for Hybrid Systems

To address these limitations, we propose a new learning framework that integrates flow function identification into the segmentation and grouping process. Rather than treating the components of the hybrid system as separate inference steps, our method, as shown in Figure 1, alternates between the identification of flow functions (A) and the detection of segments (B) using partial model information to guide the refinement of each component. This iterative approach directly targets the optimization problem in Equation (1) by prioritizing minimality: new modes are introduced only when the existing set of flow functions cannot explain the data with sufficient accuracy. This design aligns the learning process with the goal of minimality in model structure. Our approach does not guarantee finding an optimal solution for Equation (1). Instead, we optimize the length of individual segments under the constraint that a single flow function represents the complete segment. With this procedure, the number of segments needed to cover all traces is reduced. Finally, a smaller number of segments facilitates a smaller number of flow functions.

Currently, our framework focuses on the identification of flow functions and their associated modes. The inference of transition conditions is treated as a separate post-processing step, to be learned independently once the mode structure is established.

Our iterative learning approach is described as follows:

A Identification of Flow Function: Find a segment of maximal length starting from a first trace such that a flow function exists, that achieves a sufficient accuracy on this segment. Identify the flow function for the found segment.
B Detection of Accurate Segments: Assess the accuracy of the identified flow function on all observed traces.
If accuracy sufficient: **Stop**, and finalize the modes.
If accuracy insufficient:
C Refinement of Flow Function: Refine the flow function on segments with sufficient accuracy and keep this flow function for the mode.
Back to A Continue with the remaining subtraces, returning to Step A.

At each iteration on the set of traces, a single flow function is learned and then used to identify segments where the flow function is valid, i.e., achieves sufficient accuracy and subtraces where the accuracy of the flow function is insufficient. The data for all segments with sufficient accuracy is used to refine the flow function for this mode. The process repeats on the remaining data until all segments have been assigned to a mode.

In relation to Figure 1, we formulate one iteration of the approach. The identification of a flow function (A) is given as

$$\tilde{f} = \arg\min_{f \in \Phi} e(f, \mathcal{O}_j[0, l^*]), \tag{7}$$

where $e(f, \mathcal{O}_j[0, l^*])$ is the approximation error which is determined for the function $\tilde{f}$ on the segment $\mathcal{O}_j[0, l^*]$ of length l^* of an arbitrary trace $\mathcal{O}_j \in \mathcal{O}$. The length l^* is maximized in an iterative manner together with the identification of the flow function f:

$$l^{(k)} = \begin{cases} l_{\text{init}} & \text{if } k = 0, \\ \max\{n \leq k_j \mid e(f^{(k-1)}, \mathcal{O}_j[0, n]) \leq \epsilon\} & \text{if } k > 0, \end{cases} \tag{8}$$

$$f^{(k)} = \arg\min_{f \in \Phi} e(f, \mathcal{O}_j[0, l^{(k)}]), k \geq 0, \tag{9}$$

where given a flow function $f^{(k-1)}$ from the previous iteration, the length $l^{(k)}$ is determined as the maximum length of a segment where the flow function $f^{(k-1)}$ achieves sufficient accuracy. The flow function $f^{(k)}$ is then learned on this segment of length $l^{(k)}$. The parameter l_{init} is the initial length of the segment, and k_j is the length of the trace $\mathcal{O}_j$. The segment length l^* is the fixed point of Equation (8) and Equation (9), i.e., where $l^{(k)} = l^{(k+1)}$. This is the maximum length of a segment that achieves a sufficient accuracy on the trace $\mathcal{O}_j$.

The detection of accurate segments (B) finds $\mathcal{S}_A$ as a set of segments from the traces in $\mathcal{O}$ where the function $\tilde{f}$ achieves sufficient accuracy:

$$\mathcal{S}_A = \bigcup_{j \in m} \{\mathcal{O}_j[i] \mid e(\tilde{f}, \mathcal{O}_j[i]) < \epsilon \text{ with } i \in [1, ..., k_j]\}. \tag{10}$$

Finally, we use $\mathcal{S}_A$ to refine a final flow function f (C) which represents a new dynamic for the segments of $\mathcal{S}_A$:

$$f = \arg\min_{f \in \Phi} e(f, \mathcal{S}_A). \tag{11}$$

The next iteration of the algorithm starts on an updated set of observations

$$\mathcal{O} = \bigcup_{j \in m} \{\mathcal{O}_j[i] \mid e(f, \mathcal{O}_j[i]) \geq \epsilon \text{ with } i \in [1, ..., k_j]\}. \tag{12}$$

In this set, all subtraces of the traces in $\mathcal{O}$ where the flow function f does not achieve sufficient accuracy, form input traces for the next iteration. These are the subtraces that are not included in the set of segments $\mathcal{S}_A$.

The iterative process continues until all traces in $\mathcal{O}$ have been covered, i.e., $\mathcal{O} \equiv \emptyset$.

Compared to the traditional approach (Equation (2)–Equation (6)), this formulation eliminates the explicit representation of transition points and segment groups. Instead, transitions emerge naturally from the explanatory limits of each flow function on the data. Mode identification and segmentation are no longer treated as independent pre-processing steps but are tightly coupled through accuracy-driven inference. The condition for the detection of a transition point (formulated as φ in the traditional approach) is based directly on the accuracy of the identified flow function. Consequently, this procedure addresses the original problem in Equation (1) more directly than the traditional approach.

4.4 An Algorithmic Approach to the Proposed Method

We concretize the learning process in Algorithm 1, which describes the steps of the learning process.

The algorithm begins with an arbitrary trajectory from $\mathcal{O}$. For the sake of reproducibility, we choose the first trace $\mathcal{O}_0$ in Line 2. A belief flow function $\tilde{f}$ is learned over a growing time window starting at the beginning of $\mathcal{O}_0$ with an initial length of l_{init} in the loop from Line 4 to Line 8. The window is expanded in increments of size l_{step} until the approximation error exceeds the threshold ϵ. The resulting function $\tilde{f}$ is assumed to characterize a new dynamic. All trace segments where $\tilde{f}$ achieves an error below ϵ are considered additional instances of this dynamic and are used to find a final f for the found mode. This set $\mathcal{S}_A$ of segments is determined by the function *getAccurateSegments* in Line 9. The set $\mathcal{S}_A$ is then used to refine the flow function in Line 10, which identifies the new dynamic. Afterward, all identified segments are removed from the traces in Line 12 which reduces existing traces or splits traces into multiple new traces. Figure 4 visualizes the effect of the function *removeAccurateSegments*. Starting from the traces $\mathcal{O}_1$, $\mathcal{O}_2$, and $\mathcal{O}_3$, the segments where the

Algorithm 1 Identification of Hybrid Systems.

Data: $\mathcal{O}$
Result: $\mathcal{F}$
while $\mathcal{O} \neq \emptyset$ **do**
  $\mathcal{O}_0 \leftarrow \mathcal{O}[0]$;
  $i_{end} \leftarrow l_{init}$; $e \leftarrow 0.0$;
  while $i_{end} < |\mathcal{O}_0| \ \wedge \ e < \epsilon$ **do**
    window $\leftarrow \mathcal{O}_0[0, i_{end}]$;
    $\tilde{f}, e \leftarrow$ learnFlowFunction(window);
    $i_{end} \leftarrow i_{end} + l_{step}$;
  end
  $\mathcal{S}_A =$ getAccurateSegments$(\mathcal{O}, \tilde{f}, \epsilon)$;
  $f \leftarrow$ refineFlowFunction$(\mathcal{S}_A)$;
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$;
  $\mathcal{O} \leftarrow$ removeAccurateSegments$(\mathcal{O}, \mathcal{S}_A)$;
end

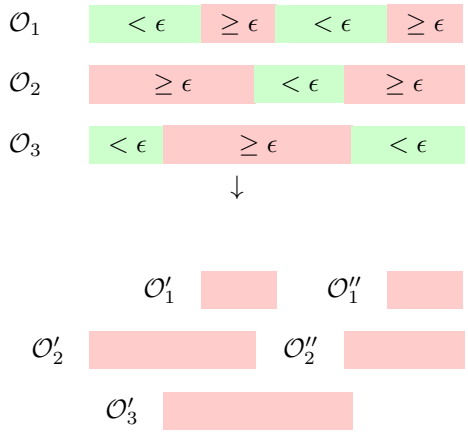


Figure 4 Segmentation using the Belief Flow Function.

error is smaller than ϵ are removed. These segments are covered by the flow function f. The new set of traces is $\mathcal{O} = \{\mathcal{O}'_1, \mathcal{O}''_1, \mathcal{O}'_2, \mathcal{O}''_2, \mathcal{O}'_3\}$. The process repeats until all observed data have been covered, i.e., $\mathcal{O} \equiv \emptyset$.

5 Empirical Analysis

This section presents an empirical evaluation of our learning approach as introduced in Algorithm 1. The approach is independent of the specific regression method used for *learnFlowFunction* and *refineFlowFunction*. To demonstrate this flexibility, we apply our learning approach with three distinct regression methods, each operating over different classes of flow functions. The results illustrate the versatility of the approach across diverse regression methods and hybrid system types.

The structure of this section is as follows: we first describe the learning methods employed, followed by a description of the evaluation examples and, finally, a presentation and discussion of the results.

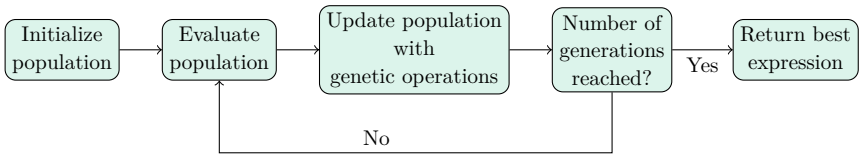


Figure 5 Genetic Programming for Symbolic Regression.

5.1 Learning Methods

The learning approach presented in this paper is modular and allows the integration of different regression methods for the identification of flow functions. We consider three methods during evaluation of our learning approach:

- Linear regression (LR) for learning simple, interpretable linear flow functions.
- Symbolic regression (SR) to identify more expressive symbolic representations.
- Linear matrix (LM) differential equations for systems with multidimensional state spaces.

For symbolic and linear regression, we restrict our evaluation to hybrid systems with a single output variable that corresponds directly to the state variable, i.e., $O = S$ and $|O| = 1$. The third regression method, i.e., linear matrix differential equations, applies to a multidimensional state.

Linear Regression

Linear regression learns flow functions in the form of affine linear mappings: $o = \mathbf{w}^T\mathbf{i} + b$, where $\mathbf{i}$ is the input vector, $\mathbf{w}$ is the learned weight vector, and b is the bias term. This method offers high computational efficiency and interpretability, but is limited in expressiveness due to the restriction to linear functions. We use the implementation provided by the `scikit-learn` library [12], which solves the regression problem using ordinary or non-negative least squares.

Symbolic Regression

Symbolic regression aims to learn expressions of the form $o = r(\mathbf{i})$, where r is a symbolic expression composed of a predefined set $\mathcal{B}$ of basic functions and operators [6]. Candidate expressions are evaluated based on a fitness metric that balances accuracy in the training data and expression complexity. A regularization term – weighted by a parsimony coefficient ρ – is typically included to prevent expression bloat [17].

In our experiments, we use the PySR framework [4], which implements symbolic regression via genetic programming. The algorithm maintains a population of size p of expressions, which evolves over multiple generations through selection, mutation, and crossover. Figure 5 illustrates this evolutionary process.

Linear Matrix Differential Equations

With a third regression method, we model time-continuous, multidimensional linear models, specifically linear matrix differential equations. These are common in system identification, particularly in ARX models [7], and are of the form:

$$\dot{\mathbf{y}}[t] = \mathbf{A}\mathbf{y}[t] + \mathbf{B}\mathbf{u}[t], \tag{13}$$

where $\mathbf{y}[t]$ and $\mathbf{u}[t]$ denote the state and input vectors, respectively, and $\mathbf{A}$ and $\mathbf{B}$ are the system and input matrices.

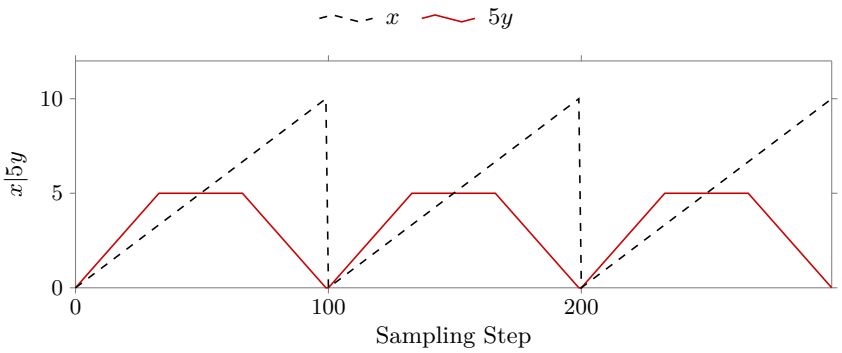


Figure 6 Piecewise Linear Function with Three Segments. The value of y is scaled for better visibility.

Given a set of observed traces $\mathcal{O}$, the goal is to identify the matrices $\mathbf{A}$ and $\mathbf{B}$ that best explain the observed dynamics. This is achieved by solving the following least-squares optimization problem:

$$\min \|\mathbf{Y}[t+1] - \mathcal{A}\mathbf{Y}[t]\|, \tag{14}$$

where $\mathbf{Y}[t]$ is constructed from all observed state and input vectors at time t:

$$\mathbf{Y}_{\mathcal{O}}[t] = \begin{bmatrix} \mathcal{Y}_1[t] & \cdots & \mathcal{Y}_m[t] \\ \mathcal{U}_1[t] & \cdots & \mathcal{U}_m[t] \end{bmatrix}. \tag{15}$$

The least squares solution is obtained using the linear regression implementation of `scikit-learn` [12].

Traditional Baseline Method

In addition to the analysis of the presented learning approach, we compare to the results of hybrid system identification with FaMoS [14]. FaMoS follows the traditional approach to hybrid system identification as given in Figure 2. The approach uses the Euclidean distance of windows of previous samples and upcomig samples of a trace for every sampling step for the segmentation of traces in Step (2). The segments are then grouped based on the similarity of the segments in Step (3) using dynamic time warping [10]. Flow functions are identified as linear matrix difference equations in Step (3).

5.2 Examples

We illustrate our learning approach with two representative examples: a piecewise linear function and a bouncing ball system. The first example serves as a simple baseline that allows a straightforward identification of the flow functions. The second example, the bouncing ball, is a classical benchmark in system identification. Although the bouncing ball exhibits discontinuities in velocity, it can be modeled using a single flow function. In addition, we demonstrate the applicability of different learning methods for both examples and compare identified flow functions with the original system dynamics.

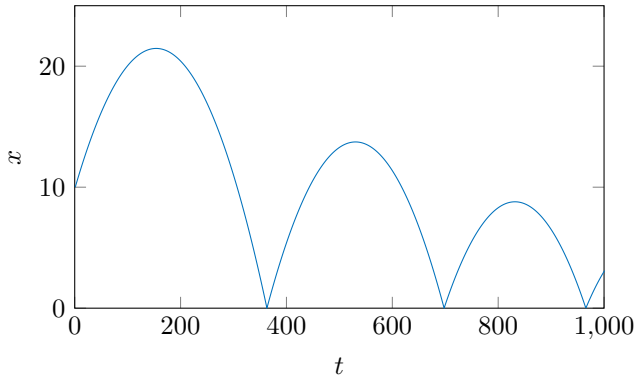


Figure 7 Bouncing Ball with Height over Time.

Piecewise Linear Function

As a simple baseline, we consider a piecewise linear function with three segments. The function is defined as follows:

$$f(x) = y = \begin{cases} 0.3 \cdot x & x < \frac{10}{3} \\ 1 & \frac{10}{3} \leq x < \frac{20}{3} \\ -0.3 \cdot x + 3 & \frac{20}{3} \leq x < 10. \end{cases} \tag{16}$$

The observation of the output variable $y = f(x)$ and the input variable x over time is shown in Figure 6. The system is excited with a repeating input signal x, which is increasing and resetting linearly.

Bouncing Ball

The bouncing ball is a well-known example for system identification. The system consists of a ball dropped from an initial height, bouncing on the ground. We use a simulation of the bouncing ball in Matlab [20] with a sampling time of $\Delta t = 0.01s$. The height x of the ball observed over time is shown in Figure 7.

The system dynamics are governed by the matrix differential equation:

$$\begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix} + \begin{pmatrix} 0 \\ -g \end{pmatrix}, \tag{17}$$

where g denotes the gravitational acceleration and v is the velocity of the ball.

We observe the system with a fixed sampling rate and, thus, learn a discrete difference equation as follows:

$$\begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x(t-1) \\ v(t-1) \end{pmatrix} + \begin{pmatrix} 0 \\ -g \end{pmatrix} \cdot \Delta t. \tag{18}$$

For model learning with symbolic regression and linear regression, one dimensional flow functions are learned. Thus, the height of the ball over time is approximated by the height of the ball at the previous sampling step and the height at the second-last sampling step. We provide a time history and approximate the height of the ball as follows:

$$x(t) = 2 \cdot x(t-1) - x(t-2) - \frac{g}{2} \cdot \Delta t. \tag{19}$$

Table 1 Learning Parameters for Hybrid System Identification (Algorithm 1) and Symbolic Regression.

Example	Algorithm 1			Symbolic Regression		
	ϵ	l_{init}	l_{step}	ρ	n	p
Piecewise Linear	0.01	5	1	0.9	10	31
Bouncing Ball	0.5	10	10	0.9	10	31

Table 2 Results of Model Learning.

Example	Method	Flow Functions	MSE	Learning Time in s
Piecewise Linear	LR	$\begin{cases} f_1(x) &= 0.3 \cdot x - 2.2 \cdot 10^{-16} \\ f_2(x) &= 1.0 \\ f_3(x) &= -0.3 \cdot x + 3.0 \end{cases}$	$2.198 \cdot 10^{-32}$	0.159
	SR	$\begin{cases} f_1(x) = & 0.3 \cdot x \\ f_2(x) = & 0.5 + 0.5 \\ f_3(x) = & 3.066 - (-1.264) \cdot \\ & (-0.237) \cdot (x + 0.220) \end{cases}$	$1.192 \cdot 10^{-15}$	568.016
Bouncing Ball	LR	$\begin{cases} f_1(x_1, x_2) = & 1.989 \cdot x_1 - 0.989 \cdot x_2 \\ & +0.003 \end{cases}$	$1.99 \cdot 10^{-4}$	0.268
	SR	$\{ f_1(x_1, x_2) = x_1 + x_1 - x_2$	$2.025 \cdot 10^{-4}$	570.75
	LM	$\begin{cases} f_1(x, v) = & \begin{pmatrix} 1 & 0.001 \\ -0.023 & 0.984 \end{pmatrix} \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} \\ & + \begin{pmatrix} 0.001 \\ 0.225 \end{pmatrix} = \begin{pmatrix} x(t+1) \\ v(t+1) \end{pmatrix} \end{cases}$	$\begin{pmatrix} 8.749 \cdot 10^{-3} \\ 4.573 \cdot 10^{-2} \end{pmatrix}$	0.038

Setup & Parameters

Table 1 lists the parameters used in the experiments. In our learning approach from Algorithm 1, ϵ denotes the error threshold, l_{init} the initial segment length, and l_{step} the increment of the segment length. For symbolic regression, ρ is the parsimony coefficient, n the number of generations, and p the population size.

5.3 Identification with Linear Regression

We apply our learning approach with linear regression to both the piecewise-linear function and the bouncing ball system. As shown in Table 2, three distinct flow functions are identified for the piecewise linear function, each corresponding to one mode. The learned functions match the true system except for a minor difference in the first flow function. The maximum length of the found segments is $l^* = 33$, which is equivalent to the sampling step of the ground truth transition point.

Similar results are obtained for the bouncing ball. Here, the error is slightly larger, but also a larger error threshold ϵ is used. The learned flow function differs in the slope and offset from the original function. The linear function based on the heights x_1 and x_2 of the last and second-last sampling step is learned, respectively. This is an approximation of

the actual differential equation of the bouncing ball. Due to the high sampling rate, the approximation is very close to the actual differential equation. This is also shown in Figure 8. Figure 8a shows the learned flow function compared to the ground truth. The learned flow function is close to the ground truth. Figure 8b shows the absolute error of the learned flow function compared to the ground truth. The error is small, but varies over time due to the mismatch of the slope and offset of the learned flow function compared to the ground truth. Larger errors occur at the transition points, where the ball bounces on the ground and the simulation resolution is not sufficient to capture the dynamics of the bouncing ball.

Further, the variant of our approach with linear regression has a very low learning time.

5.4 Identification with Symbolic Regression

Symbolic regression offers greater flexibility compared to linear regression, enabling the identification of a wider class of functions depending on the available operators and basis functions. For both the piecewise-linear function and the bouncing ball, we use addition, subtraction, and multiplication as primitives. In addition, symbolic regression identifies constants for the learned expressions.

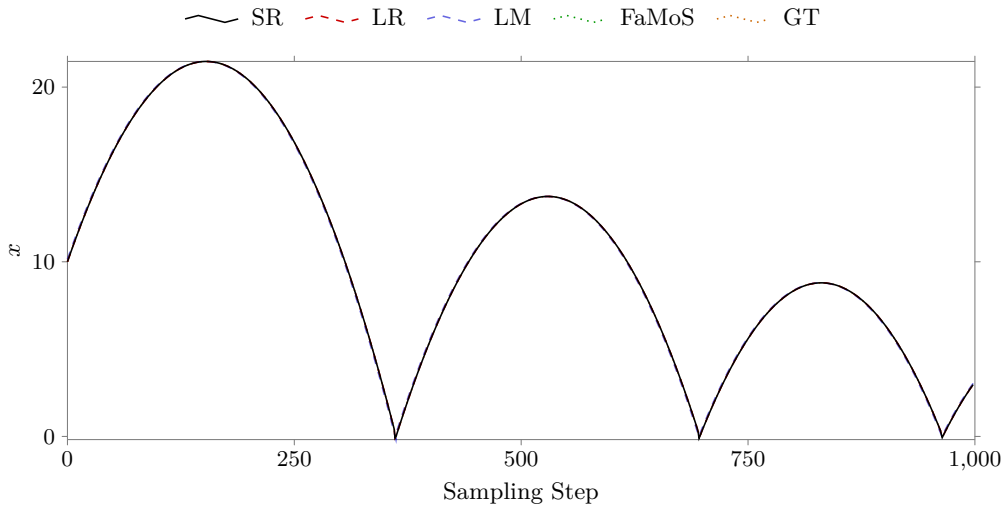
The results of the model learning are shown in Table 2. For the piecewise-linear function, we learn three flow functions, which are the three modes of the piecewise-linear function. The first two flow functions are equivalent to the original function. However, instead of 1 for the second flow function, $0.5 + 0.5$ is learned. Symbolic regression does not simplify expressions internally, as this could hinder exploration of the search space in the genetic algorithm. We simplify the expression after learning, which results in the following set of flow functions:

$$\begin{cases} f_1(x) & = 0.3 \cdot x \\ f_2(x) & = 1.0 \\ f_3(x) & \approx 3.0 - 0.3 \cdot x \end{cases} . \tag{20}$$

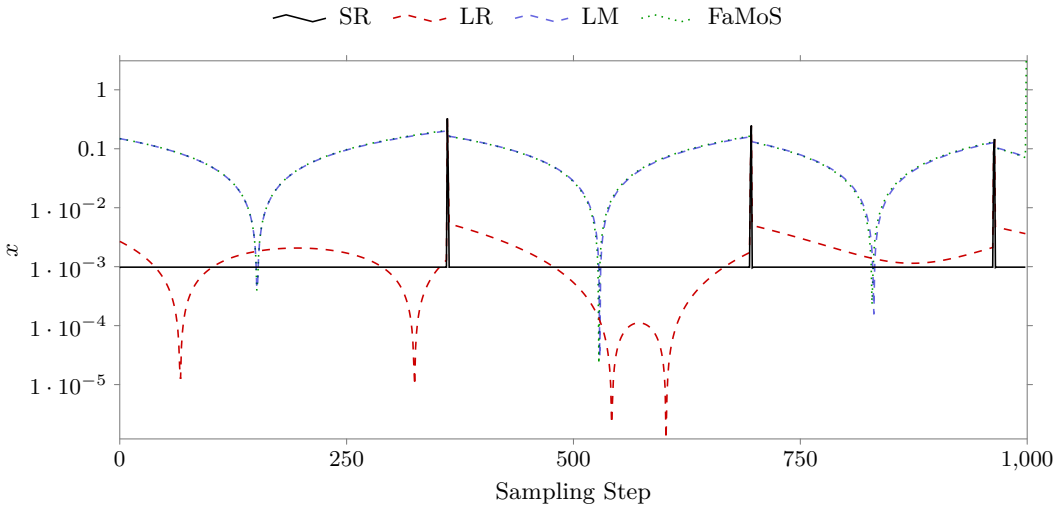
The slope and offset of the third flow function deviate only in the range 10^{-8} from the original function. Again, the maximum length of the found segments is $l^* = 33$, which is equivalent to the ground truth transition point.

Similar results are obtained for the bouncing ball. The coefficients for the features x_1 and x_2 are learned correctly, although $x_1 + x_1$ is learned instead of $2 \cdot x_1$. Compared to the original function, the constant offset is not learned. However, the mean squared error is close to the error of the linear regression. Figure 8a shows the learned flow function compared to the ground truth. The learned flow function is close to the ground truth. Figure 8b shows the absolute error of the learned flow function compared to the ground truth. The error is small for both linear regression and symbolic regression, and the error of linear regression is on average smaller than the error of symbolic regression. The error with symbolic regression is constant because only the constant offset is not learned correctly. Larger errors occur at the transition points, where the ball bounces on the ground and the simulation resolution is not sufficient to capture the dynamics of the bouncing ball.

Overall, linear regression achieves lower errors and significantly faster learning time than symbolic regression. This is due to the larger search space Φ of symbolic regression and its dependence on heuristic optimization (here, genetic programming), in contrast to the closed-form solution used in linear regression.



(a) Bouncing Ball Height x for our Learning Approach with Linear Regression (LR), Symbolic Regression (SR), and Linear Matrix Regression (LM) as well as FaMoS [14]. Additionally, the ground truth (GT) is shown.



(b) Bouncing Ball Absolute Error in Height x for our Learning Approach with Linear Regression (LR) and Symbolic Regression (SR), and Linear Matrix Regression (LM) as well as FaMoS [14] (Note that the FaMos curve coincides with that of LM).

Figure 8 Bouncing Ball Height x and Error for our Learning Approach with Linear Regression (LR), Symbolic Regression (SR), and Linear Matrix Regression (LM) as well as FaMoS [14].

Table 3 Learning Results for the Bouncing Ball with FaMoS [14].

Flow Function	MSE	Learning Time in s
$\begin{cases} f_1(x,v) = \begin{pmatrix} 1.0 & 0.008 \\ 0.0 & 1.0 \end{pmatrix} \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} + \begin{pmatrix} 0.0 \\ -0.006 \end{pmatrix} = \begin{pmatrix} x(t+1) \\ v(t+1) \end{pmatrix} \\ f_2(x,v) = \begin{pmatrix} 0.999 & 0.007 \\ -0.082 & 0.975 \end{pmatrix} \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.036 \end{pmatrix} = \begin{pmatrix} x(t+1) \\ v(t+1) \end{pmatrix} \\ f_3(x,v) = \begin{pmatrix} 1.0 & 0.0 \\ -0.177 & 0.947 \end{pmatrix} \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.052 \end{pmatrix} = \begin{pmatrix} x(t+1) \\ v(t+1) \end{pmatrix} \end{cases}$	$\begin{pmatrix} 4.021 \cdot 10^{-5} \\ 2.217 \cdot 10^{-4} \end{pmatrix}$	0.012

5.5 Identification of Linear Matrix Differential Equations

We perform our learning approach with linear matrix regression for the bouncing ball. The goal is to learn the two-dimensional state space of the ball as given in Equation (18). The results are shown in Table 2 and visualized in Figure 8a and Figure 8b.

The first line of the matrix differential equation is learned correctly except for the constant offset. The second line of the matrix differential equation differs from the original function. Instead of a constant reduction of the velocity, a factor smaller than one is learned for the previous velocity.

The MSE of the learned matrix differential equation for the height of the ball is higher compared to the results of one-dimensional linear regression and symbolic regression. The learning problem here is more complex as a two-dimensional state is learned. This regression method enables hybrid system identification for a higher-dimensional state space. The learning time for linear matrix regression is small.

5.6 Comparison to Traditional Learning of Hybrid Automata

Finally, we compare our learning approach with traditional learning of hybrid automata. We use the bouncing ball example and learn a hybrid automaton with the FaMoS tool [14]. The results are shown in Table 3.

The learning time of FaMoS is low and the MSE is small. FaMoS achieves a slightly lower MSE than all variants of our approach, which is due to the fact that FaMoS normalizes the data before learning and, thus, the MSE is evaluated on normalized data. The prediction of the state space with the learned flow function and the ground truth is shown in Figure 8a as a dotted green line and is very close to the ground truth. Figure 8b shows the denormalized absolute error of the learned flow function compared to the ground truth. The error is very similar to the error of our learning approach with linear matrix regression. However, instead of a single flow function, three flow functions are learned with FaMoS. All three flow functions are similar and are close to the ground truth, but not exactly equivalent to the original functions. This example shows that our approach can achieve a smaller number of flow functions and, thus, a smaller model. The traditional approach executed by FaMoS

detects transition points at the bouncing points of the ball, i.e., whenever $x \equiv 0$. For the given learning data, transition points are detected at the sampling steps 361 and 695. The flow functions are learned on the segments individually. The grouping step in FaMoS is not able to group the segments correctly, as they are not similar when comparing the curves of the segments. Thus, FaMoS fails to recognize identical dynamics in the segments. Our learning approach instead groups segments based on the flow functions and, thus, needs only a single flow function to represent the dynamics of the bouncing ball.

6 Discussion & Conclusion

In this work, we present a method to learn the flow functions of hybrid systems from observations integrating where the individual flow functions are valid. Revisiting the traditional sequential learning approach for hybrid systems, we propose a new iterative learning paradigm. This iterative process allows us to use the learned flow functions as intermediate results within the learning process. This ensures that transition points between dynamic modes are detected only if the current flow function is not able to explain the observations. Our learning approach, thus, is closer to the actual optimization problem of hybrid system identification and specifically focuses on the identification of hybrid systems with a small number of modes. We propose a template algorithm which is agnostic to the actual set of flow functions and learning methods used. Our empirical evaluation instantiates the template algorithm with symbolic regression, linear regression, and linear matrix identification. The results show that the approach identifies simple flow functions with high accuracy. Further, we demonstrate that the approach is able to identify hybrid systems with fewer modes compared to a traditional sequential learning approach on a bouncing ball benchmark.

Future work extends the current approach to involve the identification of events and transition conditions as well as reset relations, which enable the reset of system variables at the transition between modes.

References

1 Rajeev Alur. Principles of Cyber-Physical Systems. Technical report, MIT Press, 2015.

2 Nathalie Barbosa Roa, Louise Travé-Massuyès, and Victor H. Grisales-Palacio. Dyclee: Dynamic clustering for tracking evolving environments. *Pattern Recognition*, 94:162–186, 2019. doi:10.1016/j.patcog.2019.05.024.

3 Michael S. Branicky. *Introduction to Hybrid Systems*, pages 91–116. Birkhäuser, Boston, 2005.

4 Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression.jl, 2023. doi:10.48550/arXiv.2305.01582.

5 Nemanja Hranisavljevic, Alexander Maier, and Oliver Niggemann. Discretization of hybrid CPPS data into timed automaton using restricted Boltzmann machines. *Engineering Applications of Artificial Intelligence*, 95:103826, 2020. doi:10.1016/j.engappai.2020.103826.

6 John R. Koza. *On the programming of computers by means of natural selection*. Koza, John R. Genetic programming. MIT Press, Cambridge, Mass. u.a., 1992.

7 Lennart Ljung. *System Identification: Theory for the User*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

8 J. Lunze and F. Lamnabhi-Lagarrigue. *Handbook of Hybrid Systems Control: Theory, Tools, Applications*. Cambridge University Press, Cambridge, 2009.

9 Daniel L. Ly and Hod Lipson. Learning Symbolic Representations of Hybrid Dynamical Systems. *Journal of Machine Learning Research*, 13(115):3585–3618, 2012. doi:10.5555/2503308.2503356.

10 Meinard Müller. *Dynamic Time Warping*, chapter 4, pages 69–84. Springer-Verlag, Heidelberg, 2007.

11 Oliver Niggemann, Benno Stein, Asmir Vodencarevic, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. *AAAI Conference on Artificial Intelligence*, 26(1):1083–1090, 2012. doi:10.1609/aaai.v26i1.8296.

12 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi:10.5555/1953048.2078195.

13 Swantje Plambeck. SymbolicRegression4HA. Software, BMBF project AGenC no. 16IS22047A, swhId: swh:1:dir:00ff8c08db289ab78af743dd7ce6b71dceccb37c (visited on 2025-10-21). URL: https://github.com/TUHH-IES/SymbolicRegression4HA, doi:10.4230/artifacts.24971.

14 Swantje Plambeck, Aaron Bracht, Nemanja Hranisavljevic, and Goerschwin Fey. Famos — fast model learning for hybrid cyber-physical systems using decision trees. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2024. doi:10.1145/3641513.3650131.

15 Swantje Plambeck, Maximilian Schmidt, Goerschwin Fey, Audine Subias, and Louise Travé-Massuyès. Dynamics-based identification of hybrid systems using symbolic regression. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2024. doi:10.1109/SEAA64295.2024.00019.

16 Swantje Plambeck, Maximilian Schmidt, Audine Subias, Louise Travé-Massuyès, and Goerschwin Fey. Usability of Symbolic Regression for Hybrid System Identification - System Classes and Parameters. In *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*, 2024. doi:10.4230/OASIcs.DX.2024.30.

17 Riccardo Poli, William B Langdon, and Nicholas F McPhee. *A Field Guide to Genetic Programming*, volume 10. Springer, 2008. URL: http://www.gp-field-guide.org.uk/.

18 Iman Saberi, Fathiyeh Faghih, and Farzad Sobhi Bavil. A passive online technique for learning hybrid automata from input/output traces. *ACM Transactions on Embedded Computing Systems*, 22(1):1–24, 2021. doi:10.1145/3556543.

19 Martin Tappler, Edi Muškardin, Bernhard K. Aichernig, and Bettina Könighofer. Learning environment models with continuous stochastic dynamics, 2023. doi:10.48550/arXiv.2306.17204.

20 Inc. The MathWorks. Model a bouncing ball in continuous time, 2025. URL: https://de.mathworks.com/help/stateflow/ug/modeling-a-bouncing-ball-in-continuous-time.html.

21 Xiaodong Yang, Omar Ali Beg, Matthew Kenigsberg, and Taylor T. Johnson. A framework for identification and validation of affine hybrid automata from input-output traces. *ACM Transactions on Cyber-Physical Systems*, 6(2):1–24, 2022. doi:10.1145/3470455.

Safe to Fly? Real-Time Flight Mission Feasibility Assessment for Drone Package Delivery Operations

Abenezer Taye
Mechanical and Aerospace Engineering Department, School of Engineering and Applied Science, George Washington University, Washington DC, USA

Austin Coursey
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Marcos Quinones-Grueiro
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Chao Hu
School of Mechanical, Aerospace, and Manufacturing Engineering, University of Connecticut, Storrs, CT, USA

Gautam Biswas
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Peng Wei
Mechanical and Aerospace Engineering Department, School of Engineering and Applied Science, George Washington University, Washington DC, USA

Abstract

Ensuring flight safety for small unmanned aerial systems (sUAS) requires continuous in-flight monitoring and decision-making, as unexpected events can alter power consumption and deplete battery energy faster than anticipated. Such events may result in insufficient battery capacity to complete a mission, thereby compromising flight safety. In this paper, we present an online feasibility assessment and contingency management framework that continuously monitors the aircraft's battery state and the energy required to complete the flight in real-time, which enables informed decision-making to enhance flight safety. The framework consists of two main components: power consumption prediction and battery voltage trajectory prediction. The power consumption prediction is conducted using a model that is based on momentum theory, while the voltage trajectory prediction is performed using a Neural Ordinary Differential Equation (Neural ODE)-based data-driven model. By integrating these two components, the framework evaluates the feasibility of a flight mission in real time and determines whether to proceed with the mission or initiate rerouting. We evaluate the framework's performance in a drone delivery scenario in the Dallas–Fort Worth (DFW) area, where the aircraft encounters an unexpected energy depletion event mid-flight. The proposed framework is tasked with assessing the feasibility of completing the mission and, if necessary, rerouting the aircraft for an emergency landing. The results demonstrate that the framework accurately and efficiently detects energy insufficiencies in real-time and re-routes the aircraft to a predefined emergency landing site.

2012 ACM Subject Classification Computing methodologies → Control methods; Computing methodologies → Model development and analysis

Keywords and phrases Battery Modeling, Neural ODE, Unmanned Aerial Vehicles

Digital Object Identifier 10.4230/OASIcs.DX.2025.8

Funding This work was supported by NASA award #80NSSC21M0087-21-S06.

1 Introduction

1.1 Motivation

Drone package delivery using small unmanned aerial systems (sUAS) is rapidly advancing and nearing widespread implementation. The FAA recently granted approval for companies such as Zipline and Wing Aviation to operate commercial drones in the Dallas-Fort Worth (DFW) area without requiring visual observers, enabling beyond-visual-line-of-sight (BVLOS) operations [11, 12, 16]. This historic authorization represents a significant shift in the regulatory landscape, paving the way for the safe and routine integration of drone deliveries into the national airspace.

Despite these advancements, ensuring the safety of drone delivery operations remains a critical challenge, particularly due to various operational hazards [24]. Among these, hazards related to aircraft components pose a significant risk to the reliability of advanced air mobility (AAM) operations such as drone package delivery. In this work, we specifically focus on one such hazard, which is the risk of insufficient battery capacity to complete a flight mission.

Due to the dynamic and uncertain nature of flight operations, even if pre-departure feasibility assessments account for all known factors, unexpected events during flight can still lead to insufficient battery energy. Various operational factors that happen during flight can influence the power consumption of the aircraft and deplete the battery energy faster than anticipated. For example, a mid-flight incident such as a bird strike may damage a propeller, reducing thrust efficiency and increasing power demand [9]. Similarly, if the electronic speed controller (ESC) or battery overheats beyond a safe threshold, the system may impose a speed restriction [13], limiting the aircraft's operational envelope and requiring rerouting to the nearest landing site.

To mitigate these risks and enhance flight safety, we propose a real-time flight mission monitoring scheme that continuously evaluates the feasibility of a package delivery mission based on the available battery energy. The intended operation of this framework in a real-world drone package delivery scenario is illustrated in Figure 1. The monitoring system is activated immediately after takeoff and periodically assesses whether the mission remains feasible. This enables timely adjustments, such as rerouting to a nearby warehouse or an emergency landing site, to ensure safe operation throughout the flight until the aircraft reaches its destination.

1.2 Related Work

The two areas related to our overall problem are battery state prediction and battery feasibility-based flight planning. Here we summarize the previous works related to these two areas.

1.2.1 Battery State Prediction

Existing battery state prediction methods can be broadly categorized into two approaches: model-based and data-driven methods. Model-based approaches rely on physical models of the battery to predict key states, primarily future trajectories of terminal voltage and state of charge (SoC). These methods typically use either equivalent circuit models or electrochemical-based models, combined with estimators, to forecast battery states over time.

Equivalent circuit models represent the battery's internal dynamics using electrical components such as resistors and capacitors. Common models include the Rint model, the RC model, and the Thévenin model [14]. These models simplify the battery's behavior but

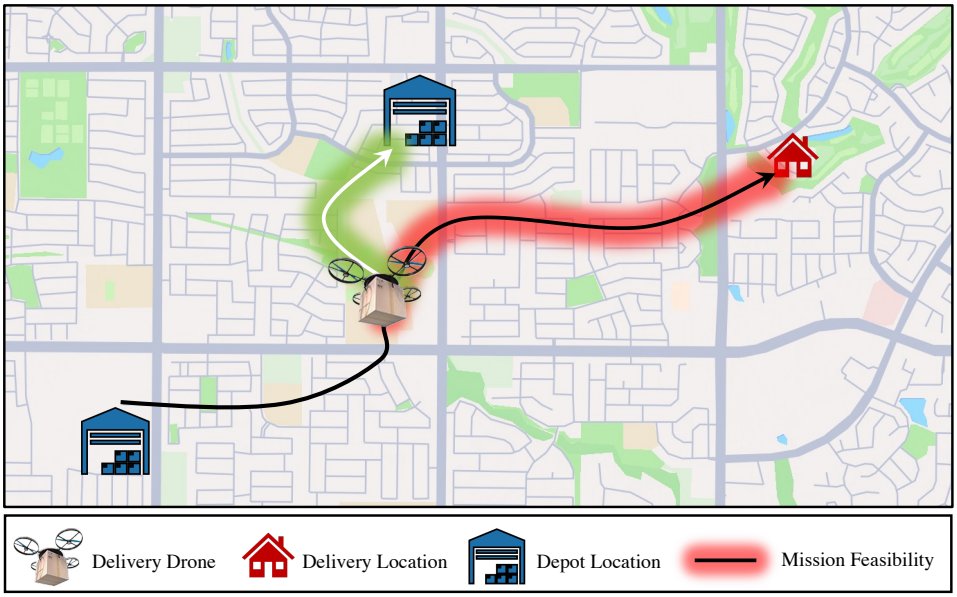


Figure 1 Schematic diagram representation of the feasibility assessment procedure.

still require parameter estimation and state estimation techniques for accurate predictions. In contrast, electrochemical-based models simulate the battery's internal chemistry using porous electrode theory [15] and describe its dynamics through partial differential equations (PDEs). There are multiple versions of these models, varying in complexity depending on the target application, with the most widely used being [8] and [10]. While model-based methods offer high accuracy and interpretability, they require simulating complex, nonlinear battery models, making them computationally expensive and impractical for real-time in-flight applications.

To address this limitation, data-driven approaches have been developed. These methods use machine learning techniques to predict battery states with reduced computational cost. Among the most common are long short-term memory (LSTM) networks [5], support vector machines (SVM) [18], and fuzzy inference systems (FIS) [17]. Data-driven approaches are more computationally efficient [3], making them suitable for real-time applications such as in-flight battery state prediction. However, their accuracy tends to degrade as the prediction horizon increases, limiting their reliability for long-term planning applications.

1.2.2 Battery Feasibility Based Flight Planning

The proposed framework aims to perform online battery state prediction for sUAS operations and assess mission feasibility. Several studies have explored similar goals. For instance, Shibl et al. [22] developed a battery management system for sUAS that employs deep neural networks (DNN) and LSTM networks to predict the SoC and the state of health (SoH). This system enhances battery monitoring and aids in mission planning based on the current battery state. Similarly, [4] proposed a method for assessing mission feasibility by considering battery performance and planning optimal routes to ensure successful mission completion. In another study, Shi et al. [20] introduced a cloud-based framework for the co-estimation of SoC and SoH, leveraging transformer-based deep learning techniques to provide accurate and real-time battery state predictions. Additionally, [21] explored a risk-aware approach for unmanned aerial vehicle and unmanned ground vehicle rendezvous planning using a chance-

constrained Markov Decision Process. Their method accounts for the stochastic nature of energy consumption and optimizes rendezvous points to enhance mission feasibility and safety. Furthermore, Choudhry et al. [7] developed a deep energy model utilizing Temporal Convolutional Networks to predict energy consumption. They introduced a Conditional Value-at-Risk (CVaR) metric to assess the risk of battery depletion during flights, providing a framework for risk-aware mission planning and feasibility assessment.

However, none of these studies approach battery feasibility-based flight planning from the perspective of forecasting the future voltage trajectory. In contrast, our framework introduces a two-stage pipeline that first predicts the power consumption profile of the aircraft for the planned mission and then leverages a data-driven battery model to forecast the corresponding voltage trajectory. This approach enables a more realistic and forward-looking assessment of mission feasibility, unlike prior methods that rely solely on the current battery state or coarse approximations of future energy demands.

2 Problem Formulation

Ensuring the real-time feasibility of flight missions is critical for safe and reliable operations, particularly in applications such as package delivery. The framework proposed in this paper periodically assesses battery feasibility during flight to determine whether the aircraft can successfully reach its destination or if it needs to reroute to an alternate landing site. This section outlines the formulations of the two key components of the feasibility assessment framework – the aircraft model and the battery model – and provides a formal description of the problem addressed in this work.

2.1 Aircraft Model

The aircraft operates in a three-dimensional environment with latitude, longitude, and altitude. The specific aircraft model considered in this paper is an octo-rotor, whose detailed dynamics is provided in [1]. For brevity, we summarize the aircraft dynamics here. At any time t, the aircraft's state in inertial space is represented as $\mathbf{x}_t \in \mathbb{R}^3$, and its evolution follows the system dynamics:

$$\dot{\zeta}_t = f(\zeta_t, u_t), \tag{1}$$

where $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is a continuous function. The vector ζ represents the aircraft's states, including its position (x, y, z), velocities $(\dot{x}, \dot{y}, \dot{z})$, angular positions $[\phi, \theta, \psi]$, and angular velocities $[p, q, r]$. The control input u_t ensures that the aircraft follows a predefined sequence of waypoints from the initial to the final destination.

2.2 Battery Model

The battery model utilized in this study is an electrochemical model of lithium-ion batteries, as described in [8], which are a popular choice for powering unmanned aerial vehicles. In this model, the battery's current draw, denoted by I_b, serves as the input, while the battery voltage V_b, temperature T_b, and state of charge (SoC) represent the system states. The battery dynamics are governed by the following system equation:

$$\dot{\boldsymbol{\xi}}_t = g(\boldsymbol{\xi}_t, I_b), \tag{2}$$

where $g : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$ is a continuous function. The state vector $\boldsymbol{\xi}$ represents the battery's internal states.

2.3 Problem Description

Consider an aircraft whose airframe and battery dynamics are given by Equation 1 and 2, respectively. The aircraft is assigned to deliver a package from an initial warehouse location $\mathbf{x}_w = (x_w, y_w, z_w)$ to a designated delivery site $\mathbf{x}_d = (x_d, y_d, z_d)$. Once the aircraft completes takeoff, the framework presented in this paper needs to perform a periodic feasibility check every τ seconds to ensure that the mission remains viable under real-time aircraft and environmental conditions. At each reassessment step, where $\mathbf{x}_t$ represents the position of the aircraft at the time of feasibility assessment, the condition for mission success is evaluated using:

$$V_{\text{success}}(t \mid \mathbf{x}_t) \geq V_{\text{thresh}}, \quad \forall t \in [t, t+T]. \tag{3}$$

If the battery voltage trajectory is greater than or equal to a predefined feasibility voltage threshold (V_{thresh}) value, the aircraft continues its flight. However, if at any point in time, the battery voltage drops below the threshold, the aircraft must reroute to the nearest warehouse or designated emergency landing site.

3 Method

To address the problem described in the previous section, we propose the framework shown in Figure 2, where, at each decision-making step, three major tasks are performed: 1) power consumption prediction, 2) battery voltage and SoC prediction, and 3) feasibility assessment and decision making. The remainder of this section discusses these three major tasks of the framework in detail.

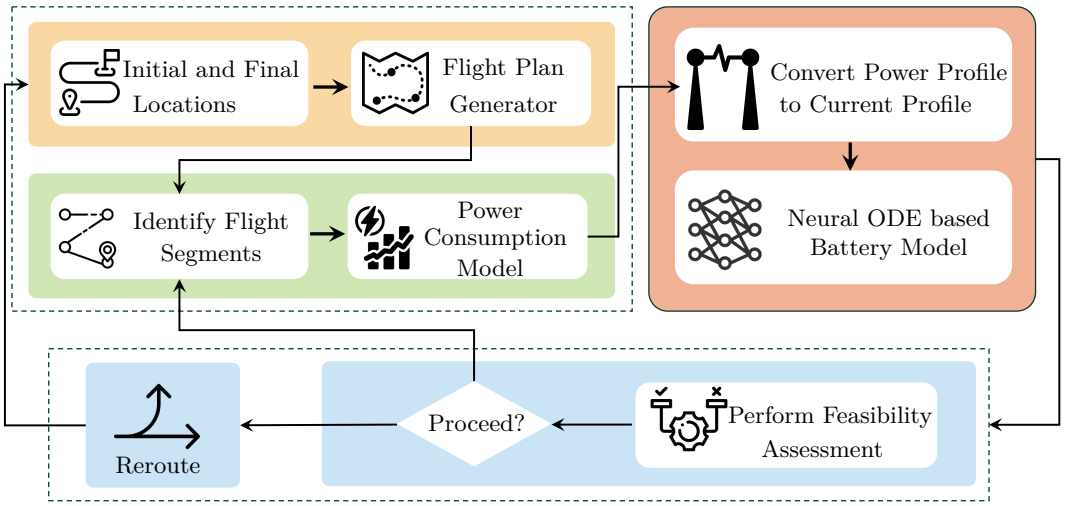


Figure 2 Schematic diagram of the feasibility assessment procedure.

3.1 Power Profile Prediction

Accurate prediction of power consumption for a future flight trajectory is critical for assessing the feasibility of a mission. The proposed framework predicts the power consumption of a future flight trajectory using the following procedure. First, a flight plan is generated for the remaining flight using the initial and final locations. This flight plan comprises waypoints

between these initial and final locations along with the velocity profile of the aircraft. Next, the framework identifies distinct flight segments – such as takeoff, cruise, and landing – and computes the power consumption and flight duration for each flight segment using the power consumption model which is discussed in the following subsection.

3.1.1 Aircraft Power Consumption Model

To determine the power required for future flight operations, the ideal approach would be to simulate the detailed aircraft dynamic model and collect the power required for future flight duration. However, since the detailed aircraft dynamic model is complex and highly nonlinear, doing so is computationally expensive. To address this challenge, we adopt a power consumption model for rotary-wing aircraft from [23]. This model is based on momentum theory and incorporates aerodynamic equations for each flight maneuver including, climb, hover, horizontal flight, and descent.

$$P_{\text{hover}} = \frac{W^{\frac{3}{2}}}{\eta_h \cdot \sqrt{2\rho A_t}}, \tag{4}$$

$$P_{\text{climb}} = \frac{W}{\eta_c}\left(\frac{V_c}{2} + \sqrt{\frac{V_c^2}{4} + \frac{W}{2\rho A_t}}\right), \tag{5}$$

$$P_{\text{descent}} = \frac{W}{\eta_c}\left(\frac{-V_d}{2} + \sqrt{\frac{V_d^2}{4} + \frac{W}{2\rho A_t}}\right), \tag{6}$$

where V_c and V_d are vertical climb speed and vertical descent speed values, respectively. In addition, η is the efficiency factor of the propulsion system, ρ is the air density, W is the total weight of the aircraft, and A_t is the sum of the n-disc actuator areas. In addition, the instantaneous power for horizontal flight is given as:

$$P_{\text{horizontal}} = \frac{W}{\eta_{\text{hor}}}\left(V_{\text{hor}}\sin(\alpha_v) + v_{\text{hor}}\right), \tag{7}$$

where α_v is the angle of attack and η_{hor} and the horizontal efficiency. In addition, the induced velocity in horizontal flight v_{hor}, is given by:

$$v_{\text{hor}} = \sqrt{-\frac{V_{\text{hor}}^2}{2} + \sqrt{\frac{V_{\text{hor}}^4}{4} + (\frac{W}{2\rho A_t})^2}}. \tag{8}$$

All the aircraft-related parameters mentioned in the equations above are given in Table 1.

3.1.2 Flight Duration Estimation

The aircraft power consumption model outlined above provides the instantaneous power required by the aircraft during a given timestamp in a specific flight phase. However, to predict the power profile for each flight segment, we must also determine the duration of each segment. Here, we discuss the approach used to estimate the flight duration of each flight segment. During a climb or descent, the aircraft changes altitude at a constant speed, and the total time required to reach the desired attitude is given by:

$$T_{\text{climb/descend},i} = \frac{h_{i+1} - h_i}{V_{\text{climb/descend}}}, \tag{9}$$

Table 1 Aircraft- and battery-related parameters.

Aircraft Parameters		Battery Parameters	
ρ	Air density (1.225 kg/m^3)	R_{int}	Internal resistance (0.05 Ω)
A_t	Rotor disk area (1.31 m^2)	η	Battery efficiency (0.95)
W	Aircraft weight (10 kg)	C_n	Nominal capacity (22000 mAh)
η_c	Climb efficiency (0.85)	k_0	Open-circuit voltage constant (22.83)
α_v	Horizontal drag coefficient (0.25)	k_1	Open-circuit voltage constant (0.39)
η_d	Descent efficiency (0.75)	k_2	Open-circuit voltage constant (−0.78)
η_{hor}	Horizontal efficiency (0.88)	V_{thresh}	Battery voltage threshold (18 V)

where $h_{i+1} - h_i$ represents the altitude change and V_{climb} or (V_{descend}) is the predefined climb (or descent) speed. Similarly, once the aircraft reaches cruising altitude, the aircraft moves with a constant horizontal velocity. The cruise time for a given segment is computed as:

$$T_{\text{cruise},i} = \frac{d_i}{V_{\text{cruise}}}, \tag{10}$$

where d_i is the horizontal distance of the segment and V_{cruise} is the predefined cruise speed. Finally, the overall flight power profile is obtained by concatenating the power profiles for the climb, cruise, and descent segments, each computed over its respective flight duration:

$$P_{\text{flight}} = [P_{\text{climb}}, P_{\text{cruise}}, P_{\text{descend}}]. \tag{11}$$

3.2 Power to Current Conversion

After predicting the power profile for the entire flight, we need to convert it into a current profile to use as input for our Neural ODE-based battery model. However, this conversion is not straightforward, as both voltage and current are unknown in our problem setting. To address this, we adopt the Rint-based equivalent circuit model [14]. This model represents the battery as an ideal voltage source in series with a single resistor and is described by the following equation:

$$V(t) = V_{\text{oc}}(t) - I(t)R_{\text{int}}, \tag{12}$$

where $V(t)$ is the battery voltage, $V_{\text{oc}}(t)$ is the open circuit voltage, and R_{int} represents the internal resistance of the battery. By rewriting this equation using the relationship $P(t) = V(t) \cdot I(t)$, we obtain the following quadratic equation:

$$I(t)^2 R_{\text{int}} - V_{\text{oc}}(t)I(t) + P(t) = 0. \tag{13}$$

In the above equation, $V_{\text{oc}}(t)$ is determined from the OCV-SoC curve of the battery, assuming the SoC of the battery at a given time is known. In this study, we use the Nernst model to represent the OCV-SoC relationship, which is commonly applied to Li-ion and Li-Po batteries:

$$V_{\text{oc}}(\text{SoC}) = k_0 + k_1 \ln(\text{SoC}) + k_2 \ln(1 - \text{SoC}), \tag{14}$$

where the parameters $k_0 = 22.83$, $k_1 = 0.39$, and $k_2 = -0.78$ are obtained by fitting battery data [2]. Figure 3 shows the fitted OCV-SoC curve. Once the SoC is determined, this curve is used to obtain the corresponding open-circuit voltage of the battery. The

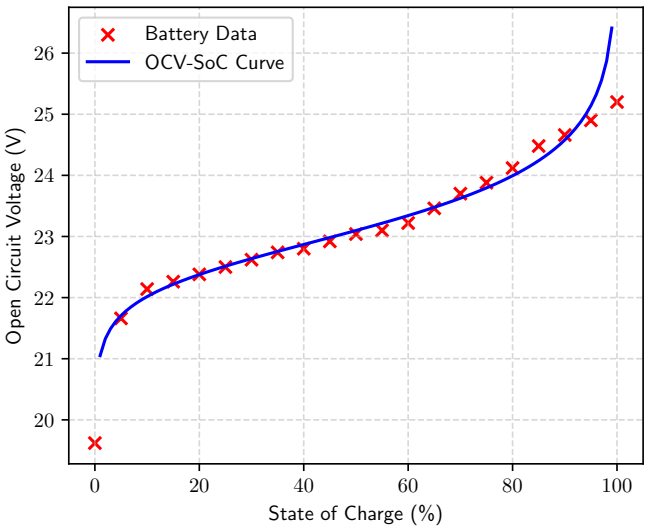


Figure 3 Fitted SoC vs OCV Curve for the 6S1P 22Ah Battery.

SoC at any given time is estimated using coulomb counting (also known as Ampere-Hour integration), which estimates the SoC by measuring the amount of charge and discharge using the following equation:

$$\mathrm{SoC}(t) = \mathrm{SoC}(t_0) - \frac{\eta}{C_n} \int_{t_0}^{t} I(t)dt, \tag{15}$$

where $\mathrm{SoC}(t_0)$ is the initial state of charge, η represents the coulumbic efficiency, C_n represents the battery capacity or rated capacity, and $I(t)$ is the instantaneous current discharged from the battery.

Finally, the derived current profile, along with the current battery voltage value and the future flight time horizon, is fed into the data-driven battery model. This learning-based model predicts the voltage and SoC evolution along the anticipated flight trajectory. A detailed description of the learning-based battery modeling is provided in Section 4.

3.3 Feasibility Assessment and Decision Making

After computing the power required to complete the remaining trajectory and predicting the battery's voltage trajectory, we assess mission success using the criterion provided in Equation 3. At each reassessment step, if the predicted voltage trajectory remains greater than or equal to the predefined threshold V_{thresh} throughout the entire prediction horizon, the aircraft continues its flight as planned. However, if the predicted voltage falls below the threshold at any point, the aircraft must reroute to the nearest warehouse or designated emergency landing site. The closest alternate landing site is determined by:

$$\mathbf{x}_e^* = \arg\min_{\mathbf{x}_e \in \mathcal{E}} d(\mathbf{x}_t, \mathbf{x}_e), \tag{16}$$

where $\mathcal{E}$ represents the set of predefined emergency landing sites, and $d(\mathbf{x}(t), \mathbf{x}_e)$ is the distance between the current position of the aircraft and each landing site. Once a new landing site is identified, the aircraft adjusts its trajectory accordingly and proceeds toward the new landing site. A feasibility assessment is then conducted for the updated trajectory to ensure that the aircraft can safely reach the new destination.

Algorithm 1 Power-to-Current Conversion Process.

Procedure `Power-to-CurrentConversion()`:

Input : Power profile $P(t)$, Battery parameters (R_{int}, η, C_n, k_0, k_1, and k_2)

Output : Current profile $I(t)$

1 **for** *each timestep* t **do**

2 Compute open circuit voltage (OCV) from SoC using equation 14

$$V_{\text{oc}}(t) \leftarrow k_0 + k_1 \ln(\text{SoC}(t)) + k_2 \ln(1 - \text{SoC}(t))$$

3 Solve quadratic equation 13

$$I(t) \leftarrow \frac{-V_{\text{oc}}(t) + \sqrt{V_{\text{oc}}(t)^2 - 4R_{\text{int}}P(t)}}{2R_{\text{int}}}$$

4 Update SoC using Coulomb counting

$$\text{SoC}(t + \Delta t) \leftarrow \text{SoC}(t) - \frac{\eta}{C_n} I(t)\Delta t$$

5 End of operation

4 Battery Modeling

To enable accurate, data-driven modeling of battery voltage under future current loads, we implemented a Neural Ordinary Differential Equation (Neural ODE) approach [6]. Neural ODEs generalize traditional neural networks by replacing discrete layer-wise transformations with a continuous-time formulation, where the hidden state $h(t)$ evolves according to a learned differential equation:

$$\frac{dh}{dt} = f(h(t), t; \theta) \tag{17}$$

Here, f is a neural network parameterized by θ that defines the dynamics of the hidden state over time. In the context of battery modeling, the input current profile and initial voltage are encoded into the initial hidden state $h(0)$, which is then evolved forward in time using an ODE solver. At each time step, the evolving hidden state is used to predict the corresponding battery voltage. This framework naturally supports irregular time sampling and produces smooth, physically coherent predictions, making it particularly effective for capturing the dynamics of battery behavior under variable loads.

In this section, we describe the procedures followed to develop the Neural-ODE-based battery model. The overall training workflow for the Neural ODE-based battery model is illustrated in the schematic diagram shown in Figure 4. As depicted in the figure, the modeling process includes dataset generation, construction of training and test sets, model training using the training set, and performance evaluation using the test set.

4.1 Dataset Generation

To develop and validate a data-driven battery voltage prediction model, we adopted a data generation procedure designed to capture the dynamic behavior of a lithium-ion battery under diverse load conditions. This procedure integrates a high-fidelity electrochemical battery

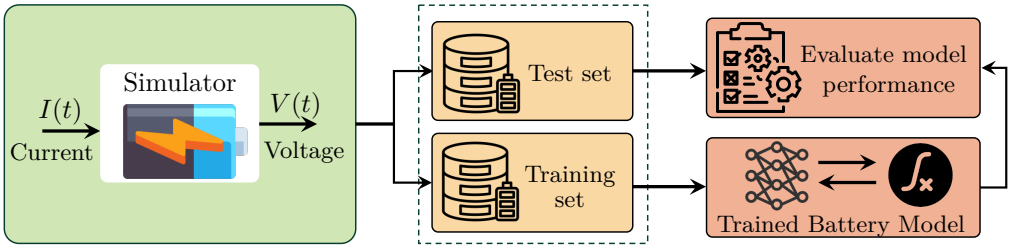


Figure 4 Neural ODE-based battery model training processes.

model with a current profile generation mechanism to emulate realistic operational scenarios in drone package delivery missions. The goal is to produce realistic current and voltage profiles that reflect battery performance during flight operations. The data generation approach consists of three main stages: *(i)* flight mission current profile generation, *(ii)* simulation of battery voltage, and *(iii)* dataset construction.

4.1.1 Flight Mission Current Profile Generation

The current profile generation process produces two types of profiles: full-flight mission profiles and mid-flight constant profiles. Full-flight profiles capture both the takeoff and cruise phases of the aircraft operations, where the takeoff phase is characterized by a higher power demand, while the cruise phase exhibits a lower current draw. Mathematically, the generated current profile at time t, denoted as $I(t)$, is defined as:

$$I(t) = \begin{cases} I_{\text{takeoff}}, & 0 \leq t < T_{\text{takeoff}} \\ I_{\text{cruise}}, & T_{\text{takeoff}} \leq t < T_{\text{total}} \end{cases} \tag{18}$$

where $I_{\text{takeoff}} \sim U(140, 225)$ A and $I_{\text{cruise}} \sim U(50, 70)$ A, with $U(a, b)$ denoting a uniform distribution. The takeoff duration, T_{takeoff}, is randomly sampled within the range $[1, 10]$ seconds, ensuring variability in the generated profiles. Mid-flight profiles, on the other hand, are created by assigning the initial voltage at various points during a full-flight mission and applying a constant current profile from that point onward. This approach enables the evaluation of battery response under different initial conditions.

4.1.2 Battery Voltage Simulation

To simulate the voltage response corresponding to the generated current profiles, we employ an electrochemical battery model [8]. The simulation procedure involves initializing the battery state, iterating over the generated current profiles, and computing the corresponding voltage response. The resulting dataset consists of $1,000$ pairs of current and voltage trajectories and is systematically divided into training and test sets using a $70\% - 30\%$ split. Each entry in the dataset comprises the input current trajectory $I(t)$, the corresponding voltage response $V(t)$, and the associated time horizon T. Figure 5 illustrates representative samples from the generated dataset, with current trajectories shown in blue and their corresponding voltage responses in black.

4.2 Neural ODE Training

In this subsection, we provide details on the data preprocessing steps, as well as the architectural and training specifications of our Neural ODE-based battery model.

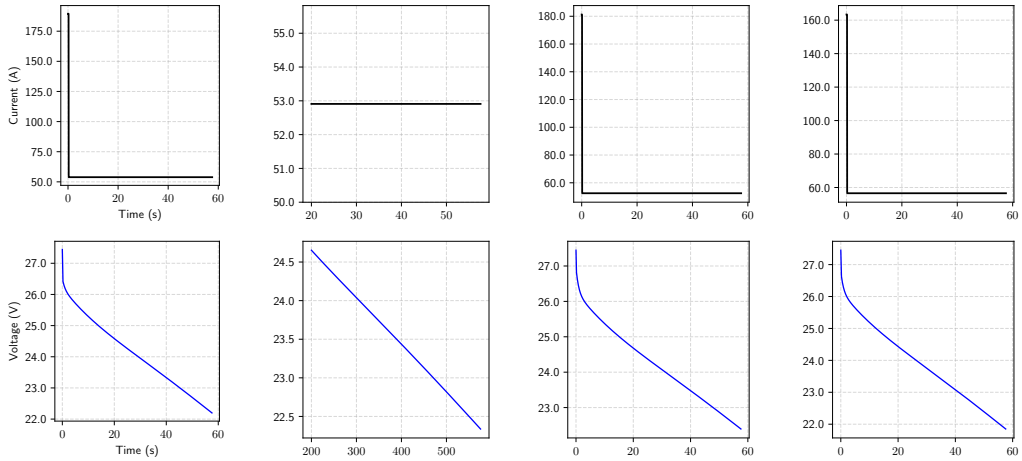


Figure 5 Representative current profiles (top row) and their corresponding voltage responses (bottom row).

4.2.1 Data Preprocessing

To ensure robust generalization across various scales of input data and to mitigate numerical instability during training, the input current and output voltage sequences are standardized using the training dataset's mean and standard deviation. The model operates on normalized time in the range $[0, 1]$ for each sequence.

4.2.2 Neural ODE Model Architecture

Our Neural ODE model consists of two primary components: a neural network-based ODE function and an ODE solver. The ODE function is implemented as a fully connected feedforward neural network with three hidden layers, each employing ReLU activation functions, and outputs the derivative of the voltage. The network takes as input the initial battery voltage, the input current, and the time horizon, enabling it to learn a continuous-time differential equation that governs the voltage dynamics. This learned function is then integrated over time using a fourth-order Runge-Kutta (RK4) solver to generate the battery voltage trajectory. The model is trained using a composite loss function that combines mean squared error (MSE) and root mean squared error (RMSE) between the predicted and ground truth voltage values. After training, the model is evaluated on test current profiles to assess its accuracy in predicting voltage trajectories. The architectural and training hyperparameter details of the model are provided in Tables 2 and 3, respectively. These parameters were identified through a series of empirical experiments, where different configurations were systematically tested to achieve optimal trade-offs between model complexity, training efficiency, and predictive performance.

5 Results and Discussion

5.1 Scenario Description

The scenario designed in this paper to evaluate the performance of the proposed real-time feasibility assessment and contingency management framework involves a drone package delivery operation within the Dallas–Fort Worth (DFW) metropolitan area, as illustrated

Table 2 Architecture of the ODE function.

Layer	# Neurons	Activation
Input	3	–
Layer 1	128	ReLU
Layer 2	128	ReLU
Layer 3	64	ReLU
Output	1	None

Table 3 Training hyperparameters.

Hyperparameter	Value
Optimizer	Adam
Learning Rate	0.001
Batch Size	4
ODE Solver	RK4
Epochs	100

in Figure 6. In this scenario, a single aircraft departs from a designated warehouse located at 33°08′48″N, 96°48′22″W, flying towards an assigned delivery location at 33°09′05″N, 96°47′15″W. We assume all necessary pre-departure feasibility assessments have already been completed, and the mission has been cleared for execution. Once airborne, the online feasibility assessment method developed in this study, which is assumed to run offboard, operates at regular intervals of 5 seconds.

Additionally, to assess the effectiveness of our proposed approach in supporting online decision-making and contingency management, we introduce a realistic in-flight anomaly scenario. In this case, the aircraft experiences an anomaly upon reaching the midpoint of the mission, located at coordinates 33°08′56.5″N, 96°47′48.5″W. Due to thermal stress affecting the electronic speed controller (ESC), the cruise speed must be reduced from the nominal 5 m/s to 3 m/s. This reduction significantly extends the expected flight duration and increases energy consumption, potentially rendering the original flight plan infeasible.

In response, the aircraft must perform a feasibility assessment under the new flight condition and dynamically reroute to one of several predefined emergency landing sites to ensure safety. These alternative landing sites are located at 33°08′40″N, 96°48′12″W; 33°09′04″N, 96°47′45″W; and 33°09′12″N, 96°47′11″W. As shown in Figure 6, these locations are labeled Emergency Landing 1, Emergency Landing 2, and Emergency Landing 3, respectively. This scenario provides a rigorous testbed for evaluating the framework's capability to forecast infeasibility and adapt flight decisions in real time.

5.2 Results

Because our framework conducts the feasibility assessment in two steps – first predicting the power profile and then predicting the battery voltage – we evaluate the accuracy of each step independently. To validate the predicted power profiles, we use the detailed aircraft model presented in [1] as our ground truth. Similarly, for battery voltage performance evaluation, we employ the detailed electrochemical Li-ion battery model described in [8] as the reference. Furthermore, since the online feasibility assessment occurs at fixed time intervals, voltage predictions made at each timestep are visualized using distinct colors, with each color corresponding to the prediction profile generated at a specific feasibility assessment timestep.

5.2.1 Power Consumption Prediction Results

As discussed in Section 5.1, our package delivery scenario involves two main flight phases: one before the mid-flight incident and another after the incident. Before the mid-flight incident, the aircraft is executing the original flight plan which is from the warehouse to the assigned destination (referred to as *"Long"*). However, once the mid-flight incident is

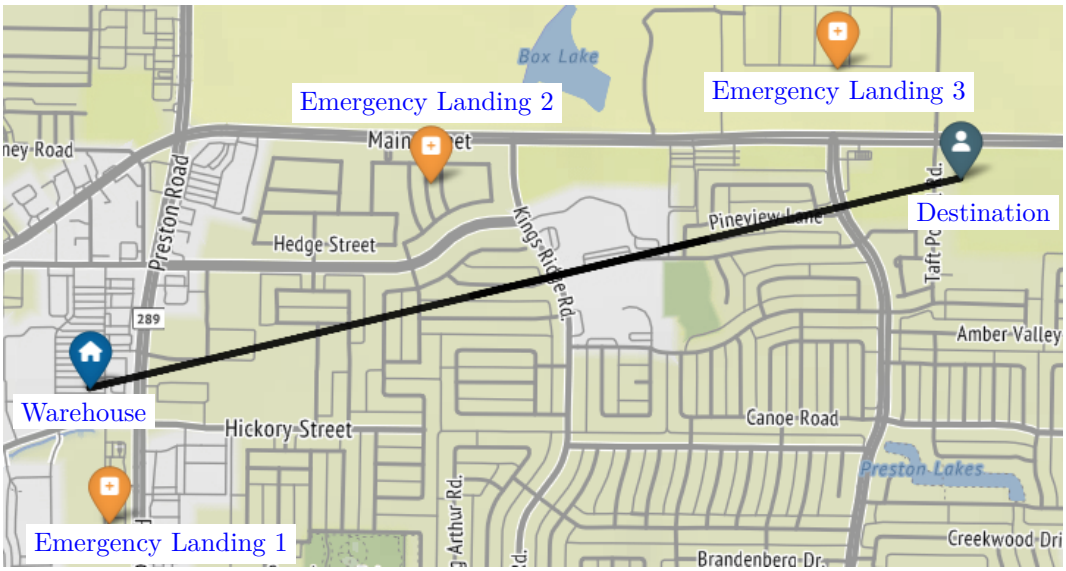


Figure 6 The developed package delivery scenario illustrating the original flight plan, warehouse, destination, and the three emergency landing sites.

identified and the speed change is applied, the aircraft needs to assess the feasibility of the original flight plan with the newly updated speed and if it's not feasible, identify the nearest emergency landing site from the already pre-defined set of locations and fly towards it while still performing the feasibility assessment at every 5 seconds.

Following the mid-flight incident, four alternative flight plans are considered, each corresponding to a potential landing site: the original destination and the three emergency landing sites. These include: (i) the flight plan to the original destination after the incident, which is expected to be infeasible and is labeled *"Infeasible"*; (ii) the flight plan to the nearest emergency landing site, labeled *"Short"*; and (iii) the flight plans to the other two emergency landing sites, labeled *"EM1"* and *"EM3"*, respectively.

Figure 7 presents a comparison between the actual aircraft power consumption profiles and the approximated power profiles for three flight plans: *"Long"*, *"Infeasible"*, and *"Short"*. These approximations are obtained using the method described in Section 3.1. Since the mid-flight incident requiring cruise speed reduction occurs at 288 seconds, the second and third plots show the power profiles only for the remaining flight duration from the moment the speed reduction is applied. Furthermore, the lower three plots in Figure 7 compare the actual current profiles against those derived from the predicted power trajectories using the conversion technique outlined in Section 3.2. This analysis enables a direct evaluation of the prediction pipeline's ability to infer current demands under altered flight conditions.

Because the reliability of our decision-making and contingency management framework heavily depends on the predictive accuracy of the power consumption model and the power-to-current conversion process, we evaluate the performance of these key components in Table 4. The table presents the results of 30 simulation runs for each of all flight plans. For each case, we report the average root mean squared error (RMSE) and mean absolute error (MAE) between the model-based reference approach and the method proposed in this paper. The results demonstrate that the proposed framework is capable of predicting both power and current profiles with reasonable accuracy.

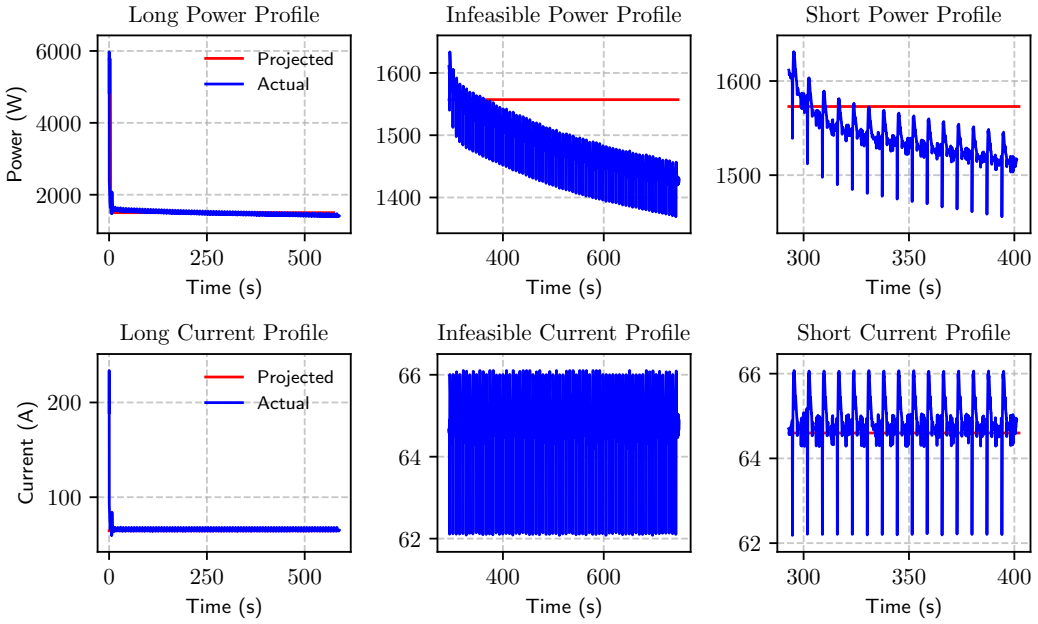


Figure 7 Power and current profiles for the three flight plans (Long, Infeasible, and Short). The top three plots illustrate the power profiles corresponding to each flight plan, while the bottom three plots show the current profiles derived from these power profiles.

5.2.2 Voltage Prediction Results

Once the current profiles for the future flight duration at each timestep are obtained, they are fed into the trained data-driven battery model to predict the corresponding voltage trajectories. Figures 8 and 9 present the predicted voltage trajectories for all flight plans using the Neural-ODE-based approach, alongside the reference battery voltage trajectories generated by simulating the detailed battery model. For each voltage prediction shown in Figures 8 and 9, feasibility assessment is performed at 5-second intervals using the criterion described in Section 1. The minimum voltage threshold for mission feasibility is set at 18 V, meaning the predicted voltage trajectory must remain above this threshold throughout the entire operational time horizon for the mission to be considered safe.

As illustrated in Figure 8, the predicted voltage trajectory for the original flight plan remains above the 18 V threshold at all times and is therefore considered feasible – until the mid-flight incident occurs. However, as shown in the middle plot of Figure 8, once the incident triggers a reduction in cruise speed to 3 m/s, the voltage trajectory violates the threshold at approximately 680 seconds. This indicates that the aircraft can no longer safely complete its flight to the assigned destination. Consequently, the mission is rerouted to the emergency landing site, and feasibility is re-evaluated using the current profile for the short flight plan with the Neural-ODE-based battery model. The result, shown in the final plot of Figure 8, demonstrates that the revised mission remains feasible under the updated flight conditions, thereby ensuring the safety of the aircraft.

To provide a more comprehensive understanding of the mid-flight incident and the rationale behind the chosen contingency plan, Figure 9 presents the voltage predictions for the two alternative flight plans directed toward the other emergency landing sites (*"EM1"* and

Table 4 Comparison of prediction accuracy (RMSE and MAE) for power and current profiles across all flight plans: Long, Infeasible, Short, EM1, and EM3.

Profile	Error Type	Long	Infeasible	Short	EM1	EM3
Power (W)	RMSE	295.01	83.82	39.41	75.36	75.37
	MAE	68.80	74.84	35.06	66.89	66.80
Current (A)	RMSE	3.76	0.63	0.60	0.63	0.63
	MAE	0.94	0.39	0.38	0.39	0.39

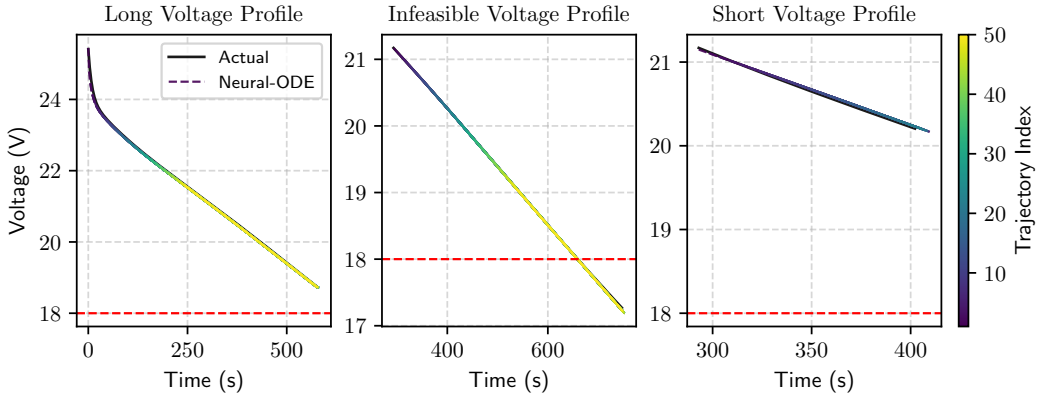


Figure 8 Voltage predictions for the three flight plans (Long, Infeasible, and Short), performed at 5-second intervals. Each voltage profile corresponds to a specific prediction index, representing the voltage trajectory forecasted at a given feasibility assessment timestep.

"EM3"). The results clearly demonstrate that both of these trajectories become infeasible, as the predicted battery voltage drops below the minimum operational threshold. This confirms the validity of the selected rerouting strategy to the nearest emergency landing site (*"Short"*).

Furthermore, Figure 10 offers a spatial visualization of the voltage predictions overlaid on the mission map. In this figure, each flight plan is illustrated using a heatmap that encodes predicted battery voltage along the flight trajectory. As shown, the battery begins fully charged at the warehouse, but following the mid-flight incident, the voltage predictions for all flight plans – except for the one directed to Emergency Landing 2 – fall below the critical threshold of 18 V at some point along the trajectory. This spatial voltage analysis further reinforces the feasibility of the re-routing decision and highlights the framework's efficacy in supporting real-time contingency management.

To evaluate the performance of the Neural-ODE-based battery model used for online feasibility assessment, we examine both its prediction accuracy and computational efficiency. For benchmarking purposes, we developed a physics-informed neural network (PINN) [19] based battery model, which is considered a state-of-the-art approach for learning battery dynamics. The PINN model implemented in this study combines a long short-term memory (LSTM) network with an equivalent circuit-based battery model adopted from [14]. This approach enhances the predictive capabilities of the data-driven LSTM by embedding physical laws – specifically, the voltage-current relationships described by the equivalent circuit – directly into the training process. Rather than relying solely on data, the PINN

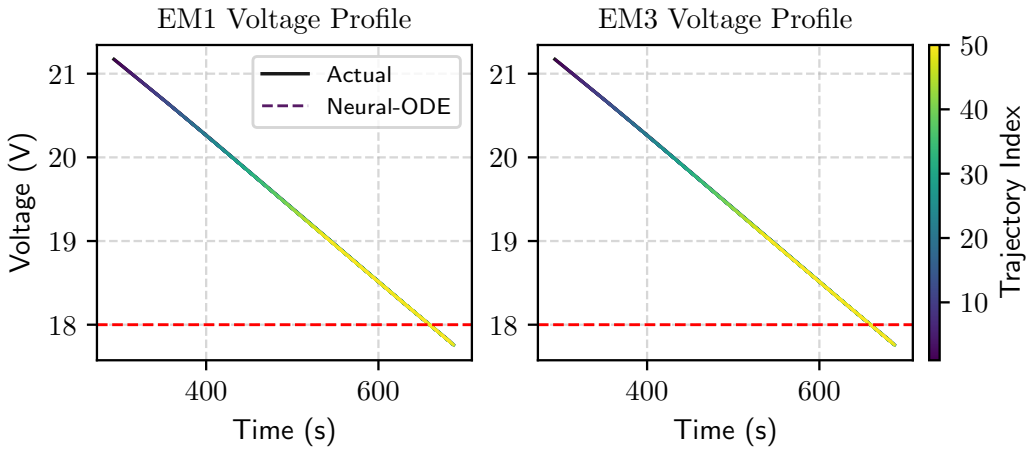


Figure 9 Voltage predictions for flight plans going to emergency landing 1 and emergency landing 3. Each voltage profile corresponds to a specific prediction index, representing the voltage trajectory forecasted at a given feasibility assessment timestep.

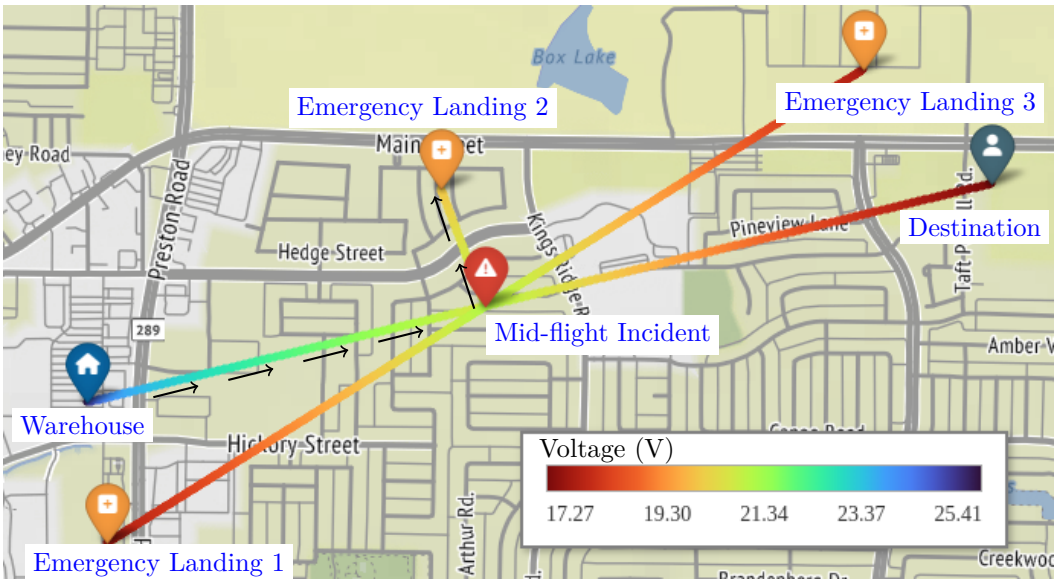


Figure 10 Spatial visualization of voltage predictions along each flight trajectory. Arrows indicate the aircraft's flight direction before and after the mid-flight incident.

minimizes a composite loss function that penalizes both data mismatch and violations of the governing physical equations. This integration of domain knowledge improves model generalization, increases robustness to noise, and enables physically consistent predictions even in extrapolated conditions.

Figure 11 presents a comparison between the Neural-ODE and PINN-based approaches for the three flight plans. As shown in the left plot, the Neural-ODE-based model consistently outperforms the PINN model in terms of prediction accuracy across all predictions. These differences in prediction accuracy have important implications for the safety and efficiency of aircraft operations. Specifically, for the long flight plan, the PINN-based feasibility

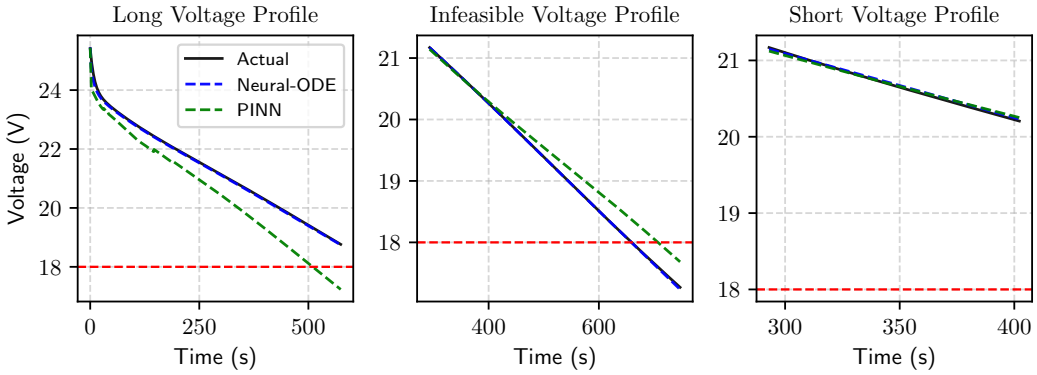


Figure 11 Performance comparison between the Neural-ODE and PINN-based battery models relative to the actual voltage trajectories across all three flight plans.

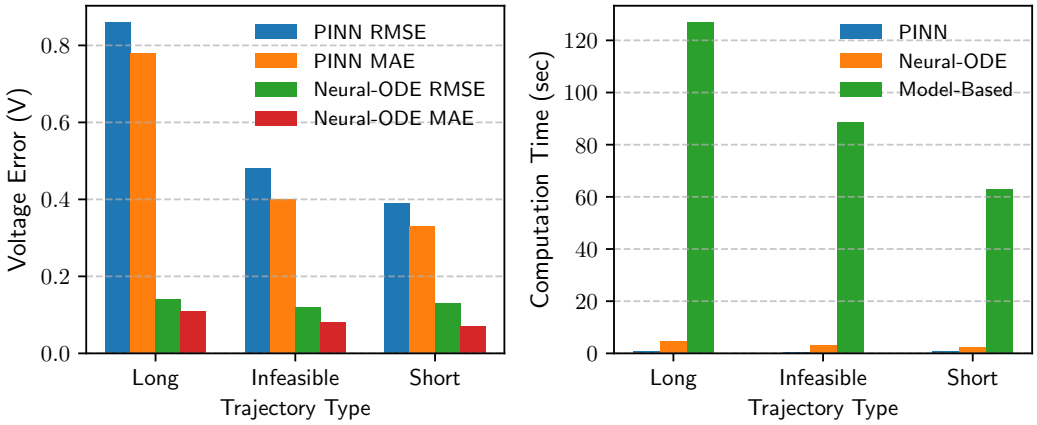


Figure 12 Comparison of accuracy and computational cost between the Neural-ODE and PINN-based battery models. The left plot shows prediction accuracy relative to the ground truth voltage trajectories for all three flight plans, while the right plot presents the average computational time required for feasibility assessments.

assessment would incorrectly classify the mission as infeasible due to its prediction inaccuracies. Conversely, in the case of the infeasible flight plan, the PINN model would fail to detect the voltage threshold violation, potentially resulting in an unsafe decision to proceed with the mission.

In addition to accuracy, computational efficiency is a critical factor, as the framework is intended for real-time use during flight, where computational resources are limited. To assess this, the right plot in Figure 12 shows the average computational time required to perform feasibility assessments for the three flight plans (Long, Infeasible, and Short), comparing the model-based approach with the proposed Neural-ODE approach. All experiments were conducted on a 3.20 GHz Intel Xeon(R) CPU with 125.4 GB of RAM. The results show that the proposed approach is approximately 36 times faster than the model-based method in the long flight plan case, with a maximum computation time of 3.76 seconds. This demonstrates that the proposed framework is well-suited for in-flight operation, where feasibility assessments must be performed every 5 seconds.

5.3 Discussion and Lessons Learned

This study provides key insights that may guide future researchers in developing real-time flight mission feasibility assessment frameworks. By implementing and evaluating our proposed framework in a realistic drone package delivery scenario, we identified several important observations and lessons learned.

- **Decoupling feasibility assessment into power consumption and voltage trajectory prediction enhances flexibility and accuracy.** The proposed framework decomposes the feasibility assessment process into two stages: power consumption prediction and voltage trajectory prediction. This separation allows greater flexibility in selecting and improving prediction models, leading to higher accuracy and computational efficiency. As demonstrated in the results, this structured approach achieves both prediction accuracy and computational efficiency required for real-time feasibility assessment.
- **The choice of the power consumption model significantly impacts performance.** The efficacy of feasibility assessment is highly dependent on the accuracy of the power consumption model. In this work, we adopted a momentum theory-based power consumption model for multirotor aircraft and fine-tuned it using experimental data. The decision to develop or adopt a power consumption model should consider key factors such as accuracy requirements, computational efficiency, and environmental conditions (e.g., wind effects). Selecting an appropriate model is crucial for ensuring reliable power predictions.
- **Power-to-current conversion must account for battery behavior.** Given the extended prediction horizon required in our study (approximately 10 minutes), power-to-current conversion must incorporate battery dynamics. Our approach utilizes an open-circuit voltage (OCV) vs state of charge (SoC) relationship modeled using the Nernst equation, combined with Coulomb counting, to achieve an accurate conversion. Careful modeling of this process is essential to maintain prediction reliability over long-duration flights.
- **The choice of battery modeling technique affects prediction accuracy.** Given that battery behavior is inherently governed by differential equations, we adopted a Neural ODE-based model to learn the underlying battery dynamics. To benchmark its performance, we compared it against other time-series prediction techniques, including physics-informed neural networks (PINNs), which combine long short-term memory (LSTM) networks with an equivalent circuit-based battery model. Our results indicate that the Neural ODE-based approach more accurately captures battery voltage trajectories, making it a promising candidate for the feasibility assessment of dynamical systems.

6 Conclusion

In this paper, we address the problem of online flight mission feasibility assessment for sUAS operations. Unexpected in-flight events can introduce significant safety risks if not properly managed. To mitigate these risks, we propose a framework that continuously monitors battery status and makes real-time decisions to prevent energy insufficiency. The framework consists of two main components: power consumption prediction and battery voltage trajectory prediction. Power consumption prediction is performed using a model based on momentum theory, while voltage trajectory prediction leverages a Neural Ordinary Differential Equation (Neural ODE)-based data-driven model. By integrating these two components, the system evaluates mission feasibility in real time and determines whether to continue the flight or initiate rerouting. We evaluate the framework's performance in a drone delivery scenario in the Dallas–Fort Worth (DFW) area, where the aircraft encounters

an unexpected energy depletion event mid-flight. The results show that the framework accurately predicts power profiles and voltage trajectories for the remaining flight duration. Additionally, its computational efficiency makes it feasible for real-time flight monitoring and contingency management. Future work will incorporate additional sources of uncertainty, such as wind disturbances and noise in the battery model, to improve prediction accuracy and decision-making capabilities. We will also extend the battery model to account for degradation and state of health effects.

References

1 Ibrahim Ahmed, Marcos Quinones-Grueiro, and Gautam Biswas. A high-fidelity simulation test-bed for fault-tolerant octo-rotor control using reinforcement learning. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2022. doi:10.1109/DASC55683.2022.9925862.

2 AMPOW. Lipo voltage chart: Show the relationship of voltage and capacity, 2023. Accessed: 2025-03-06. URL: https://blog.ampow.com/lipo-voltage-chart/.

3 Luca Biggio, Tommaso Bendinelli, Chetan Kulkarni, and Olga Fink. Ageing-aware battery discharge prediction with deep learning. *Applied Energy*, 346:121229, 2023.

4 Zdeněk Bouček and Miroslav Flídr. Mission planner for uav battery replacement. In *2024 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 1–6. IEEE, 2024.

5 Ephrem Chemali, Phillip J Kollmeyer, Matthias Preindl, Ryan Ahmed, and Ali Emadi. Long short-term memory networks for accurate state-of-charge estimation of li-ion batteries. *IEEE Transactions on Industrial Electronics*, 65(8):6730–6739, 2017. doi:10.1109/TIE.2017.2787586.

6 Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

7 Arnav Choudhry, Brady Moon, Jay Patrikar, Constantine Samaras, and Sebastian Scherer. Cvar-based flight energy risk assessment for multirotor uavs using a deep energy model. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 262–268. IEEE, 2021. doi:10.1109/ICRA48506.2021.9561658.

8 Matthew Daigle and Chetan S Kulkarni. Electrochemistry-based battery modeling for prognostics. In *Annual Conference of the PHM Society*, volume 5(1), 2013. doi:10.36001/phmconf.2013.v5i1.2252.

9 Aditya Devta, Isabel C Metz, and Sophie F Armanini. Experimental evaluation of bird strikes in urban air mobility. *arXiv preprint arXiv:2308.13022*, 2023. doi:10.48550/arXiv.2308.13022.

10 Marc Doyle, Thomas F Fuller, and John Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of the Electrochemical society*, 140(6):1526, 1993.

11 Federal Aviation Administration. Package delivery by drone (part 135), 2023. Accessed: 2024-10-19. URL: https://www.faa.gov/uas/advanced_operations/package_delivery_drone.

12 Federal Aviation Administration. Faa makes drone history in dallas area, 2024. Accessed: 2024-10-19. URL: https://www.faa.gov/newsroom/faa-makes-drone-history-dallas-area.

13 George E Gorospe Jr, Chetan S Kulkarni, Edward Hogge, Andrew Hsu, and Natalie Ownby. A study of the degradation of electronic speed controllers for brushless dc motors. In *Asia Pacific Conference of the Prognostics and Health Management Society 2017*, volume ARC-E-DAA-TN42858, 2017.

14 Hongwen He, Rui Xiong, and Jinxin Fan. Evaluation of lithium-ion battery equivalent circuit models for state of charge estimation by an experimental approach. *energies*, 4(4):582–598, 2011.

15 Dickson NT How, MA Hannan, MS Hossain Lipu, and Pin Jern Ker. State of charge estimation for lithium-ion batteries using model-based and data-driven methods: A review. *Ieee Access*, 7:136116–136136, 2019. doi:10.1109/ACCESS.2019.2942213.

16 Yves Le Marquand. Faa authorises zipline and wing for bvlos operations in dallas, 2023. Accessed: 2024-10-19. URL: https://www.revolution.aero/news/2024/07/30/faa-authorises-zipline-and-wing-for-bvlos-operations-in-dallas/.

17 Jon Ander Martin, Justin N Ouwerkerk, Anthony P Lamping, and Kelly Cohen. Comparison of battery modeling regression methods for application to unmanned aerial vehicles. *Complex Engineering Systems*, 2, 2022.

18 Adnan Nuhic, Tarik Terzimehic, Thomas Soczka-Guth, Michael Buchholz, and Klaus Dietmayer. Health diagnosis and remaining useful life prognostics of lithium-ion batteries using data-driven methods. *Journal of power sources*, 239:680–688, 2013.

19 Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019. doi:10.1016/J.JCP.2018.10.045.

20 Dapai Shi, Jingyuan Zhao, Zhenghong Wang, Heng Zhao, Chika Eze, Junbin Wang, Yubo Lian, and Andrew F Burke. Cloud-based deep learning for co-estimation of battery state of charge and state of health. *Energies*, 16(9):3855, 2023.

21 Guangyao Shi, Nare Karapetyan, Ahmad Bilal Asghar, Jean-Paul Reddinger, James Dotterweich, James Humann, and Pratap Tokekar. Risk-aware uav-ugv rendezvous with chance-constrained markov decision process. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 180–187. IEEE, 2022. doi:10.1109/CDC51059.2022.9993358.

22 Mostafa M Shibl, Loay S Ismail, and Ahmed M Massoud. A machine learning-based battery management system for state-of-charge prediction and state-of-health estimation for unmanned aerial vehicles. *Journal of Energy Storage*, 66:107380, 2023.

23 Gina Sierra, M Orchard, Kai Goebel, and C Kulkarni. Battery health management for small-size rotary-wing electric unmanned aerial vehicles: An efficient approach for constrained computing platforms. *Reliability Engineering & System Safety*, 182:166–178, 2019. doi:10.1016/J.RESS.2018.04.030.

24 Ellis L Thompson, Abenezer G Taye, Wei Guo, Peng Wei, Marcos Quinones, Ibrahim Ahmed, Gautam Biswas, Jesse Quattrociocchi, Steven Carr, Ufuk Topcu, et al. A survey of evtol aircraft and aam operation hazards. In *AIAA AVIATION 2022 Forum*, page 3539, 2022.

Optimized Spectral Fault Receptive Fields for Diagnosis-Informed Prognosis

Stan Muñoz Gutiérrez[1]
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Franz Wotawa[2]
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Abstract

This paper introduces Spectral Fault Receptive Fields (SFRFs), a biologically inspired technique for degradation state assessment in bearing fault diagnosis and remaining useful life (RUL) estimation. Drawing on the center-surround organization of retinal ganglion cell receptive fields, we propose a frequency-domain feature extraction algorithm that enhances the detection of fault signatures in vibration signals. SFRFs are designed as antagonistic spectral filters centered on characteristic fault frequencies, with inhibitory surrounds that enable robust characterization of incipient faults under variable operating conditions. A multi-objective evolutionary optimization strategy based on NSGA-II algorithm is employed to tune the receptive field parameters by simultaneously minimizing RUL prediction error, maximizing feature monotonicity, and promoting smooth degradation trajectories. The method is demonstrated on the XJTU-SY bearing run-to-failure dataset, confirming its suitability for constructing condition indicators in health monitoring applications. Key contributions include: (i) the introduction of SFRFs, inspired by the biology of vision in the primate retina; (ii) an evolutionary optimization framework guided by condition monitoring and prognosis criteria; and (iii) experimental evidence supporting the detection of early-stage faults and their precursors. Furthermore, we confirm that our diagnosis-informed spectral representation achieves accurate RUL prediction using a bagging regressor. The results highlight the interpretability and principled design of SFRFs, bridging signal processing, biological sensing principles, and data-driven prognostics in rotating machinery.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics; General and reference → Reliability; Computing methodologies → Genetic algorithms; Computing methodologies → Machine learning; Hardware → Failure prediction

Keywords and phrases Health Perception, Spectral Fault Receptive Fields, Remaining Useful Life, Incipient Fault Diagnosis, Prognostics and Health Management, Condition Monitoring, Evolutionary Multi-Objective Optimization, Bagged Regression Tree Ensemble, Bearing Fault Diagnosis

Digital Object Identifier 10.4230/OASIcs.DX.2025.9

Related Version *Previous Version*: `https://doi.org/10.48550/arXiv.2506.12375`

Supplementary Material
Software (MATLAB Notebook): `https://doi.org/10.5281/zenodo.15660819` [24]
Other (Poster): `https://doi.org/10.5281/zenodo.17141147`

Funding This study was conducted within the framework of the ARCHIMEDES project, which is supported by the Chips Joint Undertaking and its members, including top-up funding from National Authorities under Grant Agreement No. 101112295 and the FFG under Grant Agreement No. FO999899377.

Co-funded by the European Union

Acknowledgements SMG thanks Mike Denham for introducing him to the Biology of Vision and Ian Parmee for introducing him to Evolutionary Multi-Objective Engineering Design.

[1] Authors are listed in alphabetical order.
[2] Corresponding author

1 Introduction

In modern engineering, reliability is a central concern, distinguished from quality by its emphasis not only on compliance with specifications at "time zero," but also on sustained performance throughout an artifact's operational life. Central to reliability is the assessment and modeling of degradation rates and time to failure [19]. Reliability pertains to the performance and operation of systems and their components, aiming to deliver solutions that can operate without failure, nor be the cause of failure, over a specified time horizon and in accordance with specifications that define both constraints and operational conditions.

Rotary machines are ubiquitous in industrial and transportation contexts. Bearings, as key components of these machines, play a crucial role in ensuring reliable operation. Accurately estimating the degradation state of bearings throughout their operational life is essential for rational decision-making by both humans and automated systems. Such decisions include scheduling maintenance actions, investigating accelerated degradation trends, predicting component failure times, implementing closed-loop control for safety and energy efficiency, and, when done effectively, extending component lifetimes through feedback-driven control based on degradation state estimation.

As part of Project Archimedes, we investigate intelligent, data-driven methods for precise degradation state estimation and remaining useful life prediction, aiming to support decisions that extend the operational lifespan of electric vehicle powertrains (EVPs). A significant challenge in this research area is that existing knowledge on bearing diagnostics primarily pertains to bearings operating under constant conditions, which does not reflect the dynamic speed and load variations encountered in electric vehicle drives amid disturbances and aleatoric uncertainties. Although our study does not explicitly address these dynamic conditions, our approach is designed to integrate such parameters, whose implications we briefly discuss. Another critical challenge is the limited availability of open datasets; currently, only a few datasets capture run-to-failure data for electric drive components, which constrains research progress. For this study, we utilize the XJTU-SY dataset [36], one of the few publicly available run-to-failure datasets for rolling element bearings, to validate our proposed methods.

The work presented in this paper focuses on bearings, which are integral to the mechanisms that connect the electric drive to the transmission and enable vehicle propulsion. However, we have developed our model to be generalizable to some extent, making it potentially applicable to other aspects of the electric drive, such as electric winding faults, irreversible demagnetization of permanent magnets, and inverter degradation dynamics, assuming suitable adaptations are implemented.

Our work introduces a novel technique based on consolidated knowledge within the field of vibration analysis. Although many recent research efforts adopt tabula rasa methodologies, bypassing established domain knowledge in favor of black-box systems that often achieve high performance, these solutions frequently lack transparency and interpretability. In safety-critical domains such as electric drives, transparency is essential; certification requires that system behavior be understandable and trustworthy.

Our method, named Spectral Fault Receptive Fields, offers an interpretable technique to degradation state estimation, with condition indicators that correspond directly to specific failure modes in bearings. We evaluated the system primarily using the monotonicity criterion, and further incorporated smoothness and remaining useful life (RUL)-based metrics for parameter selection. Through qualitative comparison, we demonstrated clear improvements resulting from explicit multi-objective optimization of several system parameters, thereby validating the effectiveness of the approach.

This paper is organized to guide the reader through the development and validation of our approach. In Section 2, we review related work in diagnostics, prognostics, and health monitoring of engineered systems. Section 3 introduces the problem domain and dataset used for experimental evaluation. Section 4 details the design of Spectral Fault Receptive Fields (SFRFs), inspired by biological vision, as a novel feature extraction method for condition monitoring and remaining useful life prediction. Section 5 presents empirical evaluations and multi-objective optimization results demonstrating the proposed method's effectiveness. The discussion in Section 6 reflects on the biological foundations, parameter tuning, and future research directions. Finally, Section 7 concludes the paper by summarizing key contributions and implications for predictive maintenance.

2 Related Work

Traditionally, reliability assessments were based primarily on empirical field data derived statistically [5, 18]. These approaches were typically static, focused solely on random failures, often neglected underlying failure mechanisms, did not account for differences among vendors or specific devices in lifetime predictions, and excluded real-time condition monitoring [18]. In contrast, modern diagnostics and prognostics frameworks emphasize continuous degradation monitoring of components and systems. They employ a range of modeling strategies, including failure progression rates, physics-of-failure, statistical and probabilistic methods, and modeling of failure propagation between interconnected subsystems. This entails a more comprehensive and dynamic assessment of system health [35]. When the primary focus is on selecting optimal maintenance actions, predictive maintenance (PdM) serves as an appropriate conceptual framework. However, research in this area typically concentrates on two aspects, which are seldom addressed simultaneously: (1) predicting the time to failure, referred to as remaining useful life (RUL) prediction, and (2) optimizing maintenance strategies. Prognostics and health management (PHM) is conducive to informed decision-making and actions to keep systems in optimal operating condition. PHM is an integrated, modular process that includes system analysis, data acquisition, data processing, fault detection, fault diagnostics, failure prognostics, decision making, and maintenance scheduling [34]. A typical predictive maintenance workflow consists of the following steps: (1) data acquisition and organization, (2) data preprocessing, (3) development of a fault detection or prediction model, and (4) deployment and integration [16]. In this study, we focus primarily on step (3), which practitioners often divide into two sub-tasks: (i) the design of condition indicators and (ii) model training for fault detection or prediction tasks. The design of *condition indicators* (CIs) encompasses the computation and selection of features that correlate with the state of health of the system. A *health indicator* (HI) combines multiple condition indicators into a single efficient indicator that is highly informative of degradation [16, 25]. The separation of sub-tasks (i) and (ii) is instrumental in tackling the complexity of the problem, but often leads to suboptimality or extensive iterative improvements. We will address this concern in our contribution by means of multi-objective optimization methods that can inform the HI design, factoring in its prognostic efficacy.

For bearings, degradation is irreversible. Tracking the degradation state throughout the component's operational life can be effectively achieved with suitable sensors and signal processing techniques. The most prevalent failure mechanism under nominal conditions (where bearings are correctly installed and lubricated) is subsurface-originated spalling, which can be detected at an early stage using acoustic emission sensors[10]. Oil analysis sensors are highly effective for early detection of degradation in bearings and gearboxes,

as they quantify the accumulation of debris from the onset of wear processes [9]. While primarily limited to surface-related defects, the use of MEMS-based accelerometer sensors in our application domain enables cost-effective health monitoring solutions. Consequently, there is strong research interest in developing representations and algorithms capable of capturing early degradation and detecting incipient faults. Accelerometers are the standard transducers for helicopter gearbox condition monitoring, providing essential input to health and usage monitoring (HUM) systems. A healthy transmission exhibits a characteristic *fingerprint*, referred to as the *regular meshing components of the signal* [30]. For rolling bearings, the characteristic frequencies of their components are well established [29, 32], and their computation is readily available in standard predictive maintenance solutions [15]. Our work leverages this domain knowledge by explicitly computing representations that focus on the characteristic frequencies of bearing elements, aligning with established practices in vibration analysis and intelligent fault diagnosis for rotating machinery.

Faults can be classified based on their severity into three main categories: (1) abrupt, also known as stepwise fault; (2) incipient, also known as drifting; and (3) intermittent [26]. *Incipient faults* in bearings have weak signal signatures that are difficult to detect due to their stochastic nature, multiple transmission paths, and the aleatoric uncertainty present in mechanical systems [11]. Because of this, despite their well-understood spectral signatures, incipient fault detection remains an active area of research. Vibration signal analysis and modeling typically utilize degradation models that define at least two primary stages: (1) a flat, horizontal region corresponding to the healthy state, where remaining useful life (RUL) prediction is generally unreliable and arguably unnecessary, and (2) a degraded unhealthy state, which is the main focus of most analytical techniques for delivering accurate RUL estimates [13]. Piecewise linear models are often used to model this change in degradation dynamics [28]. Current research in bearing condition monitoring and prognosis is increasingly focused on extending the prediction horizon to encompass as much of the component's operational life as possible. In our present paper, we devise biologically inspired condition indicators that address the characterization of early degradation stages and not only correlate with manifested abrupt abnormalities.

There is a wide availability of vibration-based condition indicators in the literature. A taxonomy by Yan et al. [37] classifies them according to the representation domain: (1) time domain, (2) frequency domain, (3) time-frequency, and (4) wavelet. While this classification is not exhaustive, excluding some nonlinear feature extraction methods such as chaos-theoretic-based [11] and information-theoretic-based [32, 2], it nevertheless effectively represents the dominant approaches in the field. Among the most widely adopted characterizations are two statistical properties that can be computed regardless of the representational domain: (1) kurtosis-based, often spectral, and (2) RMS-based (with safety and vibration severity assessed by this metric, as in ISO 10816 [12]). Both are effective and can be used complementarily for different stages of the degradation process [6], while proven effective across diagnostic [23], condition monitoring [21], or prognostic [20] tasks. Our contribution builds on the frequency spectrum of vibration signals and is specifically designed for a low computational footprint, ensuring that it does not add significantly to the computational cost of the fast Fourier transform (FFT).

Our primary objective in this work is to engineer *health perception* systems capable of actively tracking the degradation state of bearings in alignment with defined cost and reliability constraints, thereby enabling accurate estimation of RUL. We adopt the term perception to underscore the biological inspiration behind our approach to CI construction. In biological systems, effective perception, of both the self and the environment, is essential

to survival: proprioception, homeostatic regulation [1], and autonomic functions support internal integrity and health, while environmental perception enables adaptive responses. Our approach subscribes to the principles of autopoiesis [17] and biological autonomy [22], where integrity is understood as an emergent property of systemic organization and constraint equilibria. This aligns well with closed-loop lifetime and degradation control algorithms [7].

To guide the construction of CIs, we draw inspiration from the biology of vision, specifically, the theory of center-surround opponency in the trichromatic visual system of primates, and adopt an adapted version of the difference-of-Gaussians (DoG) model, widely applied in both biological and artificial vision domains [33]. Derived from a novel transfer of computational models, we faced in our work the problem of appropriate parameterization of our Difference of Gaussians (DoG) method. Selecting the spectro-spatial scales relevant to bearing faults was achieved by relying on engineering judgement informed by field experience, a process we refer to as empirical parameter selection. We provide evidence that a DoG configured with these empirically chosen parameters encodes CIs effectively. To refine the model further for predictive-maintenance and prognostics applications, we optimised its parameters with a multiobjective genetic algorithm [4] guided by established condition-monitoring and prognosis criteria. Criteria must be quantifiable and provide foundations for certification [31]. Our results reveal a tradeoff, present among local Pareto-optimal front members, between the monotonicity criterion, widely advocated for health indicators [6, 27, 3, 8], and the accuracy of remaining-useful-life (RUL) predictions measured via normalised mean-squared error (MSE). Related work by Qin et al. [27] employs genetic programming to evolve an arithmetic condition indicator optimised for monotonicity within a Wiener stochastic framework that is subsequently refined through expectation-maximisation.

3 Problem Domain and Dataset

3.1 XJTU-SY Bearing dataset

The experimental evaluation uses the XJTU-SY[36] run-to-failure bearing dataset, consisting of three bearing groups operated at fixed speeds and loads. Vibration signals were recorded with acceleration sensors along *horizontal* and *vertical* axes. Key bearing parameters include: *inner raceway diameter* ($D_I = 29.30$ mm), *outer raceway diameter* ($D_O = 39.80$ mm), *pitch diameter* ($D_P = 34.55$ mm), *ball diameter* ($D_B = 7.92$ mm), and *contact angle* ($\phi = 0°$), as illustrated in Figure 1. Bearings are uniquely labeled (e.g., "Bearing1_1"), with data organized into snapshots (vibration signal temporal windows) taken at regular intervals.

3.2 Vibration Signatures of Bearings

We are interested in monitoring the degradation of the different elements of a bearing to detect incipient faults. Surface defects in these elements produce well-understood vibration signatures at characteristic frequencies, determined by the bearing's geometry and operational speed. Vibration signature analysis fundamentally depends on monitoring changes in vibration near the characteristic frequencies of bearings. As degradation progresses, the activity within these frequency bands evolves, reflecting the bearing's health state. Building on this established principle, our processing pipeline begins by computing these characteristic bands. While increased excitation in these bands is a hallmark of bearing defects, such activity can also be present throughout the bearing's operational life. For clarity and brevity, we refer to these as *fault frequency bands*, though their activity is not exclusively associated with faulty conditions, as some excitation is present throughout the bearing's operational life. Figure 2 shows the default sidebands obtained by MATLAB.

Table 1 Characteristic frequencies related to bearing faults. BPFO = Ball Pass Frequency Outer Race; BPFI = Ball Pass Frequency Inner Race; BSF = Ball Spin Frequency; FTF = Fundamental Train Frequency (Cage). N_B: number of rolling elements; D_B: ball diameter; $D_p = \frac{D_I + D_O}{2}$: pitch diameter ; ϕ: contact angle; f_r: shaft rotational frequency.

Acronym	Equation
BPFO	$f_{\text{BPFO}} = f_r \frac{N_B}{2} \left[1 - \frac{D_B}{D_P} \cos\phi\right]$
BPFI	$f_{\text{BPFI}} = f_r \frac{N_B}{2} \left[1 + \frac{D_B}{D_P} \cos\phi\right]$
BSF	$f_{\text{BSF}} = f_r \frac{D_P}{2D_B} \left[1 - \left(\frac{D_B}{D_P} \cos\phi\right)^2\right]$
FTF	$f_{\text{FTF}} = \frac{f_r}{2} \left[1 - \frac{D_B}{D_P} \cos\phi\right]$

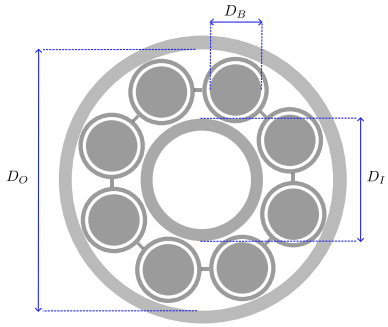


Figure 1 Schematic diagram of bearing geometry and parameters.

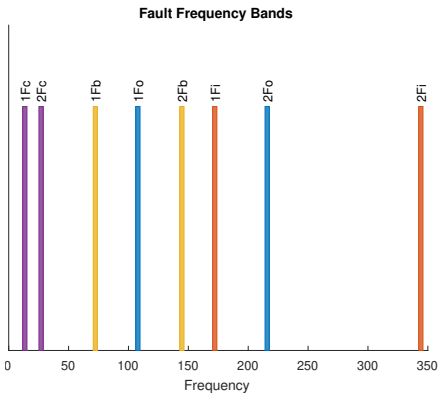


Figure 2 Fault Frequency Bands for the first and second harmonics. Notation: nF: the n-th harmonic for frequency F, $F \in \{\text{Fo}, \text{Fi}, \text{Fc}, \text{Fb}\}$. Frequencies are Fo: BPFO ($f_{\text{BPFO}} = 107.9074$), Fi: BPFI ($f_{\text{BPFI}} = 172.0926$), Fc : FTF ($f_{\text{FTF}} = 13.4884 Hz$), and Fb: BSF ($f_{\text{BSF}} = 72.3300$).

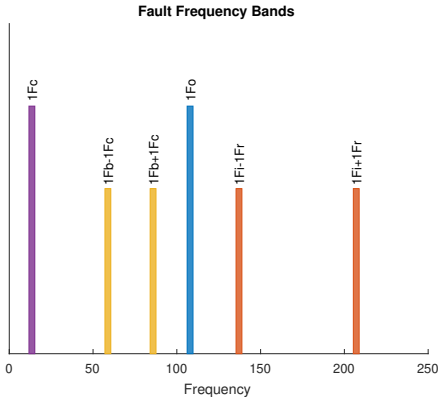


Figure 3 Fault Frequency Sidebands. Notation: nFb-mFc: the m-th negative sideband of n-th (central) harmonic frequency of Fb (nFb-mFc is obtained as $n \times F_b - m \times F_c$), nFb+mFc: the m-th positive sideband of n-th (central) harmonic frequency of Fb (obtained with the sum).

For inner race defects, the fault interacts with the shaft's rotational speed because load distribution changes during each rotation, causing amplitude modulation. In this modulation, the characteristic inner race fault frequency acts as the carrier, while the shaft rotational frequency serves as the modulating signal. Similarly, amplitude modulation occurs between the ball spin frequency (BSF) and the fundamental train frequency (FTF), with the BSF as the carrier. This arises as the ball moves in and out of the load zone during cage rotation. Figure 3 illustrates the first-order sidebands associated with these phenomena.

4 Spectral Fault Receptive Fields

For each one of the faults, we will construct fault detectors inspired by the primate retinal ganglion cell receptive fields. Receptive fields in the primate retina are specific regions of the visual field where the presence of a stimulus (such as light or its absence) can excite (or inhibit) the activity of a ganglion cell. Although the retina encodes visual information

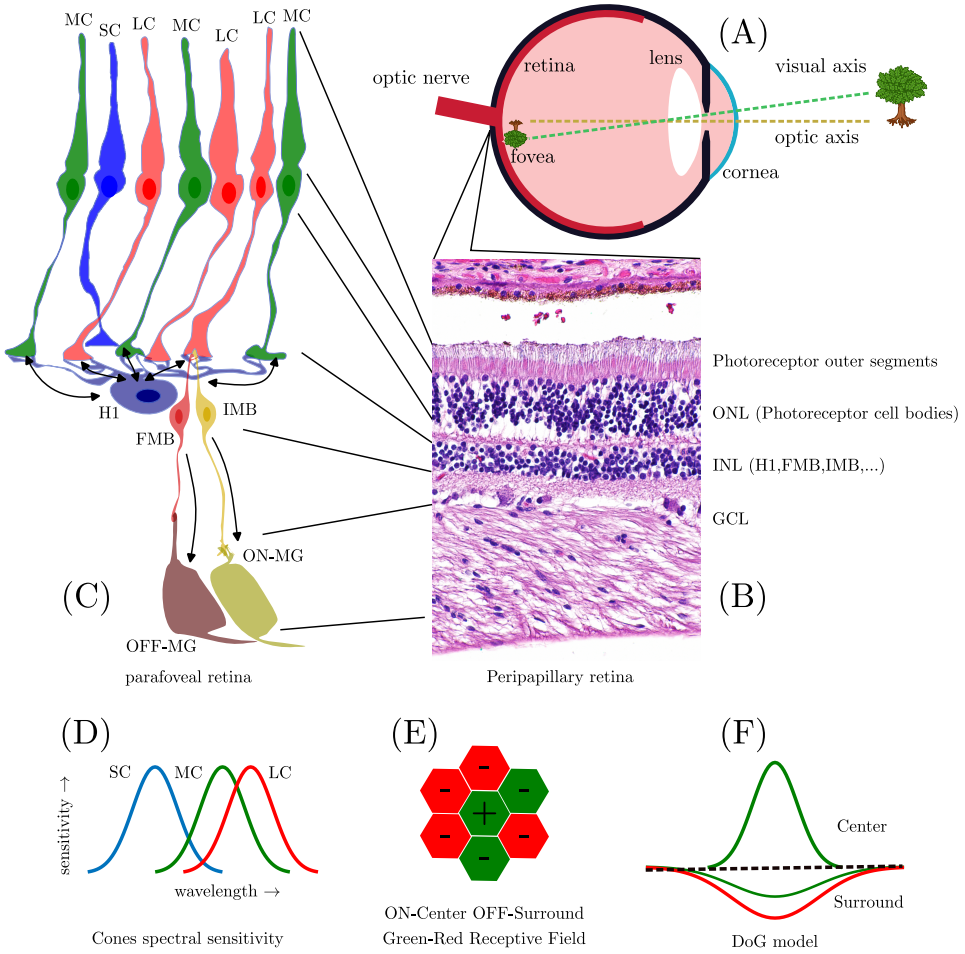


Figure 4 Encoding pathway underlying receptive field formation in primate retinal midget (P) ganglion cells. Insets: (A) eye anatomy highlighting retina, fovea, and image formation; (B) micrograph of peri-papillary retina layers (modified from [14]); (C) ON- and OFF-MG network in parafovea showing LC and MC cone synapses, H1 horizontal cells, flat and invaginating midget bipolar cells; (D) spectral sensitivity of cone types; (E) foveal cone mosaic depicting green-center red-surround receptive field; (F) DoG model illustrating center-surround color contrast.

through many parallel channels (processing chromatic, spatial, and temporal information), many image-forming retinal ganglion cells share a fundamental property: a center-surround spatial and chromatic organization. This type of processing encodes information about spatial and, for some cells, also chromatic contrast or differential excitation within the receptive field's spatial extent. Figure 4 illustrates the mechanisms of biological receptive fields inspiring our design of fault detectors that mimic center-surround contrast processing.

4.1 Frequency Masks

To define the fault detectors, we utilize the frequency bands described in Section 3.2. Inspired by the center-surround organization of ganglion cell receptive fields, we introduce two distinct *spectral extents*, that is, frequency bands, with a narrower band representing the center and a broader band representing the surround. We implement the receptive fields in the frequency domain. Operating in this domain allows us to use a Gaussian function as the gain profile, which we call a *spectral mask*; this can be interpreted as implicit bandpass filtering.

Table 2 Characteristic frequencies for bearing faults including harmonics and sidebands. Notation N_h: number of harmonics, and N_s: number of sidebands.

Fault Mode	Characteristic Frequencies
Outer race	$\mathcal{F}_O = \{nf_{\text{BPFO}} \mid n = 1..N_h\}$
Inner race	$\mathcal{F}_I = \{nf_{\text{BPFI}} + sf_r \mid n = 1..N_h,\ s = -N_s..N_s\}$
Ball	$\mathcal{F}_B = \{nf_{\text{BSF}} + sf_{\text{FTF}} \mid n = 1..N_h,\ s = -N_s..N_s\}$
Cage	$\mathcal{F}_C = \{nf_{\text{FTF}} \mid n = 1..N_h\}$

The set $\mathcal{M}$ of *admissible spectral masks* is defined as $\mathcal{M} = \left\{m : \mathbb{R}_0^+ \to [0,1]\right\}$. We restrict the domain to the non-negative real numbers because the spectral masks will act as a gain that will be multiplied by the absolute magnitude of the frequency components. Let $M \subset \mathcal{M}$ be a finite subset of masks. We define the disjunction over M as the pointwise maximum over magnitudes as:

$$\bigvee M \in \mathcal{M}, \quad \text{specifically,} \quad \bigvee M(f) = \max_{m \in M} m(f).$$

Given the frequency band $B = [f_{\min}, f_{\max}]$, and the parameter k_σ, the Gaussian frequency mask $G(f)$ is defined as:

$$G(f; B, k_\sigma) = \exp\left(-\frac{1}{2}\left(\frac{f - f_c(B)}{\sigma_f}\right)^2\right) \tag{1}$$

with $f_c(B) = \frac{f_{\min}+f_{\max}}{2}$, and $\sigma_f = \frac{f_{\max}-f_{\min}}{2 \cdot k_\sigma}$. The parameter k_σ, called the sigma rule, determines how the limits of the frequency bands are handled. A sigma rule of 3 corresponds to 99.7% of the area under the Gaussian falling within the specified frequency band.

4.2 SFRFs Computation

An advantage of filtering in the frequency domain is that it enables all operations to be computed simultaneously by precomputing a single gain mask across the spectrum for each operational mode. This strategy is particularly efficient because, although the characteristic frequencies of interest shift with the shaft speed, the frequency-domain filters can be generated in advance, and applying the filter is equivalent to a Hadamard product (elementwise multiplication) between the spectrum of the vibration signals and the mask corresponding to the appropriate operational mode.

The characteristic frequencies for each fault mode are defined as shown in Table 2. We define the corresponding *frequency bands* as $\mathcal{B}(F, W) = \{[f - \frac{W}{2}, f + \frac{W}{2}] \mid f \in F\}$. Given the set the characteristic frequencies $\mathcal{F} \in \{\mathcal{F}_O, \mathcal{F}_I, \mathcal{F}_B, \mathcal{F}_C\}$, and shape parameters $\sigma_C = (\mathcal{W}_\mathcal{C}, \kappa_C)$ for the center and $\sigma_S = (\mathcal{W}_\mathcal{S}, \kappa_S)$ for the surround. Then given $\sigma = (\mathcal{W}, \kappa) \in \{\sigma_C, \sigma_S\}$ we can define a *receptive field gain function* as:

$$\mathcal{G}_\mathcal{F}^\sigma \in \mathcal{M}, \quad \text{specifically,} \quad \mathcal{G}_\mathcal{F}^\sigma = \bigvee \{G(f; B, \kappa) : B \in \mathcal{B}(\mathcal{F}, \mathcal{W})\}.$$

We refer to $\mathcal{W}_\mathcal{C}$ as the center bandwidth and $\mathcal{W}_\mathcal{S}$ as the surround bandwidth.

The *Difference of Gaussians* used to compute the SFRF is then given by:

$$\text{DoG} = \int_0^{\frac{f_s}{2}} \left[\mathcal{G}_\mathcal{F}^{\sigma_C}(f) - \kappa_H \, \mathcal{G}_\mathcal{F}^{\sigma_S}(f)\right] |A(f)| \, df \tag{2}$$

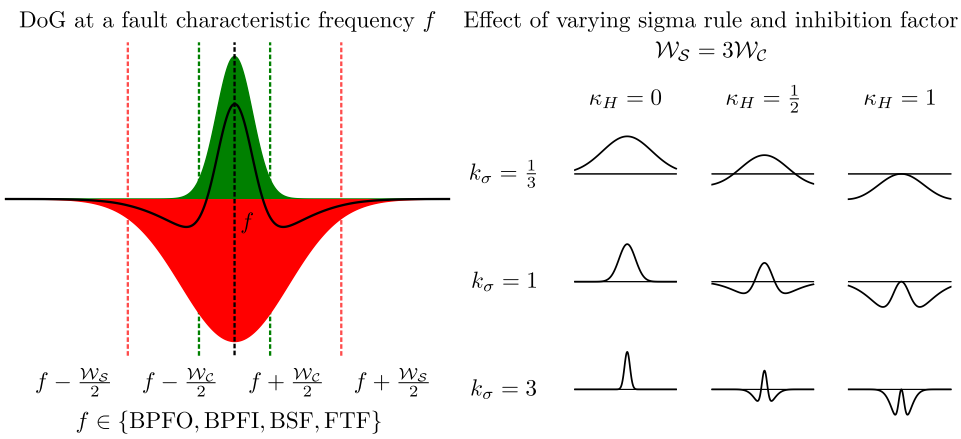


Figure 5 Parameters of the *Difference of Gaussians* (DoG) model. Left: Illustration of a DoG model centered at a fault characteristic frequency. For the receptive field gain functions $\mathcal{G}^{\sigma}_{\mathcal{F}}$, these are defined for the applicable harmonics and sidebands (see Tables 1 and 2). Right: Effect of varying the sigma rule k_σ ($k_\sigma = \kappa_C = \kappa_S$) and the inhibition factor κ_H under the condition $\mathcal{W}_S = 3\mathcal{W}_C$.

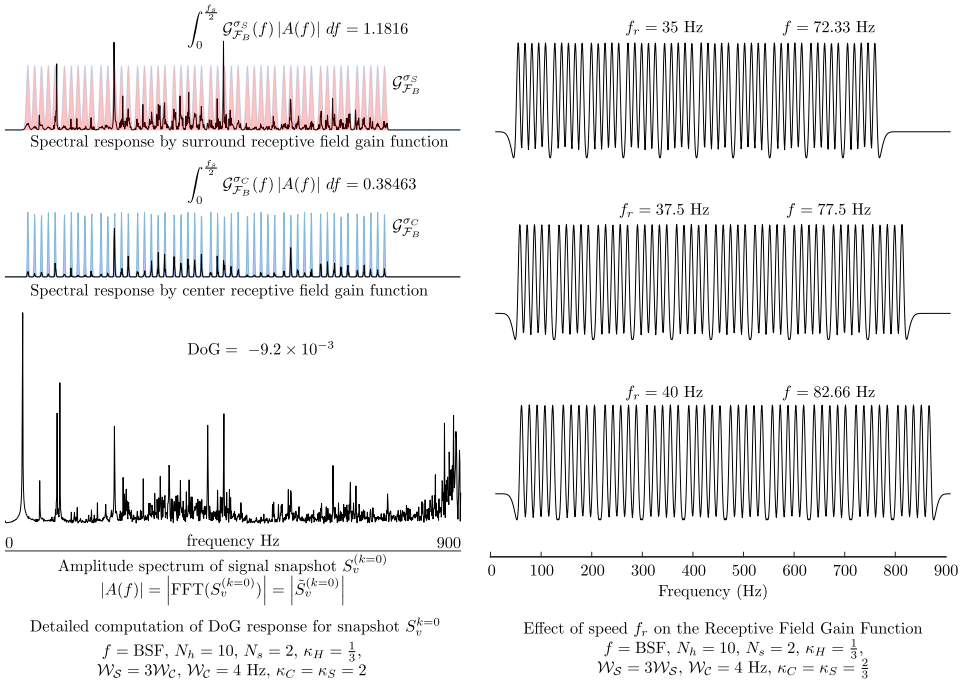


Figure 6 Computation of DoG for snapshot $S_v^{k=0}$ and fault type *Ball*, and effect of speed on receptive field gain function (RFGF). Left, bottom to top: unfiltered amplitude spectrum, spectrum filtered by center RFGF $\mathcal{G}^{\sigma_C}_{\mathcal{F}}$, and spectrum filtered by surround RFGF $\mathcal{G}^{\sigma_S}_{\mathcal{F}}$. Numeric integrals and final DoG computation included. Right: Effect of shaft speed f_r on DoG RFGF, $\mathcal{G}^{\sigma_C}_{\mathcal{F}} - \kappa_H \mathcal{G}^{\sigma_S}_{\mathcal{F}}$.

where f denotes frequency in the vibration spectrum, f_s the sampling frequency, and $A(f) = \tilde{S}^{(k)}(f)$ is the Fourier transform of the k^{th} accelerometer signal snapshot $S^{(k)}$. The parameters κ_C and κ_S control the width (sigma rule) of the center and surround Gaussians, respectively, and κ_H is the inhibition factor. Figure 5 illustrates these parameters and their effect in the shape of the gain profiles. Figure 6 illustrates the DoG computation and the adaptation of the receptive field gain functions with shaft speed. Figure 7 depicts the processing pipeline and detail SFRFs computation.

5 Experimental Evaluation

In this section, we evaluate the suitability of SFRFs for condition monitoring. Traditionally, condition monitoring relies heavily on identifying condition indicators and health indices that effectively capture the degradation trend of a system. This effectiveness is often assessed using metrics such as monotonicity, prognosability, and trendability. However, the XJTU-SY dataset contains only five samples per operational condition, which is insufficient for meaningful analysis of prognosability and trendability, since these metrics require larger datasets. Therefore, we selected *monotonicity* as the primary evaluation criterion. This choice is justified because, if a signal is to reliably capture degradation and we assume no regenerative processes, as is typical in the mechanical system under study, there must be a consistent correlation between the values of the condition indicators or health index and the operational time of the machine. However, it is important to note that expecting perfect monotonicity is unrealistic. Aleatoric uncertainties, unknown inputs, and varying environmental contexts naturally introduce fluctuations into the estimations. For this reason, we also consider the *smoothness* of the condition indicator as an additional evaluation metric.

5.1 Empirical Selection of Parameters

We report the qualitative behavior of condition indicators $C^{(k)}$ obtained using SFRFs for different fault types on the bearing labeled *Bearing1_1*. Since SFRF is a novel technique with many unknowns, we first present results based on empirical choices.

Inspired by the qualitative behavior of receptive fields of primate parvocellular ganglion cells, we chose a center contribution stronger and narrower than the surround. Two parameter sets control the frequency span: bandwidths $\mathcal{W}_C, \mathcal{W}_S$, and frequency attenuation sharpness given by sigma rules κ_C and κ_S. Bandwidth selection focused on limiting spatial overlap among fault bands while capturing natural frequency deviations near characteristic fault frequencies within the constraints of the dataset's maximum frequency resolution (0.78125 Hz). We adopt a center-to-surround bandwidth ratio of 1:3, with $\mathcal{W}_C = 4$ Hz and $\mathcal{W}_S = 12$ Hz. The sigma parameters are set as $\kappa_C = \kappa_S = 2$. For the DoG computation, $N_h = 10$ harmonics and $N_s = 2$ sidebands are used, with inhibition factor $\kappa_H = \frac{1}{3}$.

5.2 Evaluation of SFRF with Empirical Parameters

We computed the SFRFs response to horizontal and vertical acceleration and visualized the temporal behavior of the SFRFs to assess whether they can capture the degradation trend of the bearing. Figure 8 shows the temporal behavior of the four SFRFs. It can be observed that all SFRFs are capable of detecting a sudden transition in the temporal evolution. This behavior is reasonably interpreted as the manifestation of a defect, with the degradation trajectories for the inner and outer race SFRFs differing significantly around time 80.

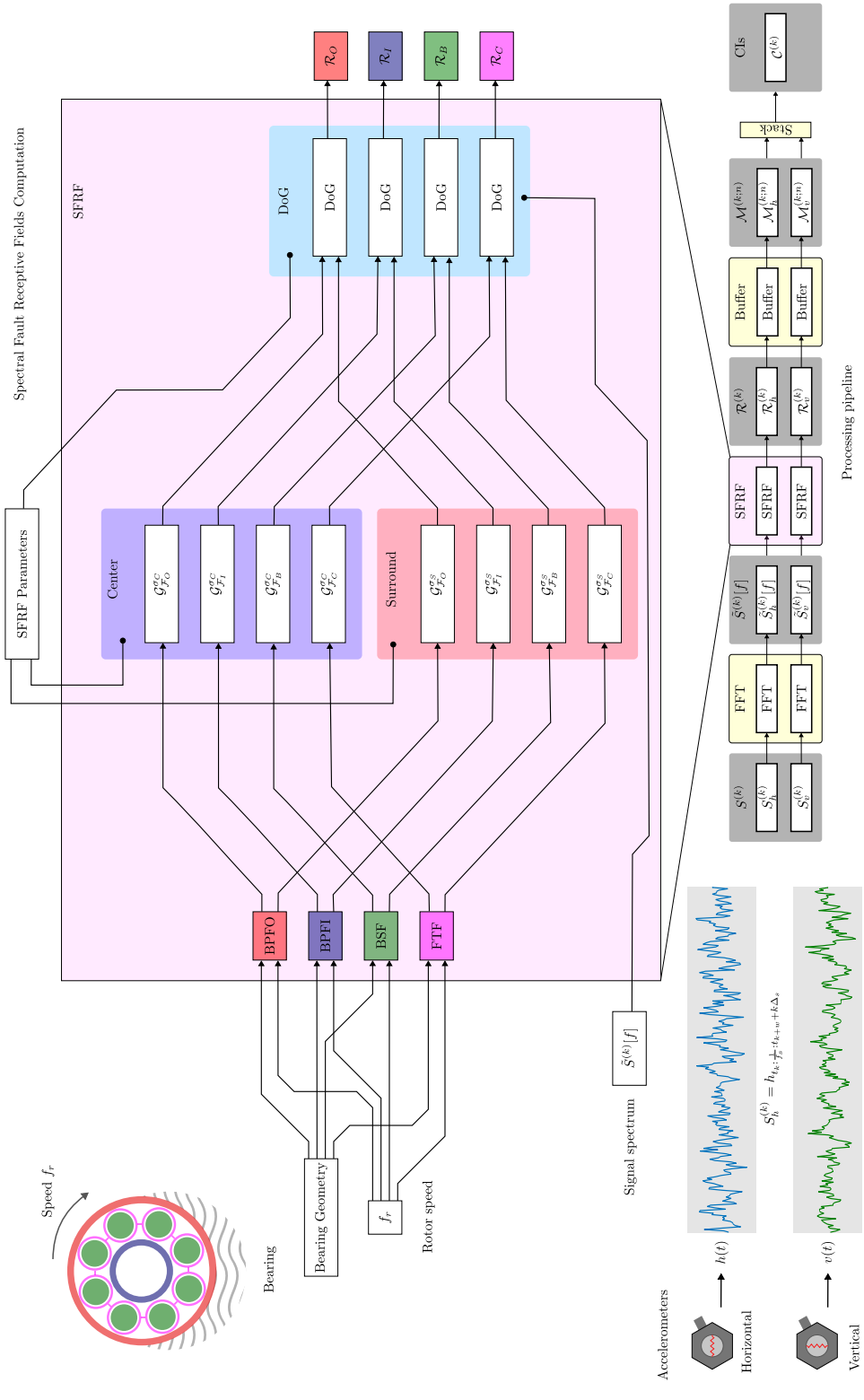


Figure 7 Computation of SFRFs and processing pipeline. Bottom left: vibration signals from two accelerometers; snapshots indexed by k. Bottom right: pipeline computes FFT and SFRFs per channel, stores computed vectors in a buffer, and stacks channels to form CIs. Top: exploded view of SFRF computation using shaft speed, vibration spectrum, bearing geometry, and SFRF parameters.

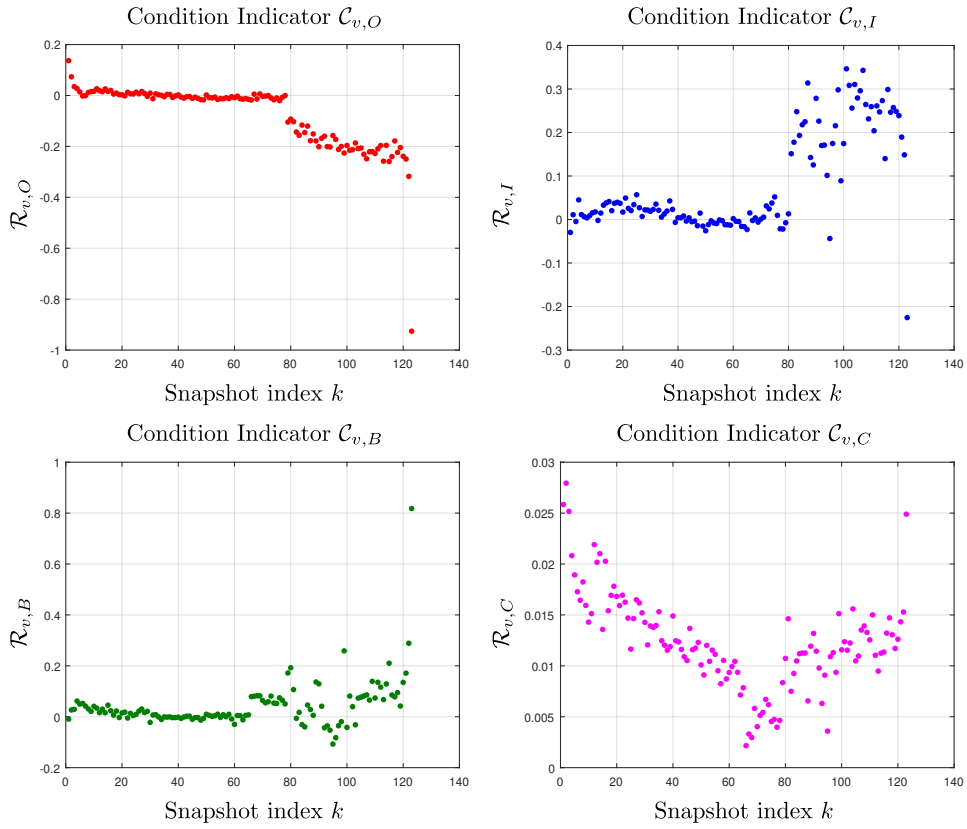


Figure 8 Temporal behavior of condition indicators for the four fault types and vertical vibrations of bearing *Bearing1_1*, under operating conditions of shaft speed $f_r = 35$ Hz and load 12 kN.

Another observation from the ball CI $\mathcal{C}_{v,B}$ in Figure 8 is its early response to an event, around time 65, before outer ($\mathcal{C}_{v,O}$) and inner race ($\mathcal{C}_{v,I}$) CIs exhibit any noticeable change. This behavior may reasonably be interpreted as indicating either the onset of a less severe defect or a precursor to the severe fault, detected at about time 80 by all CIs. Notably, this abrupt change in the degradation trajectory is also perceived by the cage CI ($\mathcal{C}_{v,C}$). The cage CI, in particular, captures the early degradation pattern effectively, displaying a consistent trend from the beginning up to the early event. This suggests that, even with a crude heuristic parameter selection, the combination of cage and ball CIs may offer a reliable monitoring of degradation since the very beginning of the operational life of the bearing.

5.3 Optimizing for Condition Monitoring and Prognosis

Our qualitative results are encouraging but also highlight several issues with the current method, the most notable being the varying sensitivity of different CIs to degradation events. Evolutionary multi-objective optimization techniques are particularly well-suited for scenarios where theoretical guidance is limited, as they require minimal assumptions and can efficiently explore complex parameter spaces. In this context, we formulated the exploration of the SFRFs parameter space as an optimization problem, explicitly quantifying our condition monitoring and prognosis criteria and defining them as objectives to be optimized. Table 3 presents these three objectives. The first objective directly assesses the model's predictive accuracy by quantifying the error in remaining useful life (RUL) estimation. The second objective encourages consistent sensitivity to degradation across the component's lifetime.

Table 3 Optimization objectives for NSGA-II. Notation: CIs are concatenated into a time-varying condition indicator vector $x^{(t)}$ (subsampled from $\mathcal{C}^{(k)}$) of dimension F (4 CIs from $\mathcal{C}_h^{(k)}$ and 4 CIs from $\mathcal{C}_v^{(k)}$), ρ_j is the Spearman correlation between feature j and snapshot time, and $\Delta x_j^{(t)}$ is the first difference of the j-th SFRFs at time t, $y^{(t)}$ is the observed RUL at time t while $\hat{y}^{(t)}$ the predicted RUL, both quantities are normalized by the maximum RUL.

Objective	Equation
RUL Error (MSE)	$\frac{1}{K}\sum_{i=1}^{K}(y^{(t)}-\hat{y}^{(t)})^2$
Monotonicity	$\left(\prod_{j=1}^{F}\lvert\rho_j\rvert\right)^{1/F}$
Smoothness (MAD)	$\left(\prod_{j=1}^{F}\underset{t}{\text{median}}\left(\left\lvert\Delta x_j^{(t)}-\underset{t'}{\text{median}}\left(\Delta x_j^{(t')}\right)\right\rvert\right)\right)^{1/F}$

The third objective penalizes jitter along the degradation trajectory, thereby promoting smoother and more interpretable trends. We chose to use the geometric mean rather than conventional averaging to ensure that no individual CI is overlooked during the optimization process. Although it is theoretically possible to avoid aggregating the behaviors altogether and instead treat each CI as an independent objective, this alternative was not pursued in the present study. In retrospect, formulating the optimization of different SFRFs as independent optimization problems may represent a better path, since their computations do not depend on one another. We leave this possibility open for exploration in future work.

A key advantage of surrogate models lies in the efficient computation of objectives, which is essential given the population-based nature of evolutionary algorithms, where a single run may involve thousands of evaluations. To predict the RUL, we trained a bagging regression ensemble model on a sub-sampled degradation trajectory of Bearing1_1. This strategy offers several benefits: (1) bagging regressors, as ensemble methods, provide robust predictions with a reduced risk of overfitting; (2) they perform well even with limited data; and (3) they are computationally efficient, making them well-suited as surrogate models during optimization.

However, this decision also introduces a notable challenge: bagging regressors are inherently nondeterministic, which can lead to fluctuations in the Pareto front during optimization. We consider this acceptable, even somewhat beneficial, as the stochasticity helps mitigate overfitting and discourages convergence toward non-robust regions of the parameter space, which is particularly important when working with limited data.

We performed the optimization of SFRFs parameters with MATLAB's *gamultiobj* function using the algorithm's default settings and the following domain bounds for the parameters: $\kappa_C, \kappa_S \in \left[\, n^{-2}, n^2\right]$ $\kappa_H \in [0,1]$ with $n = 3$. Figure 9 presents the local Pareto-optimal front after 155 iterations, at which point the convergence criterion was satisfied (i.e., the change in the spread of Pareto solutions is less than 1×10^{-4}). NSGA-II identifies a diverse set of non-dominated solutions. As the optimization function performs minimization in all objectives, we changed the sign of monotonicity and smoothness to enforce their maximization.

Several findings are noteworthy. First, all Pareto-optimal individuals cluster within a region where the sigma rule center spreads up to a maximum of 6, while the sigma rule surround spreads up to a maximum of 2. This aligns with our expectation that the surround performs better when covering a wider frequency bandwidth (the higher the sigma rule, the stricter the Gaussian). However, it is somewhat surprising that this was not already enforced by setting the surround bandwidth $\mathcal{W}_S$ to be three times that of the center $\mathcal{W}_C$.

We observe that most objectives conflict with each other. The algorithm's selection mechanism, which emphasizes boundary individuals through its use of crowding distance, tends to favor solutions at the extremes of the objective space in order to maximize diversity across the Pareto front. Despite leveraging interactive visualizations, we did not identify the

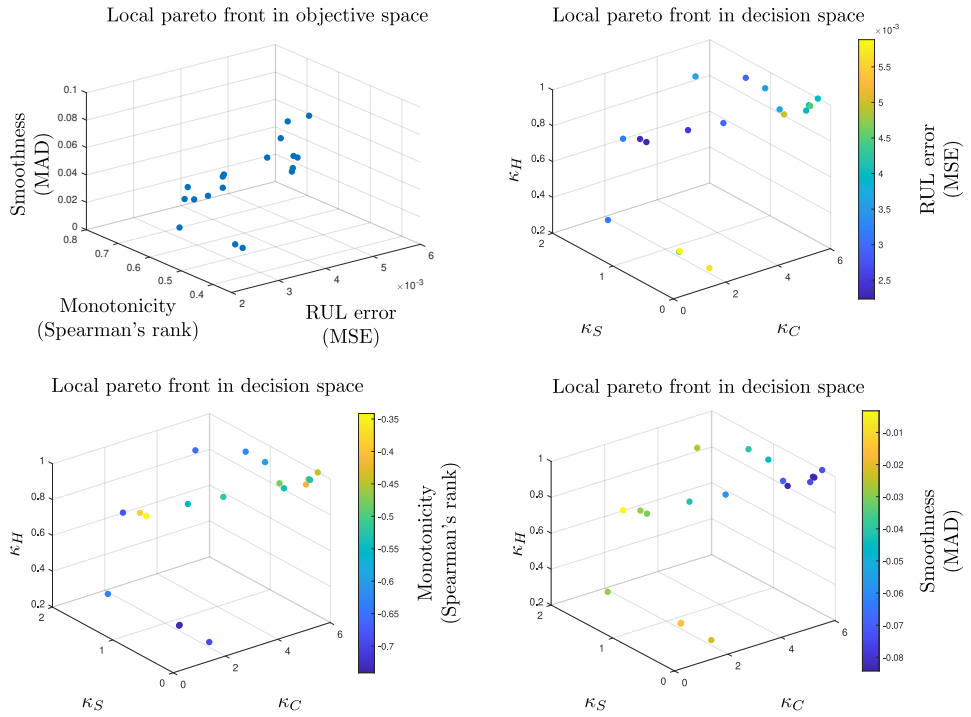


Figure 9 The local Pareto-optimal front identified by NSGA-II after 155 iterations and 7,750 function evaluations. Top left panel visualizes individuals in the objective space. The remaining panels depict the parameter space, with solution colors encoding different objectives (in the color bars, the sign of monotonicity and smoothness is reversed, deep blue colored solutions are better).

anticipated cooperation between monotonicity and RUL prediction accuracy. Specifically, solutions exhibiting high monotonicity (deep blue in the monotonicity inset) often perform poorly in terms of RUL prediction, and vice versa, those with low RUL error (deep blue in the RUL error domain) tend to score low in monotonicity.

This limitation is particularly evident in the case of our cage CI, which, despite being highly informative of degradation throughout the entire operational life, would be penalized by conventional monotonicity metrics. Its triangular shape, coupled with relatively low energy content and a noisy appearance, would result in a low monotonicity score. However, the fundamental issue extends beyond this specific example and lies in the distinction between local and global trends, as well as in the methodology used to compute monotonicity. Traditional condition indicators are computed episodically rather than as states of dynamical processes. In simple degradation models, the health condition is typically evaluated based on a single temporal snapshot, without consideration for the underlying trend or the temporal evolution of the indicator. This highlights the need for monotonicity metrics or health indicators that account for temporal dynamics on multiple temporal scales.

These findings suggest that CIs should be evaluated as dynamic processes, not merely as isolated episodes. This supports the adoption of stochastic differential equations as robust models for degradation processes. Moving forward, our future research should focus on stochastic model identification and the tracking of their parameters, which may provide a more nuanced and accurate assessment of system health and more rational RUL estimations.

We repeated our qualitative evaluation of the CIs, this time selecting the individual from the Pareto front that achieved the best RUL prediction performance. This individual is characterized by the following parameters: $\kappa_C = 1.0253$, $\kappa_S = 0.8905$, and $\kappa_H = 0.8647$.

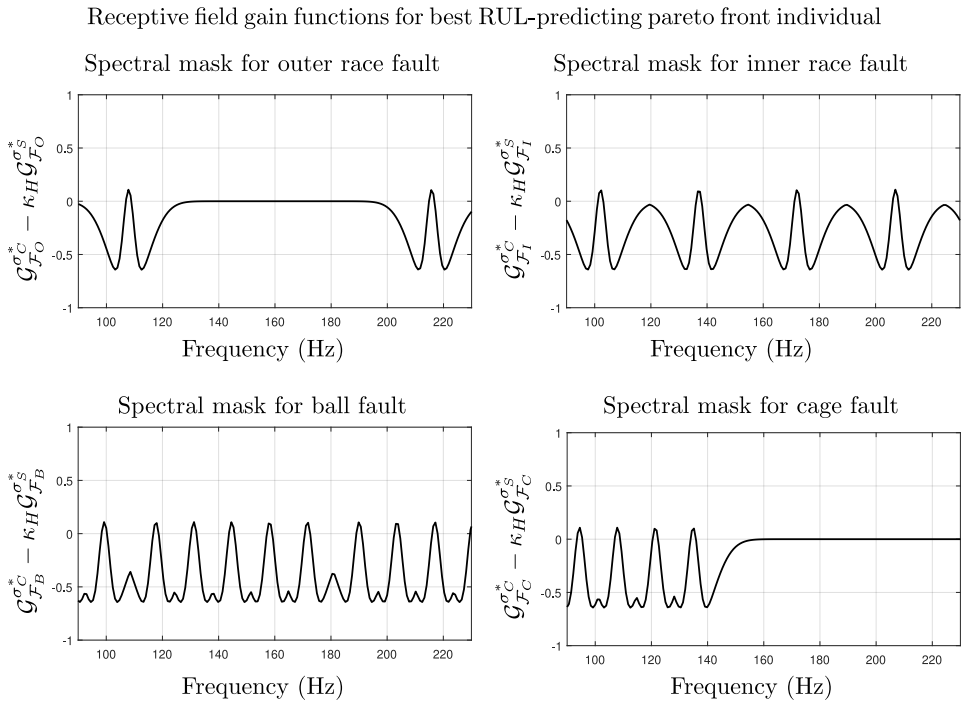


Figure 10 DoG RFGFs for the best RUL-predicting individual on the local Pareto-optimal front. Operating conditons: shaft speed $f_r = 35$ Hz and load 12 kN. Only the 90-230 Hz band is shown.

Figure 10 illustrates the RFGFs corresponding to this optimal parameter set within the local Pareto-optimal front, as determined by the lowest RUL prediction error. The receptive fields exhibit an excitatory center but primarily operate within the inhibitory region. Notably, a cumulative effect, kept in check by the max operator, emerges when the Gaussian surrounds overlap, particularly for the ball and cage CIs (see bottom insets). This overlap causes the receptive fields to compute spectral contrast across the local spectrum. As we established broad parameter ranges for the genetic algorithm to explore candidate solutions, including configurations with weak or no inhibition, the observed convergence towards solutions with strong and wide inhibitory surrounds indicates a positive relationship between spectral contrast and the criteria for condition monitoring and prognosis. If traditional approaches relying solely on filtering characteristic frequencies were superior, the genetic algorithm would have favored those simpler solutions. Instead, it evolved more complex inhibitory surrounds essential for effective contrast computation, highlighting the importance of antagonistic spectral filtering in enhancing degradation state assessment.

Regarding the utility of SFRFs as CIs, Figure 11 presents a comparison between the empirically obtained CIs analyzed in the previous section and those corresponding to the most performant RUL prediction solution. The evolved CIs better characterize degradation trends and more clearly signal the onset of defects compared to their empirical counterparts. Notably, the ability of the cage CI to correlate with degradation from the beginning of the bearing's operational life is further enhanced by the evolutionary algorithm.

To test our hypothesis that RUL predictions should account for the temporal evolution of condition indicators, we conducted experiments in which we varied the order of the condition indicators used for prediction. Following standard dynamical systems terminology, we refer to the *zero-order* indicator as the instantaneous condition indicator (although it is

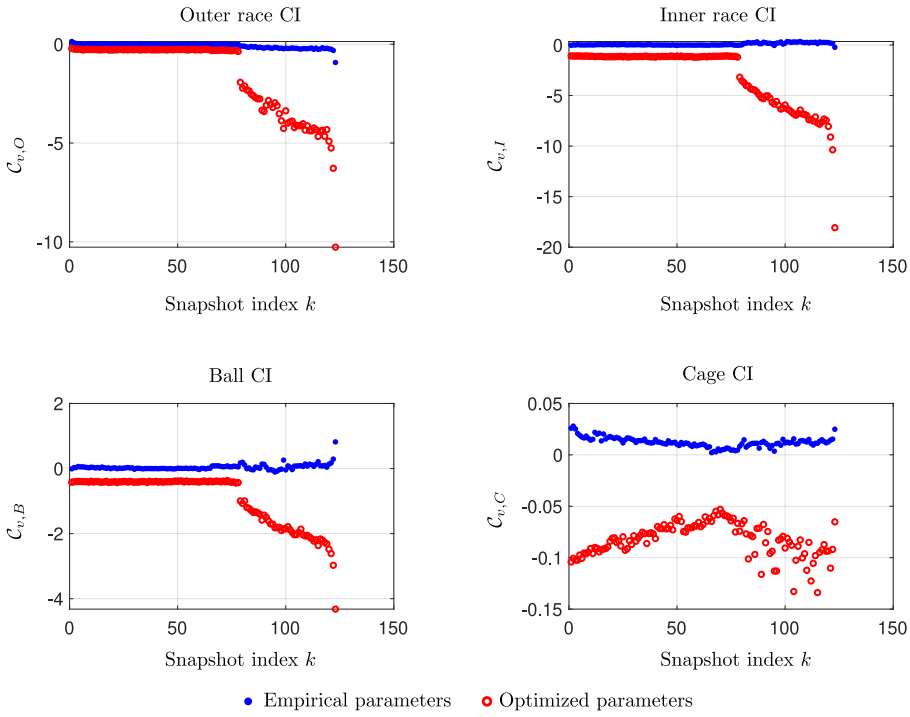


Figure 11 Comparison of CIs obtained with empirical parameters (blue filled circles) versus the best RUL-predicting local Pareto-optimal solution (open red circles).

computed from a signal snapshot of 1.28 seconds) represented by the eight CIs (four fault types across two vibration signals). The *first-order* indicator is a 16-dimensional vector formed by concatenating the current CIs with those from the previous time step. More generally, the *n-th order* CI corresponds to an $8(n+1)$-dimensional vector comprising the current CI and a buffer containing the previous n sets of CIs. This formulation allows the model to incorporate temporal context and memory into the RUL prediction process.

Figure 12 illustrates the effect of varying the SFRF order on prediction performance. The left inset shows the resubstitution loss of bagging regressor models trained with different orders. Due to the model's nondeterministic nature, we repeated the training 30 times and used box plots to represent the distribution of errors for each order. We observe that using a second-order SFRF condition indicator vector can reduce the resubstitution loss by approximately half. The right-hand visualization demonstrates the impact of SFRF order on RUL prediction accuracy; notably, the 10th-order predictor exhibits a marked improvement, closely tracking the true RUL across the entire operational life.

While these results are encouraging, they reflect only the *training* loss and must be substantiated through rigorous cross-validation methodologies. Nevertheless, the findings underscore the potential value of incorporating temporal memory into RUL estimation.

6 Discussion

Drawing inspiration from the biology of vision, we have explored the implementation of SFRFs based on the characteristic frequencies of bearing elements, their harmonics, and known amplitude modulation phenomena. This leads to implicit spectral filters that adapt naturally to varying speed conditions (see Figure 6). SFRFs compute spectral activity contrast in a

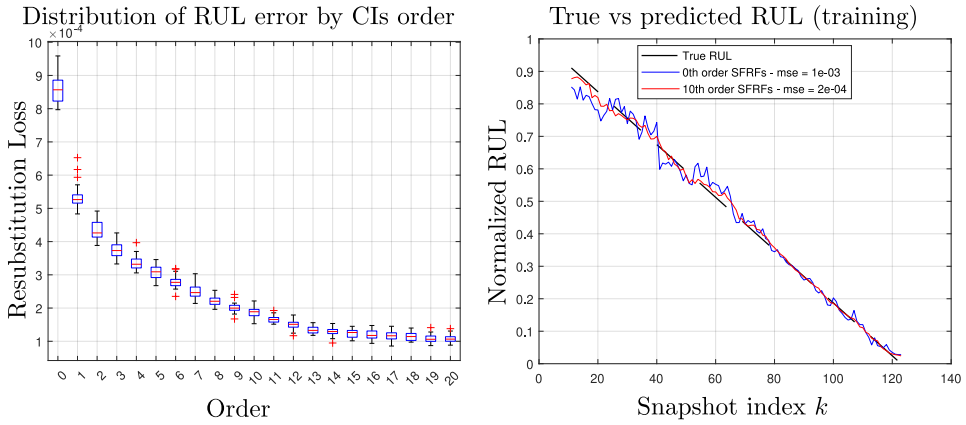


Figure 12 Training (resubstitution) MSE error with increasing orders and comparison of RUL estimations between 0th order and 10th order SFRFs.

manner analogous to how visual systems encode chromatic information. However, instead of excitation by light and preferential responses to different wavelengths by photopigments in cone photoreceptors, we analyze vibration signals transformed into the frequency domain. By monitoring the characteristic frequencies associated with different bearing components, SFRFs enable us to track degradation trends throughout the operational life of the machinery. Our particular implementation relies on the computation of Gaussian spectral filters centered at the characteristic frequencies, numerical integration of frequency bins across the spectrum, and the evaluation of spectral contrast, which depends on integrated energy within a narrow bandwidth we call the center, and a wider bandwidth, the surround. The DoG model, adapted to our domain, was formally defined and implemented. Its definition aims for computational efficiency. Our qualitative evaluation of the SFRFs for monitoring health state demonstrates their potential as effective condition indicators. The trends observed throughout the operational history indicate that SFRFs can detect abrupt defect events. Furthermore, they appear capable of capturing the gradual evolution of degradation. In addition, these findings support using SFRFs for fault detection and potentially diagnosis, as their outputs correspond to specific failure modes. However, a decision layer is essential to harmonize the degradation trends from different CIs. This integration is reserved for future research, as it requires further development and validation. Alternatively, the multi-objective algorithm could explicitly optimize diagnostic performance if suitable datasets are available.

Different SFRFs exhibit unique behaviors and require appropriate parameter tuning to maximize their effectiveness. Building on these insights, we established quantitative criteria to assess the suitability of various parameters in the Difference of Gaussians (DoG) model, ensuring the extraction of features that are relevant for condition monitoring and prognosis. We use a multiobjective genetic algorithm to compute an approximate Pareto-optimal set of solutions. To guide the optimization, we incorporated three objectives: first, the RUL prediction error, evaluated using fast surrogate RUL estimators instantiated as bagging regressor models; second, monotonicity, which is widely recognized in the PHM community as a crucial metric for feature selection in prognostic pipelines; and third, smoothness, which targets the desirable property of stability in condition monitoring indicators. We contrasted the quality of the best predictor against the empirical counterpart, demonstrating the value of the optimization stage. We also observed, through analysis of the cage CIs, that certain feature indicators may provide valuable information for condition monitoring and prognosis,

yet may be overlooked or rejected when evaluated solely by standard monotonicity metrics. This highlights the need for the development of more sophisticated evaluation criteria that can capture the full prognostic value of such features. Motivated by this observation, we investigated the impact of incorporating local temporal trends by stacking the CIs in a memory buffer. We assessed RUL predictions across different orders, and our results confirm that incorporating temporal context significantly reduces prediction error.

We acknowledge the preliminary nature of our contribution. Notably, the distinct spectrotemporal properties exhibited by different SFRFs suggest that their parameters should be tuned independently, as a one-size-fits-all approach may be overly simplistic. Drawing further inspiration from biology, it is well established that retinal ganglion cells operate in parallel channels, each capturing diverse spectro-spatio-temporal properties and contributing to a robust and flexible representation. Similarly, future research could explore the simultaneous deployment of a diversity of solutions along the Pareto front through ensemble techniques. By orchestrating and interpreting the responses of multiple SFRFs, it may be possible to achieve more comprehensive and adaptable representations, where individual SFRFs provide complementary, partial views tailored to specific objectives. For instance, some SFRFs may be better suited for condition monitoring, others for RUL prediction, and their relative importance or activity could adapt dynamically depending on the degradation state. Currently, FFT computation is the most resource-intensive step in the pipeline. However, next-generation sensors could be designed to shift the focus from general-purpose accelerometers to resonant arrays that respond preferentially to the engine's spectral fingerprint, potentially eliminating the need for FFT altogether.

7 Conclusions

This study demonstrates the value of drawing inspiration from nature to develop robust and reliable systems. Spectral fault receptive fields show considerable promise as foundational elements for condition monitoring and prognosis. They also have a minimal computational footprint, making them well-suited for onboard deployment in EVPs. Our qualitative evaluation indicates that, particularly in their optimized form, SFRFs are well-suited for both condition monitoring and remaining useful life (RUL) estimation.

We conclude that by integrating established vibrational analysis techniques with conceptual models from biological perception, and with the help of evolutionary algorithms, it is possible to devise effective solutions for tracking degradation states throughout the operational life of bearings. These types of biologically inspired solutions open new possibilities for advancing predictive maintenance and enhancing the reliability of industrial machinery.

References

1 Peter M. Asaro. From Mechanisms of Adaptation to Intelligence Amplifiers: The Philosophy of W. Ross Ashby. In Philip Husbands, Michael Wheeler, and Owen Holland, editors, *The Mechanical Mind in History*, pages 153–176. MIT Press, Cambridge, MA, 2008. doi:10.7551/mitpress/7626.003.0008.

2 Wahyu Caesarendra, Buyung Kosasih, Kiet Tieu, and Craig A.S. Moodie. An application of nonlinear feature extraction - A case study for low speed slewing bearing condition monitoring and prognosis. In *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1713–1718, 2013. doi:10.1109/AIM.2013.6584344.

3 Xiao-Dan Chen, Ke Li, Shao-Fan Wang, and Hao-Bo Liu. Switching Unscented Kalman Filters With Unknown Transition Probabilities for Remaining Useful Life Prediction of Bearings. *IEEE Sensors Journal*, 24(20):32577–32595, 2024. doi:10.1109/JSEN.2024.3445934.

4 Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi:10.1109/4235.996017.

5 Department of Defense, Washington, DC. *MIL-HDBK-217F: Military Handbook – Reliability Prediction of Electronic Equipment*, 1991. Superseding MIL-HDBK-217E, Notice 1, 2 January 1990.

6 S.J. Engel, B.J. Gilmartin, K. Bongort, and A. Hess. Prognostics, the real issues involved with predicting life remaining. In *2000 IEEE Aerospace Conference. Proceedings (Cat. No.00TH8484)*, volume 6, pages 457–469 vol.6, 2000. doi:10.1109/AERO.2000.877920.

7 Mônica S Felix, John J Martinez, and Christophe Bérenguer. Remaining Useful Life (RUL) Control of Controlled Systems under Degradation. *Authorea Preprints*, 2024. Submitted to International Journal of Robust and Nonlinear Control. doi:10.22541/au.172146328.87517875/v1.

8 Ying Fu, Ye Kwon Huh, and Kaibo Liu. Degradation Modeling and Prognostic Analysis Under Unknown Failure Modes. *IEEE Transactions on Automation Science and Engineering*, 22:11012–11025, 2025. doi:10.1109/TASE.2025.3530845.

9 Gill Sensors & Controls. WearDetect explained (5 minute video), 2024. Accessed: 2025-06-10. URL: https://gillsc.com/weardetect-explained-5-minute-video/.

10 Einar Løvli Hidle. Early Detection of Subsurface Cracks in Rolling Element Bearings using the Acoustic Emission Time Series. Master's thesis, Norwegian University of Science and Technology (NTNU), 2021. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2826382.

11 Jinqiu Hu, Laibin Zhang, Wei Liang, and Zhaohui Wang. Incipient mechanical fault detection based on multifractal and MTS methods. *Petroleum Science*, 6(2):208–216, 2009. doi:10.1007/s12182-009-0034-8.

12 International Organization for Standardization. ISO 20816-3:2022. Mechanical vibration – Measurement and evaluation of machine vibration – Part 3: Industrial machines with nominal power above 15 kW and nominal speeds between 120 r/min and 30 000 r/min, 2022.

13 Yaguo Lei, Naipeng Li, Liang Guo, Ningbo Li, Tao Yan, and Jing Lin. Machinery health prognostics: A systematic review from data acquisition to RUL prediction. *Mechanical Systems and Signal Processing*, 104:799–834, 2018. doi:10.1016/j.ymssp.2017.11.016.

14 Librepath. Micrograph showing optic nerve head and retina. H&E stain. https://commons.wikimedia.org/wiki/File:Retina_--_high_mag.jpg, 2015. Creative Commons Attribution-Share Alike 3.0 Unported license (CC BY-SA 3.0).

15 MathWorks. *Predictive Maintenance Toolbox User's Guide R2024b*. MathWorks, 2018. URL: www.mathworks.com.

16 MathWorks. *Predictive Maintenance Toolbox Getting Started Guide*. MathWorks, 2024. Release R2024b, originally published 2018. URL: https://www.mathworks.com/help/predmaint/index.html.

17 Humberto R. Maturana and Francisco J. Varela. *De máquinas y seres vivos: Autopoiesis: La organización de lo vivo*. Editorial Universitaria. LUMEN., Santiago, Chile, 1972.

18 James G. McLeish. Enhancing MIL-HDBK-217 reliability predictions with physics of failure methods. In *2010 Proceedings - Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–6, 2010. doi:10.1109/RAMS.2010.5448044.

19 J. W. McPherson. *Reliability Physics and Engineering: Time-To-Failure Modeling*. Springer, Cham, 3rd edition, 2019.

20 Kamal Medjaher, Diego Alejandro Tobon-Mejia, and Noureddine Zerhouni. Remaining Useful Life Estimation of Critical Components With Application to Bearings. *IEEE Transactions on Reliability*, 61(2):292–302, 2012. doi:10.1109/TR.2012.2194175.

21 Jiadong Meng, Changfeng Yan, Guangyi Chen, Yaofeng Liu, and Lixiao Wu. Health Indicator of Bearing Constructed by rms-CUMSUM and GRRMD-CUMSUM With Multifeatures of Envelope Spectrum. *IEEE Transactions on Instrumentation and Measurement*, 70:1–16, 2021. doi:10.1109/TIM.2021.3054000.

22 Alvaro Moreno and Matteo Mossio. *Biological Autonomy: A Philosophical and Theoretical Enquiry*. Springer, Dordrecht, 2015. doi:10.1007/978-94-017-9837-2.

23 Ali Moshrefzadeh and Alessandro Fasana. The Autogram: An effective approach for selecting the optimal demodulation band in rolling element bearings diagnosis. *Mechanical Systems and Signal Processing*, 105:294–318, 2018. doi:10.1016/j.ymssp.2017.12.009.

24 Stan Muñoz Gutiérrez and Franz Wotawa. A qualitative study on the applicability and optimization of spectral fault receptive fields for condition monitoring and prognosis, June 2025. Zenodo. doi:10.5281/zenodo.15660819.

25 Danh Ngoc Nguyen, Laurence Dieulle, and Antoine Grall. Remaining Useful Lifetime Prognosis of Controlled Systems: A Case of Stochastically Deteriorating Actuator. *Mathematical Problems in Engineering*, 2015:1–16, 2015. doi:10.1155/2015/356916.

26 You-Jin Park, Shu-Kai S. Fan, and Chia-Yu Hsu. A Review on Fault Detection and Process Diagnostics in Industrial Processes. *Processes*, 8(9), 2020. doi:10.3390/pr8091123.

27 Aisong Qin, Qinghua Zhang, Qin Hu, Guoxi Sun, Jun He, and Shuiquan Lin. Remaining Useful Life Prediction for Rotating Machinery Based on Optimal Degradation Indicator. *Shock and Vibration*, 2017:Article ID 6754968, 12 pages, 2017. doi:10.1155/2017/6754968.

28 Yi Qin, Jiahong Yang, Jianghong Zhou, Huayan Pu, and Yongfang Mao. A new supervised multi-head self-attention autoencoder for health indicator construction and similarity-based machinery RUL prediction. *Advanced Engineering Informatics*, 56:101973, 2023. doi:10.1016/j.aei.2023.101973.

29 Robert B Randall. State of the art in monitoring rotating machinery-part 1. *Sound and vibration*, 38(3):14–21, 2004.

30 Paul D Samuel and Darryll J Pines. A review of vibration-based techniques for helicopter transmission diagnostics. *Journal of Sound and Vibration*, 282(1):475–508, 2005. doi:10.1016/j.jsv.2004.02.058.

31 Abhinav Saxena, Jose Celaya, Edward Balaban, Kai Goebel, Bhaskar Saha, Sankalita Saha, and Mark Schwabacher. Metrics for evaluating performance of prognostic techniques. In *2008 International Conference on Prognostics and Health Management*, pages 1–17, 2008. doi:10.1109/PHM.2008.4711436.

32 D. F. Shi, W. J. Wang, and L. S. Qu. Defect Detection for Bearings Using Envelope Spectra of Wavelet Transform . *Journal of Vibration and Acoustics*, 126(4):567–573, December 2004. doi:10.1115/1.1804995.

33 Manula A. Somaratna and Alan W. Freeman. The receptive field construction of midget ganglion cells in primate retina. *Journal of Neurophysiology*, 133(1):268–285, 2025. doi:10.1152/jn.00302.2024.

34 Moncef Soualhi, Khanh T.P. Nguyen, Kamal Medjaher, Fatiha Nejjari, Vicenc Puig, Joaquim Blesa, Joseba Quevedo, and Francesc Marlasca. Dealing with prognostics uncertainties: Combination of direct and recursive remaining useful life estimations. *Computers in Industry*, 144:103766, 2023. doi:10.1016/j.compind.2022.103766.

35 George Vachtsevanos, Frank Lewis, Michael Roemer, Andrew Hess, and Biqing Wu. *Intelligent Fault Diagnosis and Prognosis for Engineering Systems*. Wiley, Hoboken, NJ, USA, 2006.

36 Biao Wang, Yaguo Lei, Naipeng Li, and Ningbo Li. A Hybrid Prognostics Approach for Estimating Remaining Useful Life of Rolling Element Bearings. *IEEE Transactions on Reliability*, 69(1):401–412, 2020. doi:10.1109/TR.2018.2882682.

37 Weizhong Yan, Hai Qiu, and Naresh Iyer. Feature Extraction for Bearing Prognostics and Health Management (PHM)–A Survey. Technical report, Air Force Research Laboratory (AFRL/RXLMN), 2008.

Automating Control System Design: Using Language Models for Expert Knowledge in Decentralized Controller Auto-Tuning

Marlon J. Ares-Milian
School of Computer Science, University College Cork, Ireland

Gregory Provan
School of Computer Science, University College Cork, Ireland

Marcos Quinones-Grueiro
Vanderbilt University, Nashville, TN, USA

Abstract

Fully-automated optimal controller design for engineering systems is a challenging task. While, optimization-based, automated control parameter tuning techniques have been widely discussed in the literature, most works do not discuss expert knowledge requirements for system design, which result in significant human intervention. In this work, we discuss a multistage controller tuning framework for decentralized control that highlights expert knowledge requirements in automated controller design. We propose a methodology to automate the input-output pairing and stage definition steps in the framework using Large Language Models (LLMs) for a family of multi-tank benchmarks. We achieve this by proposing a mathematical language to describe the system and design an algorithm to bind this mathematical representation to the input prompt space of an LLM. We demonstrate that our methodology can produce consistent expert knowledge outputs from the LLM with over 97% accuracy for the multi-tank benchmarks. We also empirically show that, correct stage definition by the LLM can improve tuned controller performance by up to 52%.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning

Keywords and phrases controller auto-tuning, automated system design, large language models

Digital Object Identifier 10.4230/OASIcs.DX.2025.10

Supplementary Material *Software (Source Code)*: `https://github.com/mjares/DX2025_LLMs_ExpertKnowledge.git`, archived at `swh:1:dir:81a0af39fd36f5d0fe1daae775b1e460067fd364`

Funding This work was supported by Science Foundation Ireland under Grant 13/RC/2094.

1 Introduction

Most modern engineering issues are framed as systems. For example, vehicles, robots, and industrial machines are all systems, albeit different, and each have sub-systems and components that work in close relation to achieve their intended functions. The development of modern engineering systems goes through three main stages: design, production, and operation. Proper design is the focus of this paper, since it enables engineering systems to meet desired performance objectives, e.g.: development and operational costs, safe operation, and robustness, while also supporting validation and verification, enhancing the production and operation stages [8]. However, optimal system design is a very challenging task due to the extensive space of system configurations, components, and parameters [18] that must be explored. In spite of all the advances in automation to date, solving the design problem often requires significant human expertise and iterative design protocols [7]. Therefore, a strong motivation exists to develop automated optimal design methodologies that meet desired objectives while reducing the tedious, time-consuming, and costly aspects of manual design.

With the development of cyber-physical systems and the internet of things, automation has become ubiquitous in modern engineering systems [8] and the key component for automation in any modern system is the controller. A controller (or control algorithm) refers to the sub-system that receives information from real or estimated measurements and operates the actuators in order to regulate system variables with different objectives. The design of controllers for engineering is a complex task, consisting of multiple components and steps. Controllers are usually defined by parametric functions, which results in a problem well-known as *controller tuning*: the design step where controller parameters are adjusted (or tuned) to meet desired performance objectives. Traditional controller tuning is done manually, requiring an iterative design process based on expert knowledge. This is an inefficient approach that usually results in sub-optimal solutions, which has motivated significant research on automated design of controller parameters, known as controller auto-tuning.

Many modern controller auto-tuning solutions resort to an iterative optimization approach in which some performance objectives are evaluated (either in simulation or hardware) and the controller parameter space is explored using some optimization algorithm until the *best* set of parameters is obtained; this is done through some combination of system-model knowledge, expert knowledge, and historical data, and includes tools like reinforcement learning [13], evolutionary optimization [16], and Bayesian optimization [17]. Even though the controller parameter search itself has been widely automated, the controller design problem (including controller parameter design) still requires abundant expert knowledge in different steps of the process that are often ignored in the associated literature [2]. Some of the open challenges in fully automated controller design include aspects of the parameter optimization problem definition like defining constraints, defining and restricting the search space, and defining the cost function, as well as other aspects of the control system design such as control loop definition (also known as input-output pairing) [12]; all of these are currently solved, in the majority of cases, using expert knowledge. In this work, we propose a methodology to automate expert knowledge for a subset of steps in the control system design problem: input-output (I/O) pairing and stage definition for a multistage controller auto-tuning framework. We use Large Language Models (LLMs) to meet these expert knowledge requirements.

LLMs are deep learning models trained over a large volume of natural language data spanning multiple topics and scientific disciplines while following task-agnostic architectures [5]. Therefore, they are often used as a form of zero-shot or few-shot transferable models with a variety of applications [14][7][19]. This makes LLMs a prime target for automating expert knowledge, a task that is often unstructured and heavily relies on natural language interactions. While LLMs have shown positive results, multiple challenges persist in terms of LLM output formatting, consistency, and accuracy, generalization to more complex topics/benchmarks, factually incorrect responses, etc. Therefore, multiple works have focused on how to maximize performance from an already trained LLM, with prompt engineering (PE) being the most discussed approach. PE is based on the fact that a well-designed (natural language) prompt can dramatically improve the quality of the output in an LLM [10]. In this paper, we discuss a PE methodology to automate expert knowledge in control system design.

The contributions of this work are as follows: (1) We highlight existing challenges in automated controller design and automate expert knowledge using LLMs. We focus on expert knowledge requirements for I/O pairing and stage definition in a multistage decentralized controller auto-tuning framework. (2) We define a mathematical language based on system topology to describe a family of industrial multi-tank benchmarks. (3) We propose a system-informed prompt engineering algorithm to bind the mathematical language to the input prompt space of an LLM in order to automate expert knowledge for controller design. (4) We empirically evaluate the effectiveness of LLMs for substituting expert knowledge in I/O

pairing and stage definition tasks for a family of multi-tank benchmarks using performance metrics like: proper formatting of output prompt and correct expert knowledge provided. We show that the LLM can produce expert knowledge recommendations with over 97% accuracy with consistent formatting for I/O pairing and stage definition. (5) We empirically evaluate, for the first time (to the best of our knowledge), the relevance of controller tuning order on performance of tuned controllers, when tuning decentralized controllers independently and sequentially, for a family of multi-tank benchmarks. We show that, depending on the complexity of the system, and the degree of coupling between its components, a correct stage definition can improve the tuned controller performance by up to 52%

2 Related Work

2.1 Automated Controller Design

When discussing automated controller design, most works focus on controller auto-tuning, which is popularly framed as a derivative-free optimization problem [16][17]. While gradient-based methods are also used for controller auto-tuning [13], derivative-free formulations have the desired advantage of making little or no assumptions on objective or constraint functions.

Multiple authors have proposed metaheuristic approaches to controller auto-tuning, with [16] doing a detailed survey on evolutionary optimization algorithms for multiple-input-multiple-output processes. However, Bayesian optimization (BO) approaches have become quite popular in recent years, due to their high sample efficiency [2] and overall performance. For example [17] present a joint tuning methodology that uses BO to simultaneously tune an LQR controller, an unscented Kalman Filter (UKF), and a guidance and navigation system for an autonomous underwater vehicle (UUV), yielding satisfactory results across multiple objectives. Most of the available solutions focus strictly on the parameter search itself and how to solve it using some optimization tool, leaving the design of the optimization problem, or the control algorithm, as an expert knowledge requirement.

In their work, [2] also propose a Bayesian optimization solution for controller auto-tuning of PID controllers in an underwater vehicle, improving the computational complexity of the controller tuning task by decomposing the search space into smaller dimension subspaces. In order to do so, they propose a multistage framework that aims to formalize all the necessary steps in automated controller design. This formalization highlights a variety of existing challenges in automated controller design that are often ignored, or underexplored in recent works. Examples of these challenges are how to define the parameter search space, the constraints, or the cost function for the optimization problem, issues that are often not discussed, or attributed to expert knowledge. Similarly, how to define the stages in the multistage framework defined in [2] is still one of the open problems highlighted in the paper.

An attempt to address one of these expert knowledge requirements can be found in [12], where an optimal control formulation is proposed in order to simultaneously optimize the controller parameters and produce an I/O pairing for the system. I/O pairing, or control loop design, is often assumed to be already defined in most controller auto-tuning works [17], when, in reality, it is usually performed through expert knowledge.

Overall, state-of-the-art research on automated controller design is significantly challenged by expert knowledge requirements across different stages of the problem.

2.2 LLMs in System Design

Several papers have used LLMs for some aspect of system design, but fundamentally most approaches still adopt significant manual modeling, employing LLMs after the design process.

[14] uses a five-step LLM-based prompting approach that requires as inputs a piping and instrumentation diagram (P&ID) and natural language prompts. They evaluate the approach on a one-tank, one three-tank, and one four-tank system. Our approach differs in that we focus on control tuning and not diagnostics, we use a formal language for LLM prompting that provides guarantees about optimality of controller design, and we focus on control-model optimization and not just model generation.

A closely related approach, SmartControl [19], automatically determines the most suitable PID parameters to achieve the specified performance targets using PSO and DE algorithms. This process includes selecting the optimal gains that will ensure the system's fast and stable response. After optimization, the closed-loop step response is simulated and basic performance metrics (such as settling time, rise time, overshoot, etc.) are calculated and presented to the user via an interactive graphical interface. SmartControl emphasizes interactive design, allowing users to engage directly with the LLM agent to refine the controller design. The approach in the provided paper is more automated; the LLM generates the I/O pairing and stage definitions without direct user interaction during that phase.

ControlAgent [7] employs a range of techniques for integrating domain knowledge, e.g., a knowledge graph or other structured representation of control engineering principles. Our approach aims for full automation once the initial system description is provided; in contrast, ControlAgent involves more interactive elements, allowing users to refine the design process through feedback or iterative refinement with the LLM.

2.3 Systems Modeling using Component-Based Languages

Several frameworks exist to define systems based on interconnected component models, e.g., bond graphs, Simulink, and Modelica. The proposed approach is a fundamentally different framework from such approaches, most of which rely on manually-defined component models.

The approach is most similar to bond graphs (which focuses on representing energy flow and causality), although the current usage is quite different. In a bond graph, nodes consist of a standardized set of elements (e.g., C for capacitance, I for inertia, etc.) and edges/junctions represent system components and their interactions. The resulting graph allows for deriving system equations and performing simulations. Our proposed graph model is simpler than a bond graph model, as it does not specify the inherent physical properties of the nodes.

The two approaches are not mutually exclusive. One could conceivably use bond graphs to model a multi-tank system and then use the resulting model (or a simplified representation of it) as input to the LLM-based system for automated control design. The bond graph would provide a structured representation of the system's dynamics, while the LLM would handle the more complex task of selecting appropriate control strategies and parameters. The paper, however, focuses solely on the LLM-based approach without explicit integration of bond graphs. The paper's graph representation is a simplified topological representation, not a full bond graph model.

3 Preliminaries

3.1 Decentralized parametric control

Let's define a controller by a tuple $C = (U, \mathcal{Y}, \mathcal{U})$ where $\mathcal{Y} \subset \mathbb{R}^{n_y}$ is the space of system measurements/outputs (e.g.: level in a tank, altitude in a drone), $\mathcal{U} \subset \mathbb{R}^{n_u}$ is the space of system inputs (e.g. liquid flow from a pump, propeller commands in a quadcopter), and $U : \mathcal{Y}^t \times \mathcal{U}^{t-1} \to \mathcal{U}$ is a control algorithm that produces a system input at every timestamp t

considering system inputs and outputs from previous system interactions. We are interested in a family of control frameworks called *decentralized parametric controllers*, which are commonly used in industrial and robotic systems [2].

A parametric control algorithm is a parametric function $U : \mathcal{Y}^t \times \mathcal{U}^{t-1} \times \Xi \to \mathcal{U}$ such that $\mathbf{u}(t) = U(\mathbf{y}(0), \ldots, \mathbf{y}(t), \mathbf{u}(0), \ldots, \mathbf{u}(t-1), \boldsymbol{\xi})$, with $\mathbf{y}(t) \in \mathcal{Y}, \mathbf{u}(t) \in \mathcal{U}, \boldsymbol{\xi} \in \Xi$, where the control algorithm is defined by a set of parameters $\boldsymbol{\xi}$ that must be previously adjusted (tuned). Parametric controllers are very common, with the proportional-integral-derivative controller and its P, I, and D parameters being the industry standard [16].

A decentralized control approach is defined by a set of independent control algorithms $\mathcal{C} = \{U_1, U_2, \ldots, U_{n_u}\}$ where every control algorithm U_i matches a single system input u_i with a single system output y_i, and the changes in that system input are only conditioned by the corresponding system output, i.e.: $u_i(t) = U(y_i(0), \ldots, y_i(t), u_i(0), \ldots, u_i(t-1), \boldsymbol{\xi}_i)$. This is a common simplification in control system design [2] with assumptions regarding system coupling. The implications of these assumptions are discussed further in this paper.

3.2 Directed graphs

A graph is defined by a tuple $G = (V, E)$ where $v_i \in V$ is a set of elements called *vertices* $v_i \in \mathbb{N}$ and $e \in E$ is a set of pairs $e = (v_i, v_j); v_i, v_j \in V$ called *edges* [6]. An edge describes a relationship between two vertices, we can say that an edge *joins* two vertices. For the purpose of this work, we are interested in graphs with the following properties:

- **Directed**: In a directed graph, the edges are ordered pairs and have an implied direction, i.e.: $e_1 = (v_1, v_2) \neq e_2 = (v_2, v_1)$. An edge in a directed graph is called a *directed edge*.
- **Simple**: A simple graph is a graph without loops. A loop in a graph is defined as an edge that joins a vertex to itself, i.e.: $e = (v_i, v_j); v_i = v_j$.
- **Edge-Labeled**: In an edge-labeled graph $G = (V, E, L)$, a labeling function $l : E \to L$ is defined to assign a label from the label set L to each edge in the edge set.

Let's discuss some graph theory concepts that will be relevant in the rest of this work [4].

▶ **Definition 1** (Directed Path). *A directed path is a sequence of distinct edges* $\{e_1, e_2, \ldots, e_{n-1}\}$ *in a graph* $G = (V, E)$ *that joins a sequence of distinct vertices* $\{v_1, v_2, \ldots, v_n\}$ *such that* $e_i = (v_i, v_{i+1}), e_i \in E, \{v_i, v_{i+1}\} \in V$ *is a directed edge.*

▶ **Definition 2** (Strongly Connected Component). *A directed graph is considered to be strongly connected if every vertex is reachable from every other vertex in the graph, i.e.: for any vertex pair* $(v_i, v_j) \in V$*, there is at least one directed path that goes from* v_i *to* v_j*. A strongly connected component of a graph* $G = (V, E)$ *is a subgraph* $G' \subseteq G$ *that is strongly connected, and is maximal with this property, i.e.: no additional edges or vertices from* G *can be included in* G' *without breaking its property of being strongly connected.*

Strongly connected components in a graph can be computed in linear time by a variety of algorithms (e.g.: depth-first-search algorithms) [4].

3.3 Large Language Models

Let us define the prompt space $\mathcal{Z}$ [11]. The elements in $\mathcal{Z}$ are a composition of tokens selected from a token vocabulary $t \in \mathcal{T}$, where $z = \{t_1, t_2, \ldots, t_m\}$ can be a sequence of tokens of any length $m \in \mathbb{N}$. An LLM is then defined as a transformation $LLM : \mathcal{Z} \to \mathcal{Z}, z_o = LLM(z_i)$ where, for a given input prompt z_i, the model produces an output prompt z_o.

A common feature in LLM APIs is the *system prompt*. A system prompt $z_s \in \mathcal{Z}$ permeates every interaction [20] with the LLM, providing context, setting the role and the tone for the LLM responses, ensuring output prompt formatting, etc. Particular implementations of the system prompt vary depending on the LLM; in some cases it is provided as an initial prompt before the input prompt [11], while in other cases a dedicated system token is implemented. For the purpose of this work, we are interested in LLMs with system prompt (z_s) options, i.e.: $LLM : \mathcal{Z} \times \mathcal{Z} \to \mathcal{Z}, z_o = LLM(z_s, z_i)$. In Section 4.4, we define bindings from a graph language describing a family of physical systems to the input and system prompt spaces.

3.4 Bayesian optimization

Bayesian optimization is a technique for globally optimizing black-box functions that are expensive to evaluate [9]. We consider the global minimization problem: $\boldsymbol{\xi}^* = \arg\min_{\boldsymbol{\xi} \in \Xi} f_\xi(\boldsymbol{\xi})$, with input space Ξ and objective function $f_\xi : \Xi \to \mathbb{R}$. We consider costly-to-evaluate functions f_ξ for which we have a limited number of evaluations available before we propose an optimum $\boldsymbol{\xi}^*$ in at most J_{end} iterations. We assume we have access only to noisy evaluations of the objective $l = f_\xi + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ is i.i.d. Gaussian noise with variance σ_n^2. Finally, we make no assumptions regarding gradients or convexity properties of f_ξ.

The main steps of a BO routine in iteration j involve (1) response surface learning, (2) optimal input selection $\boldsymbol{\xi}_{j+1}$, and (3) evaluation of the objective function f_ξ at $\boldsymbol{\xi}_{j+1}$.

The BO framework uses the predictive mean and variance of a Gaussian Process (GP) [15] model to formulate an acquisition function that trades off exploitation, testing promising controller parameters given the current knowledge, with exploration, sampling unexplored regions of the design space. The BO algorithm can be initiated without any past observations, but it is usually more efficient to calibrate the GP model hyper-parameters and calculate a prior by first collecting an initial set of observations.

4 Methodology

4.1 Controller tuning framework

We define the controller tuning task as a multi-objective optimization (MOO) problem:

$$\min_{\boldsymbol{\xi}} \mathbf{L}(\boldsymbol{\xi}) = [\mathcal{L}_1(\boldsymbol{\xi}), \ldots, \mathcal{L}_{n_L}(\boldsymbol{\xi})], \quad \text{s.t. } \boldsymbol{\xi} \in \Xi, \tag{1}$$

where $\boldsymbol{\xi} \in \Re^{n_\xi}$ are controller parameters and $\mathbf{L}(\boldsymbol{\xi})$ are objectives. We consider an MOO problem for a more general definition; however, controller auto-tuning is commonly framed as a scalar objective optimization problem. Our proposed framework is fully compatible with both the multi-objective and scalar-objective definitions of the problem. We assume a system with decentralized parametric controllers; other than that, the proposed framework makes no assumptions on the parameter and objective spaces. This general tuning problem can be redefined as a multistage optimization problem where a subset of controllers is tuned at every stage. Each stage can be defined as an MOO sub-task. Fig. 1 illustrates the process.

At stage $i = 1, 2, \ldots, n_g$ of the tuning process, a subset of control parameters ($\Xi^i \subseteq \Xi$) are tuned to minimize some stage-specific objectives ($\mathbf{L}^i$), given reference signal R^i. The rest of the control parameters are fixed at a previous stage, or before the tuning process begins, through the constraint parameters ($\bar{\boldsymbol{\xi}}^i, P^i$). At each stage, a subset ($\boldsymbol{\xi}^i_{min}$) of the optimal control parameters ($\boldsymbol{\xi}_{min}$) is produced as a result of an MOO sub-task described below.

The I/O pairing and the definition of the stages are task-specific steps, both of which are commonly solved using expert knowledge [2][16]. The main contribution of this work is a methodology to automate this expert knowledge requirement and produce an I/O pairing and stage definition using LLMs. Further details on these steps can be found in Section 4.3.

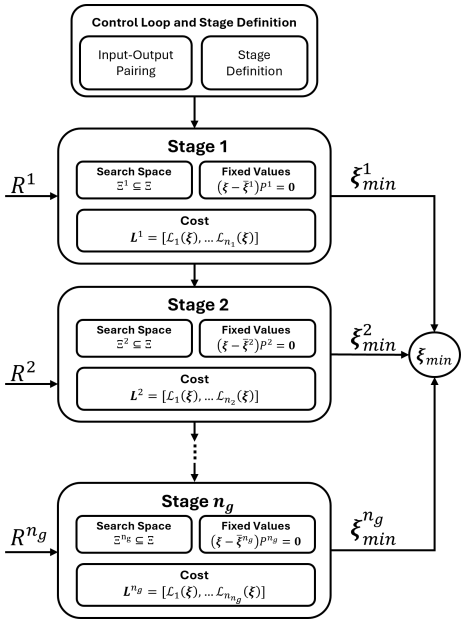


Figure 1 Controller tuning as a Multistage Problem.

The controller tuning optimization task is decomposed into sub-tasks defined as:

$$\min_{\boldsymbol{\xi}} \quad \mathbf{L}^i(\boldsymbol{\xi}) = [\mathcal{L}_1(\boldsymbol{\xi}), \ldots, \mathcal{L}_{n_i}(\boldsymbol{\xi})] \tag{2a}$$

$$\text{s.t.} \quad \boldsymbol{\xi} \in \Xi^i \tag{2b}$$

$$(\boldsymbol{\xi} - \bar{\boldsymbol{\xi}}^i)P^i = 0 \tag{2c}$$

where (2c) is a constraint that sets control parameters to specified values ($\bar{\boldsymbol{\xi}}^i$) from a previous stage or from before the tuning process begins.

These optimization sub-tasks can be solved using any optimization algorithm, as long as the cost function and search space are properly defined. For the purpose of this work, we use Bayesian Optimization to solve the optimization sub-tasks; we refer the reader to [2] for a more detailed discussion on the optimization sub-tasks and the use of Bayesian Optimization.

4.2 Graph Language For System Representation

We now define a language, in the form of an enhanced system graph, to describe physical systems in terms of reservoirs, actuators, and connections:

▶ **Definition 3** (Enhanced System Graph). *An enhanced system graph is a directed, simple (no loops), edge-labeled graph $G = (V, E, L)$ that describes relationships between tanks and pumps in a multi-tank system. The graph is separated into two subgraphs $G = G^A \cup G^T$ where $G^A = (V^A, E^A, L^A)$ is the actuator subgraph and $G^T = (V^T, E^T, L^T)$ is the tank subgraph.*

Let's first discuss the tank subgraph G^T which describes relations between the different tanks. Each vertex in the set $V^T = \{1, 2, \dots N_T\}, v_i^T \in \mathbb{N}$, represents a tank in the system, where tank labels (vertices) are assigned arbitrarily, and N_T is the number of tanks in the system. The edge set $E^T \subset V^T \times V^T, e_k^T = (v_i^T, v_j^T)$ represents a liquid flow connection between the tanks. The direction of the edges represents possible flow directions. A regular directed edge (v_i^T, v_j^T) represents liquid flow that is only feasible from tank v_i^T to tank v_j^T, for example, the drain of a gravity-drained tank that feeds a tank below it. A bidirectional edge $\{(v_i^T, v_j^T), (v_j^T, v_i^T)\}$ represents an interconnection between tanks where liquid flow can occur in both directions, for example, two horizontally aligned tanks connected by a pipe.

Based on the edge directionality and the semantics of edges, we can assign a labeling to edges for constructing LLM prompts. The edge-label set $L^T = \{above, below, interconnected\}$ provides the necessary information about the relationships between the tanks for prompt engineering. The labeling function $l^T(e)$ is defined for any edge $e \in E^T, e = (v_i, v_j)$:

$$l^T(e) = \begin{cases} interconnected & (v_j, v_i) \in E^T \\ above & (v_j, v_i) \notin E^T \wedge v_i < v_j \\ below & (v_j, v_i) \notin E^T \wedge v_i > v_j \end{cases} \tag{3}$$

where an edge is labeled *interconnected* if it is bidirectional (i.e.: the opposite direction edge is also in the graph) and *above* or *below* if the edge is not bidirectional (i.e.: the opposite direction edge is not in the graph). An edge $e = (v_i, v_j)$ is labeled *above* if vertex v_i is of lower numerical value than vertex v_j and *below* if the opposite is true. Note that the value of the vertices, which effectively act as labels for the tanks, are assigned arbitrarily, and the purpose of this labeling strategy is to ensure a consistent distinction between types of edges, which becomes relevant when building natural language prompts for the LLM. Therefore, the distinction between *above* and *below* edges is done without loss of generality.

We then define the actuator subgraph $G^A = (V^A, E^A, L^A)$ where each vertex in $V^A = \{1, 2, \dots N_A\}, v_i^A \in \mathbb{N}$ represents an actuator in the multi-tank system, e.g.: a pump. Every edge in $E^A \subseteq V^A \times V^T$ represents a connection between an actuator and a tank, where an edge $e = (v_i^A, v_j^T) \in E^A$ is always directed from an actuator vertex to a tank vertex. One actuator vertex can be connected to multiple tank vertices. The labeling function for the actuator subgraph $l^A : V_A \to L_A$ maps each actuator vertex to a corresponding vertex label that describes attributes of the fluid going through the actuator which may include: temperature of the fluid, composition or concentration of the fluid, etc.; e.g.: $l^A(v) = hot\,water$.

Fig. 2a shows an example three-tank system and its corresponding graph representation (Fig. 2b). Elements from the tank subgraph are presented in continuous lines while the actuator subgraph is represented by discontinuous lines.

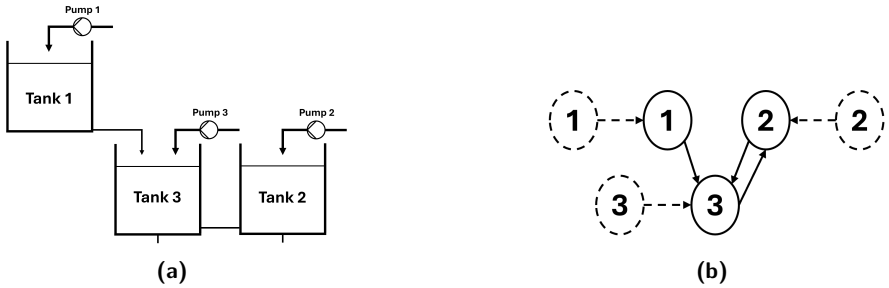


Figure 2 Graph representation (b) of an example 3-tank system (a).

4.3 Input Output Pairing and Stage Definition

In this section we formalize the concepts of I/O pairing and stage definition, and how they relate to the controller auto-tuning framework defined in Section 4.1.

4.3.1 Input-Output Pairing

Let us consider a system with input space $\mathcal{U} \subseteq \mathbb{R}^{n_u}$ consisting of n_u system inputs, corresponding to the output of n_u decentralized controllers. Let us also consider an output space $\mathcal{Y} \subseteq \mathbb{R}^{n_y}$ consisting of n_y outputs, corresponding to known system variables (either by direct measurement or estimation). In the context of decentralized controllers, we define an I/O pairing as a matrix $M^{I/O} \in \mathcal{M}_{n_u \times n_y}$, $m_{ij}^{I/O} \in \{0, 1\}$ where $m_{ij}^{I/O} = 1$ if input u_i and output y_j are paired, and $m_{ij}^{I/O} = 0$ otherwise. In practice, an input and an output being paired means that the value of the output is used to compute the paired system input through the respective control algorithm. Every input must be paired with an output exactly once.

Let us illustrate this concept using the example from Fig. 2. This three tank system is fed by three input pumps, pump 1 feeds tank one with an input flow u_1, pump 2 feeds tank two with an input flow u_2, and pump 3 feeds tank three with an input flow u_3. The levels in the three tanks are measured, resulting in the respective outputs y_1, y_2, and y_3. The I/O pairing task in this case is trivial, since it is obvious we want to control the level in tank i using the pump that feeds tank i. Assuming a set of inputs: {FlowPump1, FlowPump2, FlowPump3}, respectively $\{u_1, u_2, u_3\}$, and a set of outputs: {LevelTank1, LevelTank2, LevelTank3}, respectively $\{y_1, y_2, y_3\}$; Fig. 3 shows example I/O pairings for the system.

$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

(a) Correct Input-Output Pairing.

$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

(b) Incorrect Input-Output Pairing.

Figure 3 Example Input-Output Pairings.

I/O pairing is equivalent in the literature to control loop design [12] since, pairing an input and an output defines a feedback loop for the corresponding controller. Following the correct input-output pairing (Fig. 3a), we define three control loops: {FlowPump1-LevelTank1, FlowPump2-LevelTank2, FlowPump3-LevelTank3}. We formalize a control loop as a tuple $\pi_i = (u_i, y_j)$ and tuning a control loop refers to tuning the corresponding controller.

4.3.2 Stage Definition

Once we determine an I/O pairing in the system, we must define the stages for the multistage optimization problem presented in Section 4.1. This involves defining the total number of stages n_g and, for every stage i, the controller tuned at this stage and the control parameters fixed from previous stages. This information is then summarized into the stage-specific parameters $(\Xi^i, \bar{\boldsymbol{\xi}}^i, P^i)$ for each optimization subtask (2). In order to produce a concise description of the stages in the framework, we define a stage matrix $M^{SD} \in \mathcal{M}_{n_g \times n_u}$, $m_{ij}^{SD} \in \{0, 1, 2\}$ where every row represents a stage, and every column represents a control loop. An element in the matrix m_{ij}^{SD} is 1 if the controller for control loop j is tuned at stage i, 0 if

control loop j is open at stage i, and 2 if control loop j is closed at stage i using control parameters tuned in a previous stage. For the purpose of this work, every controller must be tuned exactly once, and at least one controller must be tuned at every stage.

What is a correct stage definition, as well as the relevance of this process, is task-specific, and is often determined from expert knowledge based on the couplings (I/O interactions) in the system. Ideally, the output of a control loop $\pi_i = (u_i, y_j)$ should only be affected by the corresponding input u_i. However, this is often not the case and output y_j is affected by variations in the inputs of other loops $u_{k \neq i}$; when this happens, we say loop π_i is coupled with loop $\pi_k = (u_k, y_l)$. This relationship is not symmetric, if control loop π_i is coupled with control loop π_k, control loop π_k can be decoupled from π_i, where variations in input u_i do not affect output y_l. In the context of controller tuning, we tune control loops that are isolated (i.e.: not coupled to any loops) first and then tune the coupled loops. When tuning a controller for a coupled loop π_i, if possible, we want all the loops that π_i is coupled with to be closed using control parameters tuned in previous stages, which minimizes variations in the variables from the coupled loops, making π_i behave like a decoupled loop. Following this methodology we can see that correct stage definition for a system is not always unique.

Let us illustrate the concept of stage definition using the example from Fig. 2. The control loops in the system are (Section 4.3.1): {FlowPump1-LevelTank1, FlowPump2-LevelTank2, FlowPump3-LevelTank3}, for simplicity, $\{(u_1, y_1), (u_2, y_2), (u_3, y_3)\}$. In this case, the correct stage definition should start by tuning the control loop corresponding to tank 1 (u_1, y_1). This is an isolated loop in the system since, due to the connection between tank 1 and 3 being gravity-based, no variations in the pumps feeding tank 2 and 3 can affect the level in 1, while variations in the pump feeding tank 1 can affect the level in tanks 2 and 3. Tanks 2 and 3 are interconnected and flow between them can go in either direction, making loops (u_2, y_2) and (u_3, y_3) coupled both ways; therefore, these loops can be tuned in any order. In summary, any stage definition that tunes control loop (u_1, y_1) in the first stage, and maintains this loop closed for further stages, is considered correct. Fig. 4 shows an example of a correct and incorrect stage definitions of the system, replacing $\{0, 1, 2\}$ with $\{open, tuned, closed\}$.

$$
\begin{array}{c} \\ \textit{Stage}\,1 \\ \textit{Stage}\,2 \\ \textit{Stage}\,3 \end{array}
\begin{array}{c}
\begin{array}{ccc} (u_1, y_1) & (u_2, y_2) & (u_3, y_3) \end{array} \\
\begin{pmatrix} \textbf{tuned} & \textit{open} & \textit{open} \\ \textit{closed} & \textit{open} & \textbf{tuned} \\ \textit{closed} & \textbf{tuned} & \textit{closed} \end{pmatrix}
\end{array}
$$

(a) Example Correct Stage Definition.

$$
\begin{array}{c} \\ \textit{Stage}\,1 \\ \textit{Stage}\,2 \\ \textit{Stage}\,3 \end{array}
\begin{array}{c}
\begin{array}{ccc} (u_1, y_1) & (u_2, y_2) & (u_3, y_3) \end{array} \\
\begin{pmatrix} \textit{open} & \textbf{tuned} & \textit{open} \\ \textbf{tuned} & \textit{closed} & \textit{open} \\ \textit{closed} & \textit{closed} & \textbf{tuned} \end{pmatrix}
\end{array}
$$

(b) Example Incorrect Stage Definition.

Figure 4 Example Stage Definitions.

4.4 LLM prompt design

In this section we discuss how to design input and system prompts (Section 3.3) for an LLM in order to automate I/O pairing and stage definition, given limited structural knowledge of the system, for a family of multi-tank systems. We present an algorithm for input prompt design given the graph representation of the system defined in Section 4.2.

A system with three gravity drained tanks filled with fluid. Tank 1 is positioned above tank 3 and the drain of tank 1 feeds tank 3 as an input flow. Tank 2 and tank 3 are horizontally aligned and interconnected by a pipe. The system is fed by 3 pumps, producing input flows. Pump 1 feeds tank 1. Pump 2 feeds tank 2. Pump 3 feeds tank 3.
Inputs: [FlowPump1, FlowPump2, FlowPump3]
Outputs: [LevelTank1, LevelTank2, LevelTank3]
3 control loops with 3 PID controllers:
[FlowPump1-LevelTank1, FlowPump2-LevelTank2, FlowPump3-LevelTank3]
Propose a methodology to tune the controllers sequentially

Figure 5 Input Prompt for Stage Definition.

4.4.1 Input prompt design

In this section we describe the methodology (Algorithm 1) used to design an input prompt for stage definition in a multi-tank system assuming a graph representation (Def. 3) of the system is available. The resulting input prompt consists of 6 distinct sections: introduction, hierarchical relations, interconnections, actuators, input-output-loop list, and task instruction.

We start by introducing the system and the number of tanks ($N_T = |V^T|$) in it and then describe the hierarchical relations between tanks. In order to do this, we define a set of single direction edges, $E_H = \{(v_i, v_j) \in E^T \mid (v_j, v_i) \notin E^T\}$. Every edge in E_H represents a hierarchical relation between tanks where only one flow direction is feasible, whether it is due to gravity or a pump forcing a fixed flow. This section of the prompt design algorithm generates a sequence of $|E_H|$ sentences (one for each element in E_H) that describe the hierarchical relations between pairs of tanks using the corresponding labels $l^T(e_i) \in \{above, below\}$. The set E_H is ordered from lowest to highest according to $min(v_i, v_j)$ (min is the minimum between v_i and v_j). The edge ordering of set E_H is done for consistency, which we have found to impact performance significantly when designing the prompt.

We then describe groups of tanks that are interconnected. To do so, we define a set of subgraphs $\mathcal{G}_I = \{G'_i \subseteq G^T \mid |G'_i| > 1 \wedge G'_i \text{ is strongly connected}\}$. For every connected subgraph G'_i, we produce a sentence highlighting interconnection between the tanks represented by the subgraph. The tanks don't need to be horizontally aligned, we use this sentence structure to stay consistent with the vertical alignment description of the system hierarchy.

Afterwards, we describe the actuators in the system and their relationship with the corresponding tanks. For the purpose of this work, we only consider the family of systems with actuators as pumps, however, the actuator space can be easily expanded by making the actuator label space L^A more expressive. First we list the number of pumps $N_A = |V_A|$. Then, we add a sentence to the prompt for each actuator $v_i \in V^A$, where we list the set of tanks $T_i = \{t^1_{v_i}, t^2_{v_i}, \ldots, t^n_{v_i}\}$ that are fed by pump v_i. If the fluid through the actuator has associated attributes, these are added to each actuator sentence by means of $l^A(v_i)$.

Subsequently, we provide an ordered list of inputs and outputs in the system and, for the case of the stage definition task, we also provide a description of the control loops. Finally, instructions about the task (stage definition or I/O pairing) are provided. Fig. 5 shows the resulting input prompt for stage definition designed for the example system in Fig. 2.

Algorithm 1 Input Prompt Design Methodology.

Input: G
$prompt \leftarrow$ "A system with $\{N_T\}$ gravity drained tanks filled with fluid."
`/* Describe hierarchical relations */`
foreach $e_i = (v_{i_1}, v_{i_2}) \in E_H$ **do**
  $prompt \leftarrow prompt +$ "Tank $\{min(v_{i_1}, v_{i_2})\}$ is positioned $\{l^T(e_i)\}$ tank $\{max(v_{i_1}, v_{i_2})\}$ and the drain of tank $\{v_{i_1}\}$ feeds tank $\{v_{i_2}\}$ as an input flow."
end
`/* Describe interconnection between tanks */`
foreach $G'_i \in \mathcal{G}_I$ **do**
  $prompt \leftarrow prompt +$ "Tank $\{v'_{i_1}\}$, tank $\{v'_{i_2}\}$, ..., and tank $\{v'_{i_n}\}$ are horizontally aligned and interconnected by a pipe."
end
`/* Describe actuator relations */`
$prompt \leftarrow prompt +$ "The system is fed by $\{N_A\}$ pumps, producing input flows."
foreach $v_i \in V_A$ **do**
  $prompt \leftarrow prompt +$ "Pump $\{v_i\}$ feeds tank $\{t^1_{v_i}\}$, tank $\{t^2_{v_i}\}$, ..."
  if $l^A(v_i) \neq \varnothing$ **then**
    $prompt \leftarrow prompt +$ " with $\{l^A(v_i)\}$."
end
`/* System Inputs, Outputs, Loops, and Task */`
$prompt \leftarrow prompt + \{$*List of Inputs and Outputs*$\}$
if *task* = *StageDefinition* **then**
  $prompt \leftarrow prompt +$ "$\{N_u\}$ control loops with $\{N_u\}$ PID controllers"
  $prompt \leftarrow prompt + \{$*List of Control Loops*$\}$
  $prompt \leftarrow prompt +$ "Propose a methodology to tune the controllers sequentially".
else if *task* = *IOPairing* **then**
  $prompt \leftarrow prompt +$ "Propose an input-output pairing for the system".
Output: *prompt*

4.4.2 System Prompt Design

A well-designed system prompt can enhance the performance of the LLM interaction. The system prompt design strategy consists of four sections: a role statement, a description of the task, output formatting instructions, and an output format example.

The role statement is a common practice in system prompt design, usually implemented with a single sentence in the form: *"You are {role}"*. In this case we define the role as *"expert control engineer"*. We then describe the desired task (I/O pairing or stage definition).

The large output prompt space $\mathcal{Z}$, along with the stochastic nature of the LLM (assuming non-zero temperature), results in challenges when processing the LLM output programmatically in a fully automated framework. For example, in the case of I/O pairing, we want to produce a matrix like the one in Fig. 3 from a natural language output $z_o \in \mathcal{Z}$. To parse this output correctly, a consistent output prompt formatting is desired. We ensure this by adding formatting instructions to the system prompt, as well as an example desired output.

System prompts are independent from the particular configuration of the multi-tank system, and their design depends on the desired task. Figures 6a and 6b show the system prompts used for the I/O pairing and stage definition tasks respectively.

You are an expert control engineer. Provide a matrix showing recommended input-output pairings for a system. Your answer must contain a table wrapped in <ans></ans> and nothing more. The table must be formatted exactly as described below:
- Columns: System Outputs
- Rows: System Inputs
- Cells: "1"=input and output are paired, "0"=input and output are not paired
All inputs must be paired exactly once.
Example:
<ans>
[Output1] [Output2] [Output3] ... [OutputN]
Input1 0 1 0 ... 0
Input2 1 0 0 ... 0
...
InputS 0 0 1 ... 0
</ans>

(a) System prompt for Input-Output Pairing.

You are an expert control engineer. Your task is to provide a matrix showing sequential PID controller tuning order. Upstream control loops should be tuned first. Your answer must contain a table wrapped in <ans></ans> and nothing more. The table must be formatted exactly as described below:
- Columns: PID controllers by control loop
- Rows: Tuning steps
- Cells: "1"=tuned at this step, "0"=opened at this step, "2"=tuned in previous step
At least one controller must be tuned per step. Each controller is tuned exactly once.
Example:
<ans>
[Loop1] [Loop2] [Loop3] ... [LoopN]
Step1 0 1 0 ... 0
Step2 1 2 0 ... 0
...
StepS 0 2 1 ... 2
</ans>

(b) System prompt for Stage Definition.

5 Experimental Design

5.1 Performance Metrics

In order to evaluate the proposed I/O pairing and stage definition methodologies we use multiple performance metrics which evaluate output prompt formatting and accuracy, as well as auto-tuned controller performance.

5.1.1 Performance Metrics for LLM output

Consider a system benchmark b, e.g.: three cascading tanks, two interconnected tanks, and a task $\gamma \in \{IOPairing, StageDefinition\}$. Let's then assume an LLM is prompted with the corresponding system (z_s^γ) and input $(z_i^{b,\gamma})$ prompts such that the output prompt $z_o = LLM(z_s^\gamma, z_i^{b,\gamma})$ is produced. Let us define the following set of performance metrics in order to evaluate output prompt z_o: correct formatting, accuracy, and percentage of accurate over correctly formatted.

Correct Formatting. For a given task γ and benchmark b, let us define a correctly formatted set $\mathcal{Z}_F^{b,\gamma}$ with all the output prompts that match the restrictions imposed in the corresponding system and input prompts $[z_s^\gamma, z_i^{s,\gamma}]$. An output prompt $z_o = LLM(z_s^\gamma, z_i^{b,\gamma})$ is considered correctly formatted if $z_o \in \mathcal{Z}_F^{b,\gamma}$. We are also interested in the average performance over a set of N interactions (to account for the stochastic nature of the LLM). We define percent of correctly formatted outputs as:

$$\Psi_F = \sum_{i=1}^{N} \frac{L_F(z_o)}{N} * 100,\ z_o = LLM(z_s^\gamma, z_i^{b,\gamma});\ L_F(z_o) = \begin{cases} 1, & z_o \in \mathcal{Z}_F^{b,\gamma} \\ 0, & z_o \notin \mathcal{Z}_F^{b,\gamma} \end{cases} \tag{4}$$

Accuracy. For a given task γ and benchmark b, let us define a ground truth set $\mathcal{Z}_T^{b,\gamma}$ with all the output prompts that an expert would consider correct. An output prompt $z_o = LLM(z_s^\gamma, z_i^{b,\gamma})$ is considered accurate if it is in the ground truth set $z_o \in \mathcal{Z}_T^{s,\gamma}$. An output prompt must be correctly formatted in order to be accurate, i.e.: $\mathcal{Z}_T^{b,\gamma} \subseteq \mathcal{Z}_F^{b,\gamma}$. Similarly to proper formatting, we are interested in the percent of accurate outputs:

$$\Psi_T = \sum_{i=1}^{N} \frac{L_T(z_o)}{N} * 100,\ z_o = LLM(z_s^\gamma, z_i^{b,\gamma});\ L_T(z_o) = \begin{cases} 1, & z_o \in \mathcal{Z}_T^{b,\gamma} \\ 0, & z_o \notin \mathcal{Z}_T^{b,\gamma} \end{cases} \tag{5}$$

Accuracy over Correct Formatting. Finally, we define a metric for LLM outputs that is only valid over multiple interactions with the LLM: $\Psi_{T/F} = \frac{\Psi_T}{\Psi_F} * 100$. The purpose of this metric is to evaluate the ability of the LLM to produce accurate outputs, assuming that this output is correct (hence ignoring incorrectly formatted responses). This is arguably the most important metric for LLM outputs due to the fact that, if an output is incorrectly formatted, the interaction with the LLM can be repeated $z_o = LLM(z_s^\gamma, z_i^{b,\gamma})$ until a correctly formatted output $z_o \in \mathcal{Z}_F^{b,\gamma}$ is produced. This approach is viable since expert knowledge is not required to verify the output formatting, however, a low correct formatting metric Ψ_F can lead to higher inference costs due to unnecessary interactions.

5.1.2 Performance of tuned controllers

We are also interested in evaluating the performance of the tuned controllers given a stage definition produced by the LLM. This performance metric is only valid for properly formatted output prompts ($z_o \in \mathcal{Z}_F^{b,\gamma}$). We are particularly interested in evaluating performance of tuned controllers for different stage definitions in order to validate our hypotheses that stage definition is critical to achieve good controller performance when auto-tuning decentralized controllers sequentially, which, to the best of our knowledge, is a comparison that has not been performed before. We do not evaluate the performance of the tuned controllers for different I/O pairings since this is a well-known topic in the literature [12].

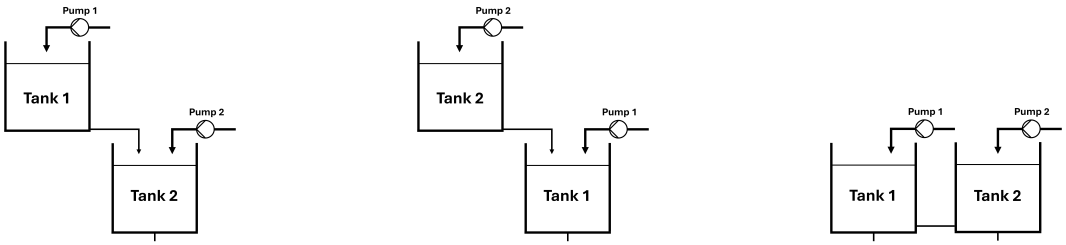
We measure performance of tuned controllers using the integral absolute error (IAE) metric [16][12][2], a well known controller performance metric that evaluates the ability of the system, using the tuned controllers, to minimize the error in the measured variables $\mathbf{y}$ with respect to a reference trajectory $\mathbf{y}_R$. We use MATLAB *bayesopt* method to implement a Bayesian optimization algorithm (with IAE as the cost function) and solve the controller tuning problem described in Section 4.1 for different stage definitions proposed by the LLM interactions. A detailed explanation of the full auto-tuning algorithm can be found in [2].

5.2 Multi-tank benchmarks

In this section, we will describe the different multi-tank configurations used to empirically evaluate the proposed methodology. In order to highlight the details of each task, we use three distinct set of benchmarks to evaluate: I/O pairing LLM output, stage definition LLM output, and performance of tuned controllers for stage definition.

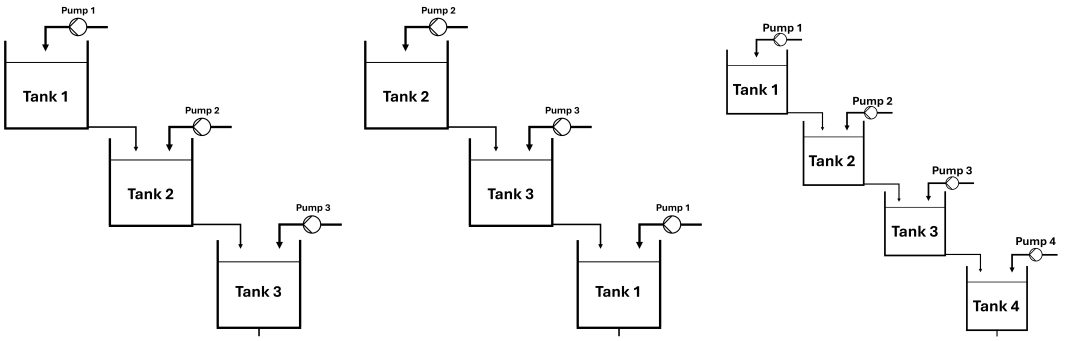
For I/O pairing evaluation we use 5 configurations of the multi-tank system which include: (1) three cascaded tanks (Fig. 8a) and (2) four cascaded tanks (Fig. 8c). We also evaluate I/O pairing on three system configurations taken from [12], where input-output pairing is framed as an optimization problem, and solved iteratively. These configurations are: (3) a well-known quadruple tank system (Fig. 9a), (4) a variation of the quadruple tank system (Fig. 9b), and (5) a single tank system where both level and temperature in the tank are controlled (Fig. 9c). Table 1 shows the list of configurations for I/O pairing evaluation.

For stage definition LLM output evaluation we use 7 configurations of the multi-tank system: (1) two cascaded tanks (Fig. 7a), (2) alternative two cascaded tanks (Fig. 7b), (3) two interconnected tanks (Fig. 7c), (4) three cascaded tanks (Fig. 8a) (5) alternative three cascaded tanks (Fig. 8b), (6) the three tank system illustrated in Fig. 2, and (7) four cascaded tanks (Fig. 8c). Table 2 shows the list of configurations for stage definition LLM output evaluation. The pairs of configurations (1)-(2) and (4)-(5) respectively describe a system with the same layout, but different tank labeling. We evaluate alternative labeling of the same configurations to test robustness of the prompt design methodology to small changes in the system description, which has shown to be a challenging task [3].



(a) (2T, 1a2). **(b)** (2T, 1b2). **(c)** (2T, 1-2ic).

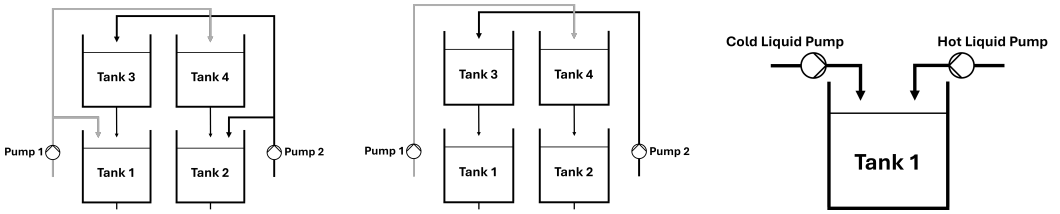
Figure 7 Two-Tank Benchmarks for Empirical Evaluation.



(a) (3T, 1a2, 2a3). **(b)** (3T, 1b3, 2a3). **(c)** (4T, 1a2, 2a3, 3a4).

Figure 8 Three and Four Cascaded Tanks Benchmarks for Empirical Evaluation.

Finally, we evaluate the performance of tuned controllers following different stage definitions. We evaluate the performance for three benchmarks: (1) two cascaded tanks, (2) three cascaded tanks, (3) four cascaded tanks. Tank labeling is not relevant in the case of control performance evaluation.



(a) (4T, 1b3, 2b4) [12]. **(b)** (4T, 1b3, 2b4)*. **(c)** 1T Temperature [12].

Figure 9 Multi-tank Benchmarks taken from [12] for Empirical Evaluation.

Controller performance is evaluated for different levels of coupling. For this purpose, we define a coupling parameter φ that represents how coupled the control loops in the system are. For the case of the cascaded configurations used, we model the coupling parameter φ as a factor that linearly maps the diameters of the drain in the tanks. A high value of φ results in a larger drain diameter, which results in higher drain flows for each tank, which in turn results in higher input flows into the respective tanks below, translating into higher coupling between tanks. Conversely, a small drain diameter might result in practically non-existent coupling between the tanks, which would result in a set of isolated control loops, rendering the motivation for proper stage definition null. We evaluate the controller performance for three coupling levels: nominal coupling and 1.5 and 2 times nominal coupling.

Three stage definitions are evaluated, labeled: *correct stage definition*, *incorrect stage definition*, and *independent stage definition*. Correct stage definition for the three cascaded configurations refers to a solution that tunes the control loops following the hierarchy; i.e.: the control loop for the tank on top is tuned first, then the tank below, and so on until the bottom tank is reached, while at each stage, control loops tuned in previous stages are closed and controllers are fixed to their tuned parameters. Incorrect stage definition refers to a solution that tunes control loops in the opposite order to the correct stage definition while maintaining control loops from previous stages closed. Independent stage definition refers to the process of tuning a control loop independently at each stage, ie.: control tunings from previous stages do not carry over to the current stage and every loop, except the loop being tuned, is open at each stage. The tuning order in this case is not important. This is technically also an incorrect stage definition for the cascaded configurations, however, we include it as a separate case because it is a common approach to controller tuning in the related literature [2][12]. Fig. 10 shows example stage definitions for the two cascaded tanks system.

$$\begin{array}{c} \\ \textit{Stage}\,1 \\ \textit{Stage}\,2 \end{array} \begin{array}{c} (u_1, y_1) \\ \begin{pmatrix} \textbf{tuned} \\ \textit{closed} \end{pmatrix} \end{array} \begin{array}{c} (u_2, y_2) \\ \begin{pmatrix} \textit{open} \\ \textbf{tuned} \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ \textit{Stage}\,1 \\ \textit{Stage}\,2 \end{array} \begin{array}{c} (u_1, y_1) \\ \begin{pmatrix} \textit{open} \\ \textbf{tuned} \end{pmatrix} \end{array} \begin{array}{c} (u_2, y_2) \\ \begin{pmatrix} \textbf{tuned} \\ \textit{closed} \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ \textit{Stage}\,1 \\ \textit{Stage}\,2 \end{array} \begin{array}{c} (u_1, y_1) \\ \begin{pmatrix} \textbf{tuned} \\ \textit{open} \end{pmatrix} \end{array} \begin{array}{c} (u_2, y_2) \\ \begin{pmatrix} \textit{open} \\ \textbf{tuned} \end{pmatrix} \end{array}$$

(a) Correct Stage Definition. **(b)** Incorrect Stage Definition. **(c)** Independent Stage Definition.

Figure 10 Example Stage Definitions for Two Cascaded Tanks System.

6 Results and Discussion

In this section we empirically validate the proposed methodology and evaluate the I/O pairing and stage definition tasks for multiple configurations of the multi-tanks system, using performance metrics discussed in the previous section. All LLM experimental results were produced using the Claude-Sonnet-3.7 and Claude-Sonnet-4 LLMs, accessed via Anthropic API [1]. The temperature parameter for the API calls was set to 1.0, which promotes stochasticity and variety in the LLM outputs. The associated codebase and results can be found at `https://github.com/mjares/DX2025_LLMs_ExpertKnowledge.git`.

6.1 Input-Output Pairing

To evaluate the performance of the LLM-based methodology when proposing I/O pairings in a given system, we consider the benchmarks described in Section 5.2. Every input-system prompt pair was evaluated $N = 100$ times and percentage values were reported. Overall, we

Table 1 Analyzing performance for input-output pairing LLM responses.

Benchmark	Correct Format (%)		Accuracy (%)		**Accuracy/Format (%)**	
	Sonnet3.7	Sonnet4	Sonnet3.7	Sonnet4	**Sonnet3.7**	**Sonnet4**
(1): (3T, 1a2, 2a3)	100	100	100	100	**100**	**100**
(2): (4T, 1a2, 2a3, 3a4)	100	100	100	100	**100**	**100**
(3): (4T, 1b3, 2b4)[12]	100	100	100	98	**100**	**98**
(4): (4T, 1b3, 2b4)*	100	100	99	100	**99**	**100**
(5): 1T Temperature [12]	100	100	98	100	**98**	**100**

Table 2 Analyzing performance for stage definition LLM responses.

Benchmark	Correct Format (%)		Accuracy (%)		**Accuracy/Format (%)**	
	Sonnet3.7	Sonnet4	Sonnet3.7	Sonnet4	**Sonnet3.7**	**Sonnet4**
(1): (2T, 1a2)	98	100	98	99	**100**	**99**
(2): (2T, 1b2)	97	100	97	100	**100**	**100**
(3): (2T, 1-2ic)	100	100	100	100	**100**	**100**
(4): (3T, 1a2, 2a3)	93	100	93	100	**100**	**100**
(5): (3T, 1b3, 2a3)	96	100	96	100	**100**	**100**
(6): (3T, 1a3, 2-3ic)	96	100	94	98	**97.9**	**98**
(7): (4T, 1a2, 2a3, 3a4)	94	100	94	100	**100**	**100**

can see in Table 1 that the performance was high across all metrics, with 100% consistency in terms of formatting, which is a challenging task. Accuracy was also high at over 98% for all cases. This empirically shows that an LLM-based solution can yield expert knowledge recommendations regarding I/O pairing for benchmarks within the family of multi-tank systems. Furthermore, for the case of the benchmarks taken from [12], the I/O pairing matches the one achieved by the optimization approach with over 98% accuracy, while requiring less resources in terms of optimization problem formulation, expert interactions, and, potentially, computational cost (assuming inference cost from an LLM interaction is lower than the cost from solving the optimization problem).

6.2 Stage Definition

In order to evaluate the performance of the LLM-based methodology to define stages for a multistage control tuning framework, we consider the system configurations described in Section 5.2. Each input-system prompt pair was evaluated $N = 100$ times and percentage values were reported. Table 2 shows a summary of the performance. While formatting was not as consistent as the I/O pairing case, we can see a very high accuracy/formatting metric with 100% accuracy in most benchmarks. As discussed in Section 5.1.1, accuracy/formatting should be the priority metric, since incorrect formatting can be automatically detected, repeating the interaction with the LLM until a correctly formatted output is produced.

6.2.1 Comparing performance of tuned controllers for different stage definitions

In this section, we empirically compare the performance of tuned controllers described in Section 5.2. Controllers are tuned $N_R = 10$ times for every combination of system configuration, coupling level, and stage definition, and box-plot results over the 10 runs are

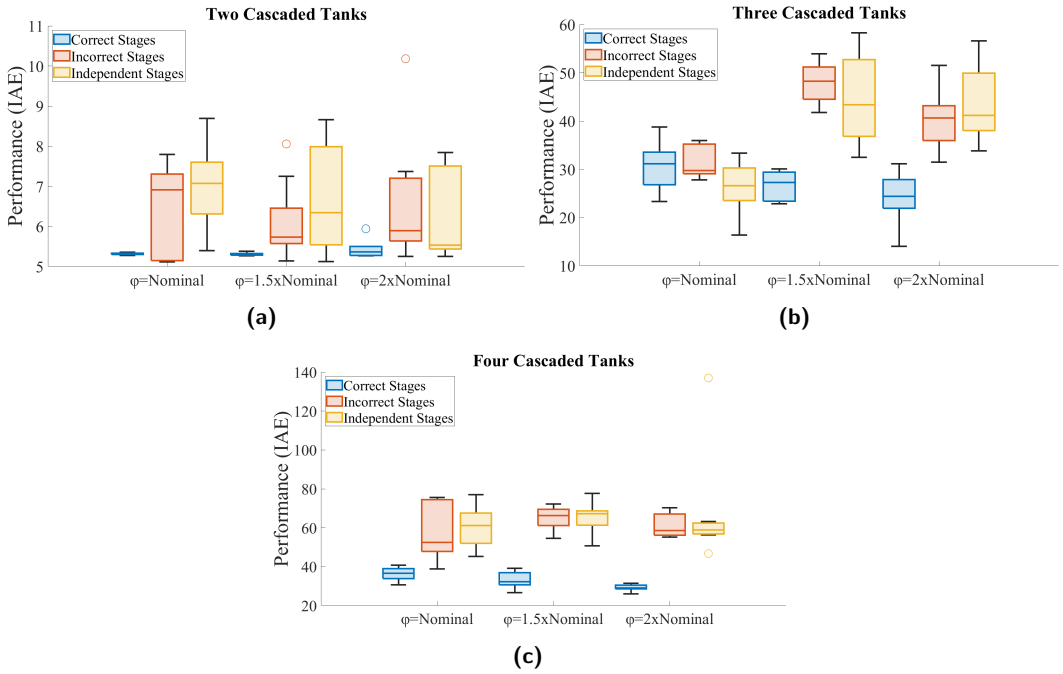


Figure 11 Comparing performance of tuned controllers for different stage definitions and coupling levels.

reported in Fig. 11. Performance is measured in integral absolute error (IAE), and lower error values are desired. Overall, we can see that, as the complexity of the system increases, i.e.: input-output space dimensions, number of components (tanks), etc., the impact of the stage definition on performance increases. In Fig. 11c, which is the most complex system evaluated (four cascaded tanks), there is a distinct difference between the performance achieved when following the correct stage definition and the incorrect stage definitions; however, for the lower complexity benchmarks (Figures 11a and 11b) the difference is not obvious, with the two cascaded tank system showing the largest overlap between stage definitions. Similarly, for every benchmark, an increase in coupling (represented by the φ parameter) is also associated with higher differences between the performances achieved under the correct and incorrect stage definitions; this is an expected result, as stated in Section 5.2. When we compare performance improvement for the three tank benchmark, we can see there is a significant performance improvement when the coupling in the system is high, but there is barely any difference between stage definitions when the coupling is lower. In general, the greatest improvement can be seen for the four cascaded tank benchmark at 2 times nominal coupling (Figure 11c) where a correct stage definition can reduce the error of the tuned controllers by 52% with respect to the incorrect stage definitions.

6.3 Discussion, Limitations, and Future Works

We have evaluated the proposed methodology for a variety of multi-tank benchmarks (Figures 7, 8, and 9), including a benchmark where temperature and level in a tank are controlled simultaneously (Fig. 9c), and a set of cascading tank benchmarks with different levels of coupling (Fig. 11). We have empirically shown that, for these benchmarks, stage definition can have significant impact on the performance of the tuned controllers.

Furthermore, we have shown that the proposed methodology for stage definition using LLMs can propose the correct stage definition and I/O pairing with a high degree of consistency and accuracy.

We acknowledge that, even though the actuator label space ($L^A(e)$) can integrate notions of temperature control, as well as other potential extensions (e.g.: liquid composition/concentration), the proposed methodology is mostly centered around the problem of level control in a family of systems consisting of reservoirs, pumps, and connections. In spite of this limitation, this is a first step towards a system-informed general methodology for prompt engineering with the purpose of expert knowledge automation. An extension of these results to a wider variety of benchmarks would require a redefinition of the mathematical language and the prompt engineering algorithm; however, the core elements of the methodology would persist: a graph representation of the system components and interactions, and a prompt engineering algorithm that binds this graph language to the input prompt space of an LLM. Future works should focus on proposing a generalized mathematical language and prompt engineering algorithm, or, if this is not possible, a set of general guidelines for developing these graph representations and prompt engineering algorithms for different families of benchmarks.

We have proposed a simple topology-based graph representation of the system since we are aiming for a solution that requires minimal expert interaction and knowledge of the system, which would be a limitation for more complex system representations such as bond graphs, process graphs, or structural analysis. However, a generalization of this methodology for systems other than the multi-tank benchmarks will most likely require more complex representations of the system, such as the ones mentioned above.

We have focused on systems with centralized control since I/O pairing and stage definition for sequential controller tuning would rarely be relevant in a centralized control approach. Furthermore, decentralized parametric controllers, or a hybrid between decentralized and supervisory control are industry standard solutions. However, future works should focus on extending the methodology to consider systems with centralized/supervisory control, as well as systems with integrated decoupling strategies.

This work is also limited by the lack of formal definition of what is a correct I/O pairing or stage definition apart from what is referenced as expert knowledge in the associated literature. Therefore, future works will provide a formal framework to measure validity of expert knowledge in terms of I/O pairing and stage definition.

7 Conclusions

This paper addresses controller-tuning aspects of automated engineering system design. We proposed a mathematical language to describe a family of multi-tank benchmarks based on system topology. We implemented an algorithm to bind this mathematical language to the input prompt space of a Large Language Model as a form of prompt engineering. Following this methodology, we automated expert knowledge requirements in decentralized controller auto-tuning related to I/O pairing and stage definition for a multistage tuning framework. Our proposed methodology showed an accuracy of over 97% for both I/O pairing and stage definition, with consistent output formatting. Furthermore, we empirically evaluated the performance of tuned controllers for different stage definitions in a family of multi-tank benchmarks and showed that, depending on the complexity of the system, and degree of coupling between control loops, a correct stage definition can improve performance over the incorrect stage definitions by up to 52%. Future works will extend the mathematical language and the prompt engineering algorithm to a wider variety of benchmarks.

References

1 Anthropic api. URL: https://console.anthropic.com.

2 Marlon J. Ares-Milian, Gregory Provan, and Marcos Quinones-Grueiro. Towards automated controller parameter design in cyber-physical systems: Improving computational cost. In *2025 IEEE International Conference on Smart Computing, SMARTCOMP 2025*, 2025.

3 Aman Bhargava, Cameron Witkowski, Shi-Zhuo Looi, and Matt Thomson. What's the magic word? a control theory of llm prompting, October 2023. doi:10.48550/arXiv.2310.04444.

4 B. Ould Bouamama, G. Biswas, R. Loureiro, and R. Merzouki. Graphical methods for diagnosis of dynamic systems: Review, 2014. doi:10.1016/j.arcontrol.2014.09.004.

5 Tom B Brown, Benjamin Mann, Nick Ryder, and Melanie Subbiah et. al. Language models are few-shot learners. In *NeurIPS 2020*, 2020.

6 Y. Chen, R. Goebel, G. Lin, B. Su, Y. Xu, and A. Zhang. An improved approximation algorithm for the minimum 3-path partition problem. *Journal of Combinatorial Optimization*, 38:150–164, 2019. doi:10.1007/s10878-018-00372-z.

7 Xingang Guo, Darioush Keivan, Usman Syed, Lianhui Qin, Huan Zhang, Geir Dullerud, Peter Seiler, and Bin Hu. Controlagent: Automating control system design via novel integration of llm agents and domain expertise. *arXiv preprint arXiv:2410.19811*, 2024. doi:10.48550/arXiv.2410.19811.

8 P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, and S. Achiche. Design, modelling, simulation and integration of cyber physical systems: Methods and applications. *Computers in Industry*, 82:273–289, October 2016. doi:10.1016/j.compind.2016.05.006.

9 H J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 1964.

10 Jiang Lu, Pinghua Gong, Jieping Ye, Jianwei Zhang, and Changshui Zhang. A survey on machine learning from few samples, September 2020. arXiv:2009.02653.

11 Yifan Luo, Yiming Tang, Chengfeng Shen, Zhenan Zhou null, and Bin Dong. Prompt engineering through the lens of optimal control. *Journal of Machine Learning*, 2:241–258, January 2023. doi:10.4208/jml.231023.

12 Vasileios K. Mappas, Vassilios S. Vassiliadis, Bogdan Dorneanu, Alexander F. Routh, and Harvey Arellano-Garcia. Automated control loop selection via multistage optimal control formulation and nonlinear programming. *Chemical Engineering Research and Design*, 195:76–95, July 2023. doi:10.1016/j.cherd.2023.05.041.

13 Sammyak Mate, Pawankumar Pal, Anshumali Jaiswal, and Sharad Bhartiya. Simultaneous tuning of multiple pid controllers for multivariable systems using deep reinforcement learning. *Digital Chemical Engineering*, 9, December 2023. doi:10.1016/j.dche.2023.100131.

14 Silke Merkelbach, Alexander Diedrich, Anna Sztyber-Betley, Louise Travé-Massuyès, Elodie Chanthery, Oliver Niggemann, and Roman Dumitrescu. Using multi-modal llms to create models for fault diagnosis. In *The 35th International Conference on Principles of Diagnosis and Resilient Systems (DX'24)*, volume 125, November 2024. doi:10.4230/OASIcs.DX.2024.6.

15 C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. USA:MIT Press, January 2006.

16 Alejandro Rodríguez-Molina, Efrén Mezura-Montes, Miguel G. Villarreal-Cervantes, and Mario Aldape-Pérez. Multi-objective meta-heuristic optimization in intelligent control: A survey on the controller tuning problem. *Applied Soft Computing Journal*, 93, August 2020.

17 D. Stenger, M. Nitsch, and D. Abel. Joint constrained bayesian optimization of planning, guidance, control, and state estimation of an autonomous underwater vehicle. In *ECC 2022*, 2022.

18 Prerit Terway, Kenza Hamidouche, and Niraj K. Jha. Dispatch: Design space exploration of cyber-physical systems, September 2020. arXiv:2009.10214.

19 K. Tohma, H. İ. Okur, H. Gürsoy-Demir, M. N. Aydn, and C. Yeroğlu. Smartcontrol: Interactive pid controller design powered by llm agents and control system expertise. *SoftwareX*, 31:102194, 2025. doi:10.1016/J.SOFTX.2025.102194.

20 Collin Zhang, John X. Morris, and Vitaly Shmatikov. Extracting prompts by inverting llm outputs, May 2024. doi:10.48550/arXiv.2405.15012.

A Data-Driven Particle Filter Approach for System-Level Prediction of Remaining Useful Life

Abel Diaz-Gonzalez
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Austin Coursey
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Marcos Quinones-Grueiro
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Gautam Biswas
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Abstract

Accurate estimation of the remaining useful life (RUL) of industrial systems is a critical component of predictive maintenance strategies. This work presents a data-driven method for RUL prediction that also quantifies uncertainty, drawing inspiration from model-based particle filtering techniques. Instead of simulating system state transitions, we model degradation as a stochastic process governed by performance metrics and use a Bayesian particle filtering framework to infer its underlying parameters. Our approach bypasses traditional state-space modeling by directly estimating the end-of-life distribution from observed performance data. Key characteristics of the filter, such as propagation noise and observation correction strength, are adapted over time based on current observations and past predictive performance, enabling better capture of future uncertainty. We evaluate the proposed method using an unmanned aerial vehicle simulation dataset developed for system-level prognostics research, which includes high-fidelity degradation signals and ground-truth system performance metrics for validating predictive accuracy.

2012 ACM Subject Classification Applied computing → Aerospace; Computing methodologies → Artificial intelligence; Computing methodologies → Machine learning approaches

Keywords and phrases remaining useful life, particle filter methods, data-driven methods, system-level prognostics, performance metrics

Digital Object Identifier 10.4230/OASIcs.DX.2025.11

Funding *Abel Diaz-Gonzalez*: NASA University Leadership Initiative (ULI)

1 Introduction

Industrial systems inevitably degrade over time, posing a risk of failure and unexpected downtime. Accurate estimation of a system's remaining useful life (RUL) is essential for implementing predictive maintenance strategies that improve operational efficiency, reduce unplanned outages, and lower life-cycle costs [12].

Prognostic techniques for RUL estimation are typically classified into model-based, data-driven, and hybrid approaches [9]. Model-based techniques leverage physical principles to construct degradation models and can yield interpretable and reliable predictions when the underlying models are accurate and well-calibrated. However, they require extensive domain knowledge and are often impractical for systems with complex or poorly understood dynamics [6]. In contrast, data-driven methods, including deep neural networks, learn degradation patterns directly from sensor data and are well-suited for modern applications where large volumes of operational data are available [10, 5]. Despite their success, these methods often lack interpretability and provide limited tools for quantifying predictive uncertainty, which is critical for risk-aware decision-making. Hybrid approaches combine

elements of both paradigms to exploit the complementary advantages of physics-based modeling and data-driven learning. By incorporating domain knowledge into data-driven frameworks, hybrid methods can improve prediction accuracy, robustness, and extrapolation capability. However, they also inherit the limitations of both families of techniques, such as the modeling cost of physics-based approaches and the data requirements of data-driven methods, which makes their implementation challenging in practice.

In this paper, we introduce a novel prognostic framework that adapts the particle filter (PF) methodology to a data-driven setting. In traditional model-based prognostics, PFs are used to estimate the hidden state of a system over time by recursively applying a known or learned state transition model. However, in data-driven applications, these models must be inferred from data, and even small errors in the learned dynamics can compound over time, degrading predictive accuracy. To avoid this issue, our method does not rely on recursive state estimation. Instead, it directly models the distribution of the performance metrics that define the system's end of life (EOL). We assume that these performance values are available both online and in historical run-to-failure datasets. How they are obtained from raw system observations, whether through direct measurement or a separate modeling stage, is outside the scope of this work. Given these inputs, we represent degradation as a stochastic process indexed by performance level rather than time, allowing for flexible and interpretable modeling of failure progression. To demonstrate the method, we use an extension of the Gamma process proposed by [11], which models the distribution of time required to reach a given performance value.

The parameters of this process, which govern the system's degradation behavior, are treated as latent variables within a Bayesian framework and are inferred from the observed performance values using particle filtering. Our method builds on the probabilistic foundation of particle filtering but repurposes it to estimate the EOL distribution directly from past and current observations. This formulation captures both stochastic uncertainty, arising from inherent randomness in the degradation process, and epistemic uncertainty, due to limited knowledge about future evolution. As the system progresses, particle weights are updated through a modified correction step that remains faithful to the particle filtering framework but accounts for future uncertainty by adjusting the "strength" of the correction. Observations made early in the degradation timeline, when future outcomes are still highly uncertain, have a weaker influence on particle weights, while those made closer to failure carry more weight. Both the propagation noise, used to sample hypothetical values of the latent process parameters, and the correction strength are learned from historical run-to-failure data, enabling a data-driven and theoretically grounded approach to modeling uncertainty across the degradation timeline.

The method can be applied in parallel to multiple performance metrics that reflect different aspects of system degradation. For each metric, we estimate a dedicated EOL distribution based on a predefined threshold. The final RUL prediction for the system is then obtained using standard aggregation rules. In this work, we select the earliest predicted failure time across all metrics.

The main contributions of this work are as follows:

- A data-driven prognostic framework inspired by model-based filtering, which directly estimates RUL and its uncertainty from performance metrics without relying on system models or expert knowledge.
- A learning-based particle filtering scheme that infers degradation behavior and adapts uncertainty handling from historical run-to-failure data, offering improved interpretability over black-box methods.

We evaluate our approach using a custom dataset generated in Simulink, which provides ground-truth system-level performance metrics under run-to-failure conditions. By modeling degradation as a stochastic process indexed by performance level and treating its governing parameters as latent variables, our method uses particle filtering not for state tracking, but for sequential inference of these degradation parameters. The resulting EOL predictions are expressed as probabilistic mixtures over learned process trajectories, providing both interpretable and uncertainty-aware RUL estimates. This formulation offers a principled alternative to black-box models, with improved transparency and statistical grounding.

2 EOL Particle Filter

Performance metrics used to define the EOL process typically exhibit a monotonic trend [13]; performance degrades over time, and does not recover except following maintenance. We assume this monotonicity in our analysis and treat noise-induced fluctuations as minor deviations. We then linearly scale these monotonic metrics so that the performance level s ranges from 1 (fully healthy) to 0 (the EOL threshold).

Consider the stochastic process $\{T_s\}_{s\in[0,1]}$ that models the time at which the scaled performance metric reaches the level s. Notice that the distribution of the time to performance metric EOL is given by the random variable T_0. To model unit-specific degradation behaviour, we introduce latent degradation parameters X that govern the evolution of the stochastic process T_s. The parameters X capture intrinsic degradation characteristics that remain fixed for each unit over its lifetime. Different units may follow distinct degradation trajectories, and this unit-to-unit variability is encoded through variation across their corresponding latent parameters X. In practice, the true values of the parameters X are unknown and must be inferred from data. Once the true parameter vector is identified, i.e., $X = x^*$, the process $\{T_s(x^*)\}_{s\in[0,1]}$ defines a distribution over the times at which each performance level s is reached. The randomness inherent in $T_s(x^*)$ captures stochastic uncertainty in the degradation process.

Accurately estimating x^* typically requires a dense sequence of observations spanning the full performance range. However, such data is only available at the end of a unit's life, when predictions are no longer needed. For online predictions prior to failure, it is crucial to account for the uncertainty in the estimation of x^* due to missing future observations. To address this, we adopt a Bayesian framework in which the latent degradation parameters X are treated as latent random variables with a prior distribution that is updated over time as new observations are collected. At each time step k, we maintain a posterior approximation denoted X_k, allowing us to infer the likely values of x^* as additional measurements y_i become available. The stochasticity of X_k captures epistemic uncertainty arising from incomplete information. We employ particle filtering to perform this sequential Bayesian inference in real time as the system evolves.

2.1 Particle Filters

PFs are a class of Bayesian filters that use sequential Monte Carlo sampling to approximate the posterior distribution of latent variables with a set of weighted samples or "particles" [7, 14]. While classical filters such as the Kalman Filter are optimal under linear and Gaussian assumptions, PFs offer a more flexible and nonparametric alternative that performs well in nonlinear and non-Gaussian settings. In our work, we adapt the particle filtering framework to sequentially infer the latent degradation parameter X as more observations become available. This allows us to capture both epistemic and stochastic uncertainty in EOL predictions. Below we present the details of this degradation model and the inference procedure.

Bayesian filtering provides a principled framework for sequentially updating our knowledge about a sequence of latent variables as new indirect observations of these variables arrive. At time step k, the goal is to estimate the posterior distribution of the latent variable X_k given all observations $y_{0:k}$ up to current time k. This posterior is often referred to as the belief distribution and is computed in two recursive steps:

$$\begin{aligned} bel_k(x_k) &:= p(X_k = x_k \mid Y_{0:k-1} = y_{0:k-1}) \\ &= \int p(x_k \mid x_{k-1})\, \overline{bel}_{k-1}(x_{k-1})\, dx_{k-1} \quad \text{(prediction)} \end{aligned} \tag{1}$$

$$\overline{bel}_k(x_k) := p(X_k = x_k \mid Y_{0:k} = y_{0:k}) \propto p(y_k \mid x_k)\, bel_k(x_k) \quad \text{(correction)} \tag{2}$$

The belief distribution $\overline{bel}_k$ thus captures the current probabilistic estimate of X_k. Notice that the particle filtering framework depends only on the transition, sensor models:

$$\begin{aligned} p(x_k \mid x_{k-1}) &:= p(X_k = x_k \mid X_{k-1} = x_{k-1}) \quad \text{(transition model)}, \\ p(y_k \mid x_k) &:= p(Y_k = y_k \mid X_k = x_k) \quad \text{(sensor model)}. \end{aligned}$$

and the initial distribution of the latent variable $bel_0(x) = p(X_0 = x)$.

In our setting, the latent process parameters are static, so ideally we have $X_k = X$ for all $k \geq 1$. However, similar to standard particle filtering, we adopt a pseudo-transition model with Gaussian propagation noise to maintain particle diversity and mitigate degeneracy.

Each observation $y_k = (t_k, s_k)$ provides an indirect measurement of the underlying degradation parameters X through the stochastic process T_s. This leads to the following sensor and transition models:

$$\begin{aligned} p(y_k \mid x_k) &= P(T_{s_k} = t_k \mid X_k = x_k), \\ p(x_k \mid x_{k-1}) &= \mathcal{N}(x_{k-1}, \Sigma), \end{aligned} \tag{3}$$

where the transition model injects artificial Gaussian noise centered at the previous state. As usual in particle filtering, for simplification, we assume the covariance matrix Σ to be diagonal, which corresponds to injecting independent noise into each state component. Its diagonal entries control the strength of the perturbation, allowing the filter to explore broader regions of the parameter space when uncertainty is high, and to concentrate the posterior distribution as more informative data become available.

The algorithm then follows the standard bootstrap PF framework [7, 8], with a modified correction step tailored for parameter inference. This modification arises from the fact that, unlike in standard particle filtering, not all observations provide the same level of insight into the degradation parameters. Early in the system's life, observations tend to be less informative, as much of the uncertainty lies in the unobserved future. Consequently, particles that appear unlikely based on current data may still correspond to plausible future scenarios and should not be prematurely discarded. Conversely, near the EOL, observations become more informative and should exert greater influence on particle selection.

To account for this, we introduce a tempering parameter $\kappa > 0$, referred to as the observation correction strength, which modulates the influence of the sensor model. We temper the likelihood function by raising it to the power κ, yielding tempered importance weights:

$$\tilde{w}_k^{(i)} = P(T_{s_k} = t_k \mid X_k = x_k^{(i)})^{\kappa}. \tag{4}$$

This approach follows the power posterior formulation in Bayesian inference, where the likelihood is scaled to adjust its impact on the posterior. When $\kappa < 1$, the update is softened to preserve particle diversity under high uncertainty. When $\kappa > 1$, the correction becomes sharper and more selective. The standard PF corresponds to the special case $\kappa = 1$.

In our framework, both the observation correction strength κ and the variances in the diagonal matrix Σ are learned offline from historical data as a function of the observation. Offline data is also used to construct the initial distribution $p(X_0)$. Specifically, we obtain a parameter estimate $x_0^{(i)}$ for each training unit by maximizing the likelihood of its run-to-failure data, for $i = 1, 2, \ldots, n$. These estimates are then augmented with Gaussian noise to generate the desired number of particles N, which approximates the initial distribution $p(X_0)$ as:

$$p(X_0 = x) \approx \frac{1}{N} \sum_{i=1}^{N} \delta_{x_0^{(i)}}(x), \tag{5}$$

where δ_z is the Dirac measure centered at z. This corresponds to a standard PF initialization, where all particles are equally weighted. The variance of the Gaussian noise used for augmentation is treated as a hyperparameter of the algorithm.

Then the RUL prediction PF we propose can be written as:

1. *Initialization*, $k = 0$.
   - For $i = 1, \ldots, N$, we consider $x_0^{(i)}$ and set $k = 1$.
2. *Prediction step*
   - For $i = 1, \ldots, N$, sample $\tilde{x}_k^{(i)} \sim \mathcal{N}(x_{k-1}^{(i)}, \Sigma)$
3. Correction step
   - For $i = 1, \ldots, N$, evaluate the importance weights:

     $$\tilde{w}_k^{(i)} = p(y_k \mid \tilde{x}_k^{(i)})^{\kappa}.$$

   - Normalise the importance weights.

     $$w_k^{(i)} = \frac{\tilde{w}_k^{(i)}}{\sum_{j=1}^{N} \tilde{w}_k^{(j)}}.$$

4. *Selection step*
   - Resample with replacement N particles $\{x_k^{(i)}\}_{i=1}^{N}$ from the set $\{\tilde{x}_k^{(i)}\}_{i=1}^{N}$ according to the importance weights $\{w_k^{(i)}\}_{i=1}^{N}$.
5. Set $k \leftarrow k + 1$ and go to step 2.

2.2 Predictive distribution

At each time step k, particle filtering maintains an empirical approximation $\overline{bel}_k(x)$ of the posterior distribution over the latent degradation parameters. This distribution represents the current belief about the true degradation parameter X. In practice, PFs approximate this belief using a discrete posterior distribution composed of Dirac delta functions centered at the particles:

$$\overline{bel}_k(x) \approx \sum_{i=1}^{N} w_k^{(i)} \, \delta_{x_k^{(i)}}(x),$$

where $w_k^{(i)}$ is the importance weights associated with the particle $x_k^{(i)}$.

We use this empirical belief distribution to compute the predictive probability distribution of T_s, marginalizing over the posterior distribution of X_k:

$$\begin{aligned} P(T_s = t \mid y_{0:k}) &= \int P(t \mid x_k) P(x_k \mid y_{0:k})\, dx_k \\ &= \int P(t \mid x_k)\, \overline{bel}_k(x_k)\, dx_k \\ &\approx \sum_{i=1}^{N} w_k^{(i)}\, P(T_s(X_k) = t \mid X_k = x_k^{(i)}) \\ &= \sum_{i=1}^{N} w_k^{(i)}\, P(T_s(x_k^{(i)}) = t). \end{aligned} \tag{6}$$

The result is a weighted mixture distribution, where each component corresponds to the performance metric stochastic process where the degradation parameter is the particle, that is, $T_s(x_k^{(i)})$ for each particle $x_k^{(i)}$. This can be interpreted as:

- Each particle $x_k^{(i)}$ represents a distinct hypothesis about the unit's degradation behavior.
- Its weight $w_k^{(i)}$ reflects the current confidence in that hypothesis.

2.3 EOL prediction

In particular, the EOL predictive distribution at time step k can be approximated by

$$P(T_0 = t \mid y_{0:k}) \approx \sum_{i=1}^{N} w_k^{(i)}\, P(T_0(x_k^{(i)}) = t). \tag{7}$$

From the distribution of the stochastic process $\{T_s\}_{s\in[0,1]}$, we compute the EOL prediction at time k as the mean of the EOL predictive distribution:

$$\widehat{\mathrm{EOL}}_k := \mathbb{E}[T_0 \mid y_{0:k}] = \sum_{i=1}^{N} w_k^{(i)}\, \mathbb{E}[T_0(x_k^{(i)})]. \tag{8}$$

In general, the distribution $P(T_0 = t \mid y_{0:k})$ does not have a closed-form solution, but we can use any numerical method to approximate confidence intervals. To compute a confidence interval at level ℓ, we find the quantiles a and b such that

$$\begin{aligned} P(T_0 < a) &= \frac{1-\ell}{2}, \\ P(T_0 < b) &= \frac{1+\ell}{2}, \end{aligned}$$

so that $P(a < T_0 < b) = \ell$.

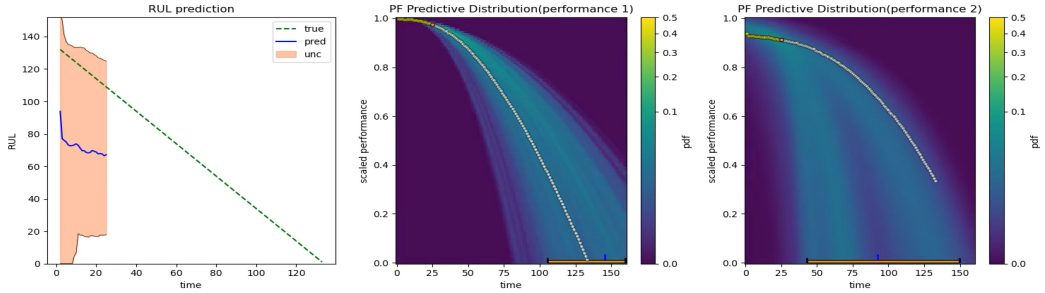
Finally, the EOL prediction and confidence interval at time step k of the system are computed by applying any aggregation of the corresponding performance predictions. In this work, we use the earliest among the estimated quantities (i.e., mean and quantiles) across all performance metrics as the final prediction and uncertainty bounds.

Figure 1 provides a visual overview of the proposed RUL prediction method at three different time points. Each row corresponds to a distinct stage in the system's lifecycle.

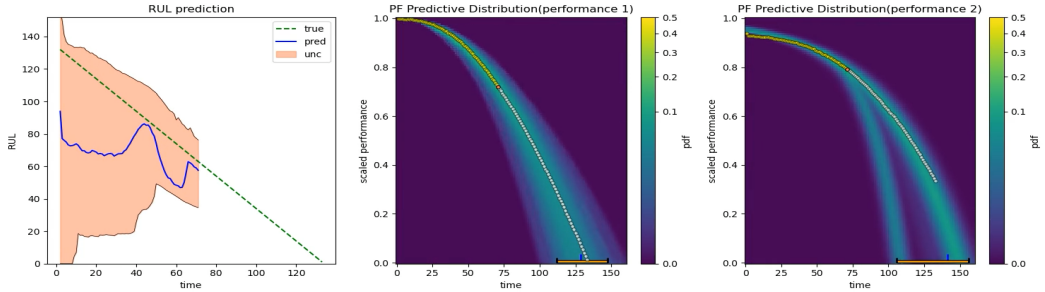
The left panel in each row displays the predicted RUL of the system, along with a 95% confidence interval computed from the empirical particle distribution. As the system approaches the end of its operational life, the uncertainty decreases and the predictions

become more confident. The green dashed line indicates the ground-truth RUL, while the solid blue line and shaded salmon region represent the predicted RUL and its confidence interval, respectively.

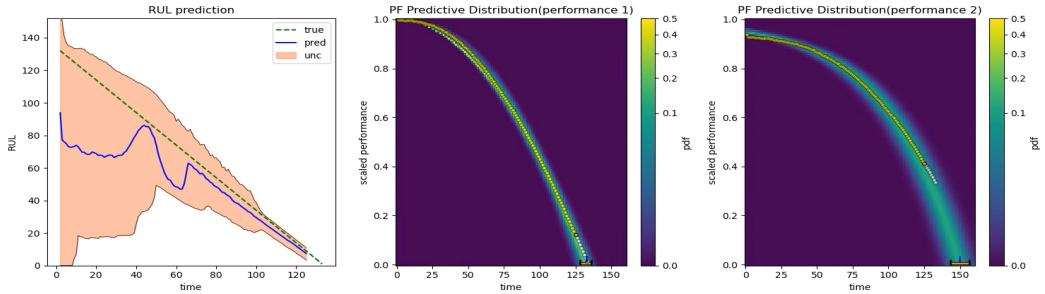
The center and right panels depict the predictive probability densities of the stochastic process $\{T_s\}_{s\in[0,1]}$, each corresponding to a different performance metric, and are based on their respective current PF estimate of the latent degradation parameters X. Yellow, orange, and white points indicate past, current, and future observations, respectively. The horizontal orange bars at the bottom mark the 95% confidence interval of the predicted end-of-life distributions at the current time, and the vertical blue line denotes the predicted RUL. Both are aggregated to produce the RUL prediction and confidence interval shown in the left panel. A brighter point (t, s) in the plot indicates a higher likelihood of reaching performance level s at time t. At the second time point, for example, a bimodal predictive distribution emerges, suggesting multiple plausible degradation trajectories consistent with the current particle estimates.



(a) Visualization at an early time step.



(b) Visualization at a mid-lifecycle time step.



(c) Visualization near EOL.

Figure 1 Evolution of RUL prediction and predictive probability densities over three stages of a system's life. Each row corresponds to a different time step during the run-to-failure simulation.

3 Performance Model

Following [11], we use the commonly encountered Gamma family of end-of-life distributions to illustrate our method and present the results. While that work also considers the Weibull distribution, we focus on the Gamma distribution for simplicity. Another key difference is that their model is applied to the health index that aggregates all performance metrics and units distributions, whereas we use it to capture the stochasticity of individual unit performance values separately.

We extend the parametric model presented in equation (11) of [11] to increase its flexibility. In the original formulation, all performance trajectories begin at the fully healthy state $s = 1$, and the initial variance of the process (at $t = 0$) is zero. To relax these constraints, we introduce two additional parameters: α, which decouples the initial performance value from $s = 1$, and μ, which allows the model to capture non-zero variance at early stages.

We define the performance degradation model as:

$$\mathbf{s}(t) = \alpha\left(1 - [b(t-\mu)]^{\rho}\right), \quad b \sim \text{InvGamma}(\beta, \lambda), \tag{9}$$

where $\mathbf{s}(t)$ represents the performance level at time t, and b is a random variable following an inverse Gamma distribution.

The parameters of the model are summarized as:

- $\alpha > 0$: controls the initial performance level,
- $\rho > 0$: controls the curvature or steepness of the degradation curve,
- $\mu \leq 0$: shifts the gamma function from zero to a negative value allowing the degradation to not start with zero variance,
- $\beta, \lambda > 0$: shape and rate (inverse scale) parameters of the inverse Gamma distribution.

From this performance model, and following a procedure similar to [11], the distribution of the time T_s to reach a given performance level s can be derived as follows:

$$\begin{aligned} P(T_s > t) &= P(\mathbf{s}(t) > s) = P\left(\alpha\left(1 - [b(t-\mu)]^{\rho}\right) > s\right) \\ &= P\left([b(t-\mu)]^{\rho} < 1 - \frac{s}{\alpha}\right) = P\left(b < \frac{(1-\frac{s}{\alpha})^{1/\rho}}{t-\mu}\right) \\ &= P\left(\frac{1}{b} > \frac{t-\mu}{(1-\frac{s}{\alpha})^{1/\rho}}\right). \end{aligned}$$

Since $b \sim \text{InvGamma}(\beta, \lambda)$, we have $\frac{1}{b} \sim \text{Gamma}(\beta, \lambda)$. Therefore,

$$P(T_s < t) = P\left(\frac{1}{b} < \frac{t-\mu}{(1-\frac{s}{\alpha})^{1/\rho}}\right) = \frac{1}{\Gamma(\beta)}\gamma\left(\beta, \lambda\frac{t-\mu}{(1-\frac{s}{\alpha})^{1/\rho}}\right),$$

where γ denotes the lower incomplete Gamma function. Hence, the stochastic process T_s representing the time to reach performance level s follows the shifted Gamma distribution:

$$T_s \sim \text{Gamma}\left(\beta, \lambda_s, \mu\right), \quad \text{where} \quad \lambda_s = \frac{\lambda}{(1-\frac{s}{\alpha})^{1/\rho}}.$$

In particular, the EOL distribution corresponds to $s = 0$, is given by:

$$T_0 \sim \text{Gamma}\left(\beta, \lambda, \mu\right).$$

Finally, we collect all the parameters of the degradation model into a single vector:

$$X = (\beta, \lambda, \rho, \mu, \alpha),$$

which defines the latent degradation parameters of the stochastic process. The PF is then used to infer the value of X that best explains the observed data.

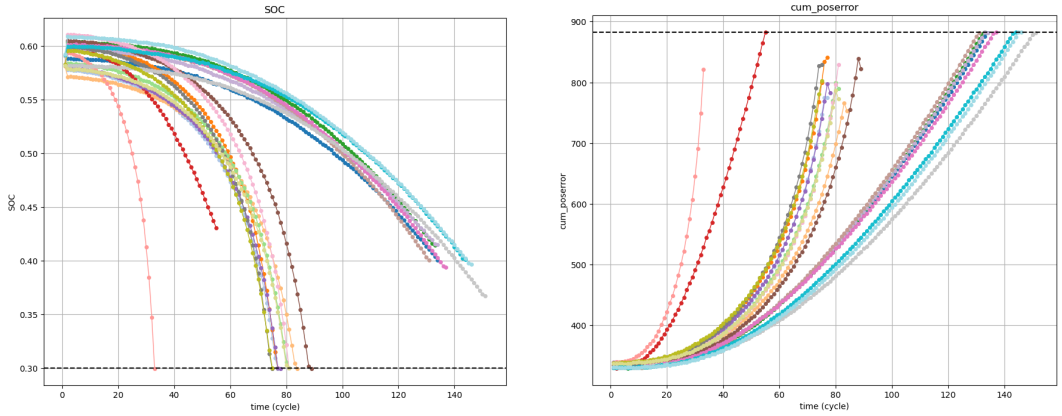
3.1 Dataset

To generate realistic degradation data for evaluating our proposed method, we used a MATLAB implementation of the octo-rotor simulation model introduced in [1]. This high-fidelity framework emulates a fault-tolerant unmanned aerial vehicle (UAV) system by modeling detailed dynamics, control mechanisms, and component degradation under realistic flight conditions. The simulator incorporates empirically validated degradation behaviors for key components such as motor internal resistance, battery capacity, and internal resistance, capturing both component aging and environmental interactions.

Using this platform and the original degradation profiles, we created a custom dataset aligned with our experimental goals. Each UAV is modeled with nine degrading components: eight motors (each with degrading internal resistance) and one battery (modeled with degrading charge capacity and internal resistance). We simulated run-to-failure trajectories for 20 UAVs, each performing repeated missions along a square 1km flight path at a fixed altitude of 50meters. The missions include full take-off and landing cycles and are conducted under a constant 10 m/s wind disturbance in the negative x-direction. This simulation environment provides a controlled yet complex test bed for validating prognostic methods under realistic operational and degradation conditions.

Two performance metrics were monitored throughout each UAV's run-to-failure simulation: a component-level metric, the battery state of charge (SOC), and a system-level metric, the cumulative position error over the flight trajectory. The position error was computed as the accumulated distance between the UAV's measured position and its projection onto the intended straight-line path between consecutive waypoints. After each flight, these metrics were recorded, and the simulation continued until either metric exceeded a predefined failure threshold. Specifically, the failure threshold for SOC was set to 0.3, while the threshold for cumulative position error was defined as the equivalent of 1 meter per second of expected flight time.

Figure 2 illustrates the evolution of both SOC and cumulative position error across the 20 simulated UAVs, with dashed lines indicating the failure thresholds for each metric.



(a) Battery SOC. The dashed black line marks the failure threshold at 0.3.

(b) Cumulative position error. The dashed black line indicates the failure threshold of 884 meters.

Figure 2 Evolution of system-level performance metrics for 20 UAVs throughout their run-to-failure simulations.

As a post-processing step, the final flight in which failure occurred was removed from each trajectory, and both performance metrics were normalized to the interval $[0, 1]$. The maximum healthy value observed across the dataset was mapped to 1, and the respective

failure threshold to 0. Among the 20 UAVs, failure was triggered by the cumulative position error in 9 cases, and by SOC depletion in 11 cases. We randomly selected 15 UAVs for training and 5 for testing.

4 Results

We applied our method described in Section 2 using the degradation model described in Section 3 to our generated dataset described in Section 3.1. Then we have our latent space $X = (\beta, \lambda, \rho, \mu, \alpha)$, defined by the degradation model (9).

4.1 Training

The first step of our method is to estimate the initial distribution of the EOL particle filter, denoted $P(X_0)$. To do this, we learn an individual parameter vector $x_0^{(i)}$ for each training unit $i = 1, 2, \ldots, n$ by maximizing the likelihood of its degradation data and using the performance degradation model defined in equation (9). In our dataset, the number of training units is $n = 15$. So the initial estimation of the latent distribution $P(X_0)$ is given by equation (5) for the learned values $x_0^{(i)}$, $i = 1, 2, \ldots, 15$.

Figure 3 shows the predictive distribution of the particle filter deduced in equation (6),

$$P(T_s(X_0) = t) \approx \frac{1}{N} \sum_{i=1}^{N} P(T_s(x_0^{(i)}) = t), \quad s \in [0, 1], \quad t > 0$$

for the initial step $k = 0$ and without considering the augmentation step (i.e. $N = n$). We include the augmentation step in the next subsection because the augmentation noise variance is a hyperparameter.

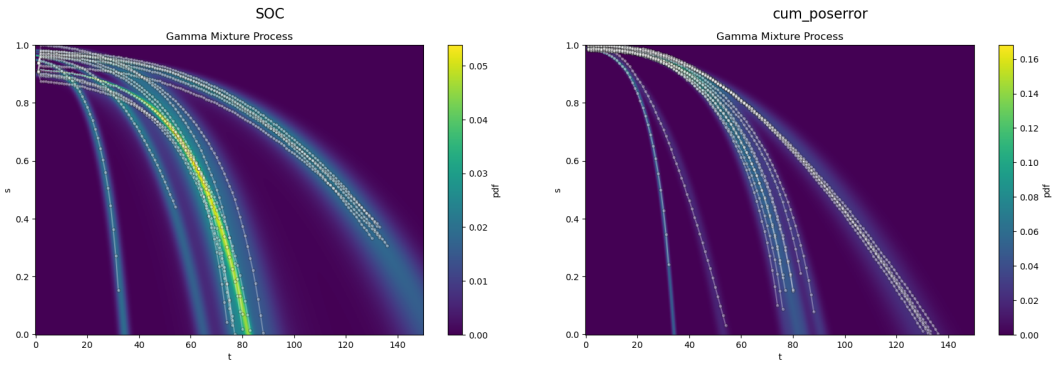


Figure 3 Initial prediction probability density functions for the degradation gamma mixture model learned from training data, shown for both performance metrics. These define the initial belief distribution of the particle filter, $bel_0(x) = p(X_0 = x)$. Circled points denote the training data. For illustration purposes, we have not included the augmentation step, so the number of particles N is equal to the number of training units n. At this stage, no observations have been received and the distribution is based solely on the particles learned from data.

4.2 Evaluation

To learn the correction strength and the variance of the PF propagation noise, we learned a function f that maps the current observation to these six parameters: κ (as defined in equation (4)) and the diagonal entries of Σ as defined in equation (3)) corresponding to the

noise of the degradation model parameters $(\beta, \lambda, \rho, \mu, \alpha)$. The function is linear, except for a softplus activation applied to the output to ensure positivity. Since the observation is two-dimensional, $y_k = (t_k, s_k)$, and we included bias terms, we obtained a total of $3 \times 6 = 18$ hyperparameters for learning this function. An additional hyperparameter is used for the variance of the augmentation noise employed to build the initial distribution, resulting in a total of 19 hyperparameters.

We used Optuna [2] for hyperparameter optimization, employing its implementation of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [4], which is well-suited for efficiently exploring complex and non-convex search spaces. We applied leave-one-out cross-validation. In each fold, the initial distribution was estimated using 14 out of the 15 training units, and the hyperparameters were optimized by evaluating performance on the remaining unit. For the experiments, we set the number of particles at $n_{\text{particles}} = 3010$ (we added 10 to make the number of particles a multiple of the 14 training unit fold). We applied our EOL particle filter method independently to both performance metrics: SOC and cumulative position error and at each time step, we evaluated the predictive accuracy using the log-likelihood of future observations under the predictive distribution of the EOL particle filter, as defined in equation (6).

4.3 RUL Prediction

To assess performance, we used three standard prognostic metrics: one for RUL prediction quality (root mean square error, RMSE) and two for uncertainty estimation (prediction interval coverage probability, PICP, and prediction interval normalized width, PINAW). RMSE captures overall prediction accuracy; PICP measures how often the ground-truth RUL falls within the prediction interval; and PINAW captures how tight the interval is, normalized over the range of RUL values.

Figure 4 shows the predicted RUL trajectories along with 95% confidence intervals for both predictive and uncertainty variance across the 5 test UAVs. The experiment was repeated 10 times to assess statistical robustness. The model effectively captures degradation trends and represents both aleatoric and epistemic uncertainty. As the system approaches EOL, the uncertainty naturally decreases due to the accumulation of informative observations. Table 1 reports the numerical results over the full lifetime and the last 50 flights, highlighting improved accuracy in late-stage predictions.

Table 1 RUL prediction performance on the simulated dataset using the proposed EOL PF method.

Prediction window	RMSE	PICP	PINAW
Full life	20.37	94.72%	39.95%
Last 50 flights	6.47	100.00%	57.71%

5 Conclusions

We introduced a data-driven particle filtering framework for system-level prognostics that estimates remaining useful life and its uncertainty directly from performance observations. Unlike traditional model-based approaches, our method operates without requiring a physical model of the system. Instead, it learns a probabilistic degradation process offline from historical data and updates this belief online using a modified PF. This structure naturally captures both stochastic and epistemic uncertainty, while enhancing the interpretability

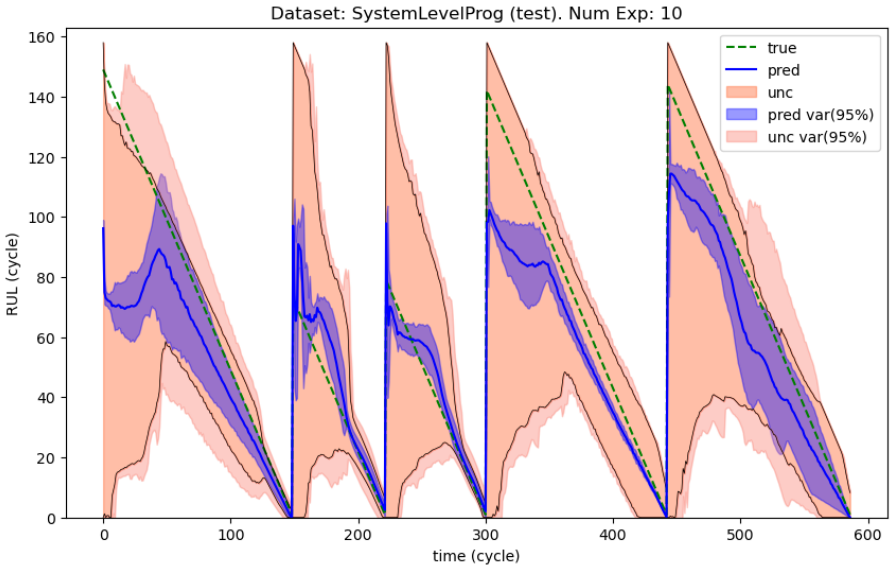


Figure 4 Predicted remaining useful life over flight cycles for the five test UAVs. The green dashed line shows the true RUL, the solid blue line is the mean predicted RUL, and the black line represents the mean of the 95% confidence interval across 10 repeated runs. The shaded regions show the 95% confidence intervals for the predictive variance (blue) and uncertainty variance (salmon), estimated using 1.95 standard deviations.

of purely data-driven models through performance-indexed degradation trajectories. We evaluated the approach on a high-fidelity UAV simulation environment and demonstrated that it produces accurate and uncertainty-aware predictions throughout the degradation lifecycle.

In future work, we plan to evaluate our method on the N-CMAPSS benchmark [3] and on real-world datasets to compare its performance against state-of-the-art approaches. Although N-CMAPSS lacks explicit performance metric trajectories, we are developing a strategy to infer them post hoc. We also plan to enhance the flexibility of the PF by learning the correction strength and propagation noise parameters by using neural networks. Due to the stochastic nature of PFs, this will likely require a differentiable approximation or a reparameterization-style trick to enable gradient-based training. Such extensions could improve both learning efficiency and expressiveness, allowing the model to better adapt to complex, nonlinear degradation behaviors.

References

1 Ibrahim Ahmed, Marcos Quinones-Grueiro, and Gautam Biswas. A high-fidelity simulation test-bed for fault-tolerant octo-rotor control using reinforcement learning. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2022.

2 Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019. doi:10.1145/3292500.3330701.

3 Manuel Arias Chao, Chetan Kulkarni, Kai Goebel, and Olga Fink. Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics. *Data*, 6(1):5, 2021. doi:10.3390/DATA6010005.

4 Anne Auger and Nikolaus Hansen. A restart cma evolution strategy with increasing population size. In *2005 IEEE congress on evolutionary computation*, volume 2, pages 1769–1776. IEEE, 2005. doi:10.1109/CEC.2005.1554902.

5 Yuanhong Chang, Fudong Li, Jinglong Chen, Yulang Liu, and Zipeng Li. Efficient temporal flow transformer accompanied with multi-head probsparse self-attention mechanism for remaining useful life prognostics. *Reliability Engineering & System Safety*, 226:108701, 2022. doi:10.1016/J.RESS.2022.108701.

6 Manuel Chao, Chetan Kulkarni, Kai Goebel, and Olga Fink. Fusing physics-based and deep learning models for prognostics. *Reliability Engineering & System Safety*, 217:107961, 2022. doi:10.1016/J.RESS.2021.107961.

7 Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, New York, 2001.

8 Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140(2):107–113, 1993.

9 Jian Guo, Zhaojun Li, and Meiyan Li. A review on prognostics methods for engineering systems. *IEEE Transactions on Reliability*, 69(3):1110–1129, 2019. doi:10.1109/TR.2019.2957965.

10 Cheng-Geng Huang, Hong-Zhong Huang, and Yan-Feng Li. A bidirectional lstm prognostics method under multiple operational conditions. *IEEE Transactions on Industrial Electronics*, 66(11):8792–8802, 2019. doi:10.1109/TIE.2019.2891463.

11 Dersin Pierre, Kristupas Bajarunas, and Manuel Arias-Chao. Analytical modeling of health indices for prognostics and health management. In *PHM Society European Conference*, volume 8, pages 11–11, 2024.

12 Darius V Roman, Ross W Dickie, David Flynn, and Valentin Robu. A review of the role of prognostics in predicting the remaining useful life of assets. In *27th European Safety and Reliability Conference 2017*, pages 897–904. CRC Press, 2017.

13 Abhinav Saxena, Jose Celaya, Bhaskar Saha, Sankalita Saha, and Kai Goebel. Metrics for offline evaluation of prognostic performance. *International Journal of Prognostics and health management*, 1(1):4–23, 2010.

14 Enrico Zio and Giovanni Peloni. Particle filtering prognostic estimation of the remaining useful life of nonlinear components. *Reliability Engineering & System Safety*, 96(3):403–409, 2011. doi:10.1016/J.RESS.2010.08.009.

GEMMA-FD: Zero-Shot Fault Detection in Heat Pumps Using Multimodal Language Models

Herbert Muehlburger
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Franz Wotawa
Institute of Software Engineering and Artificial Intelligence, Graz University of Technology, Austria

Abstract

Fault detection in heating systems is critical for ensuring energy efficiency and operational reliability. Traditional approaches rely on labeled fault data and expert-defined rules, which are often unavailable or costly to obtain. We introduce GEMMA-FD (*GEMMA for Fault Detection*), a novel zero-shot framework for fault detection in heat pumps that leverages large language models (LLMs) without requiring labeled anomalies or predefined fault signatures. Our method transforms multivariate sensor time series into structured natural language prompts and augments them with visual features, such as line plots of key variables, to facilitate multimodal reasoning. Using GEMMA-3, an open-weight multimodal LLM, we classify heat pump system states as either normal or faulty. Experiments on a real-world heat pump dataset show that GEMMA-FD can identify unseen faults with reasonable precision, although its performance remains lower than a supervised XGBoost baseline trained on the same prompts. Specifically, GEMMA-FD achieves a macro-F1 score of 0.252, compared to 0.69 for XGBoost, underscoring the trade-off between generalization and targeted accuracy. Nevertheless, GEMMA-FD demonstrates the potential of foundation models for interpretable, multilingual fault detection in cyber-physical systems, while highlighting the need for prompt engineering, few-shot augmentation, and multimodal inputs to improve the classification of rare and complex fault types.

2012 ACM Subject Classification Computing methodologies → Machine learning; Computing methodologies → Natural language processing; Software and its engineering → Software testing and debugging; Computing methodologies → Knowledge representation and reasoning; Computing methodologies → Anomaly detection; Computer systems organization → Embedded and cyber-physical systems

Keywords and phrases fault detection, anomaly detection, cyber-physical systems, HVAC, heat pumps, energy systems, large language models, zero-shot learning, open-weight LLMs, interpretable AI, multimodal prompts, smart energy

Digital Object Identifier 10.4230/OASIcs.DX.2025.12

Supplementary Material *Software (Reproducibility package)*: `https://doi.org/10.5281/zenodo.16948128`

Funding This work was partially funded by the FFG project "Scalable Agents for Building Management and Energy Efficiency" under grant number FO999923190.
Herbert Muehlburger: The position of this author is funded by the FFG project "The automated ontology generator" under grant number FO999901761.

Acknowledgements We thank the anonymous reviewers for their constructive and valuable feedback.

1 Introduction

Heating, ventilation, and air conditioning (HVAC) systems account for over 30% of global energy consumption, with heat pumps playing a critical role in modern energy-efficient buildings [3]. As cyber-physical systems (CPS), heat pumps generate high-dimensional sensor data that reflect complex operational states. Detecting faults in these systems is essential to ensure energy efficiency, equipment longevity, and user comfort.

Traditional fault detection and diagnosis (FDD) approaches rely on physics-based models or expert-defined rules [6], which require deep domain expertise and are difficult to scale. Supervised machine learning (ML) has emerged as a scalable alternative [6, 9], but it relies on large volumes of labeled fault data – often unavailable in real-world CPS deployments. To mitigate this issue, semi-supervised and weakly supervised methods [5, 16] have been explored, and regression-based field modeling approaches have also been proposed [8]. However, these still depend on handcrafted features or partial annotations.

Recent advances in large language models (LLMs) offer a new paradigm for fault detection. LLMs show strong generalization capabilities across diverse tasks, including structured reasoning and zero-shot classification. Prior work has explored their use in CPS contexts, such as prompt-based anomaly detection in battery systems [7], and time-series anomaly detection using structured inputs [1]. In addition, Xu et al. [15] proposed a benchmark called VisualTimeAnomaly that translates time series into visual representations, enabling multimodal LLMs to reason about sensor dynamics. However, LLM performance often lags behind deep learning models unless guided by carefully engineered prompts.

Meanwhile, foundation models for time series forecasting have shown that prompt structure and in-context reasoning play a key role in zero-shot generalization. TimesFM [4] demonstrates strong forecasting performance using a decoder-only architecture trained on time-series data, and TiRex [2] improves forecasting via context distillation and hybrid memory attention. These developments highlight the importance of architectural and representational choices in zero-shot time-series tasks.

Building on these insights, we ask: can LLMs detect faults in heat pump systems using only data from normal operation – without labeled fault data or expert rules?

To answer this, we introduce **GEMMA-FD**, a zero-shot, multimodal LLM framework for fault detection in CPS. We reformulate sensor-based fault detection as a structured prompting task: multivariate time series are converted into natural language descriptions enriched with statistical summaries, trend cues, and operating modes. We augment these textual prompts with visual inputs (e.g., heatmaps, line plots, histograms) to guide multimodal reasoning in GEMMA-3 [13], an open-weight, vision-and-language multimodal LLM released by Google DeepMind, in a purely zero-shot setting. No fine-tuning or labeled faults are required. Details on prompt construction and visual summarization are described in Section 3.2.

Our evaluation shows that GEMMA-3 tends to identify normal operating states correctly, but struggles to detect rare faults – reflecting the current limitations of zero-shot classification in highly imbalanced CPS settings. Nevertheless, the model captures CPS dynamics to a surprising extent and offers a reproducible, interpretable, and multilingual baseline for low-resource FDD.

Our main contributions are:

- We present GEMMA-FD, a zero-shot fault detection framework using GEMMA-3, which transforms sensor time series into structured natural language prompts without requiring labeled fault data.
- We introduce prompt designs that incorporate domain-specific interpretations, sensor correlations, and visual features (e.g., heatmaps, timeseries plots, histograms) to enhance LLM reasoning over complex system behavior.
- We benchmark GEMMA-3 against a supervised XGBoost baseline trained on the same prompts, highlighting the trade-offs between zero-shot generalization and supervised learning for rare fault detection.
- We release a fully reproducible pipeline for heat pump fault detection with open-weight LLMs, including data preprocessing, prompt generation, and evaluation code.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 details our methodology and dataset. Section 4 presents and discusses the experimental results. Finally, Section 6 concludes the paper and outlines directions for future research.

2 Related Work

Fault Detection and Diagnosis (FDD) in cyber-physical systems (CPS), including HVAC and heat pump systems, has traditionally relied on model-based or data-driven methods. Classical approaches use physics-based models or expert-defined rules [6], which require significant domain knowledge and manual effort. As sensor data has become more abundant, supervised machine learning (ML) methods have emerged [6], although they rely on labeled fault data that are often costly and limited in diversity.

Several studies have benchmarked ML algorithms for heat pump FDD. Rahman et al. [9] compared XGBoost, Random Forest, SVM, and k-NN, identifying XGBoost as the most effective. Weigert et al. [14] utilized smart meter data for operational and anomaly classification. Shin and Cho [11] predicted the Coefficient of Performance (COP) to support FDD. Other work, such as by Sunal et al. [12], applied deep learning to fault detection in centrifugal pumps, offering insights transferable to HVAC systems. To reduce dependency on labeled data, semi-supervised and weakly supervised methods have been explored [5, 16]. Puttige et al. [8] demonstrated how regression and neural networks trained on field data can model heat pump behavior effectively.

Time Series and LLM-based Anomaly Detection. Recent advances in large language models (LLMs) have introduced new opportunities for time series anomaly detection. Alnegheimish et al. [1] explored whether general-purpose LLMs can act as zero-shot anomaly detectors and found that structured prompting is essential, though performance often trails deep learning baselines. Xu et al. [15] proposed AnomalyTransformer, a self-attention-based approach for modeling dependencies in time series anomalies. The same authors introduced VisualTimeAnomaly, a benchmark that translates time series into visual representations to evaluate multimodal LLM performance on TSAD tasks.

Foundation models tailored to time series have further advanced zero-shot reasoning. TimesFM [4] presents a decoder-only transformer trained on large-scale time series for forecasting tasks. TiRex [2] builds on this by improving in-context forecasting performance across short and long horizons via context distillation and hybrid memory attention. While both focus on forecasting, their findings inform how architectural and prompt design choices affect zero-shot generalization for time-dependent tasks.

LLMs in Fault Detection. Muehlburger et al. [7] demonstrated prompt-based anomaly detection in battery systems using open-weight LLMs, without requiring fine-tuning. In the HVAC domain, Hofer and Wotawa [6] showed that supervised learning informed by expert knowledge can yield high-performance FDD models for heat pumps.

Despite their generalization capacity, current LLMs often struggle with zero-shot classification of rare anomalies unless augmented with domain-specific cues, visual summaries, or few-shot examples. Our work builds on these insights by introducing a fully prompt-based, multimodal, zero-shot fault detection framework. We show that open-weight foundation models can partially encode CPS operational manifolds through structured prompts and visualizations – though further enhancement is needed for reliable minority-class classification.

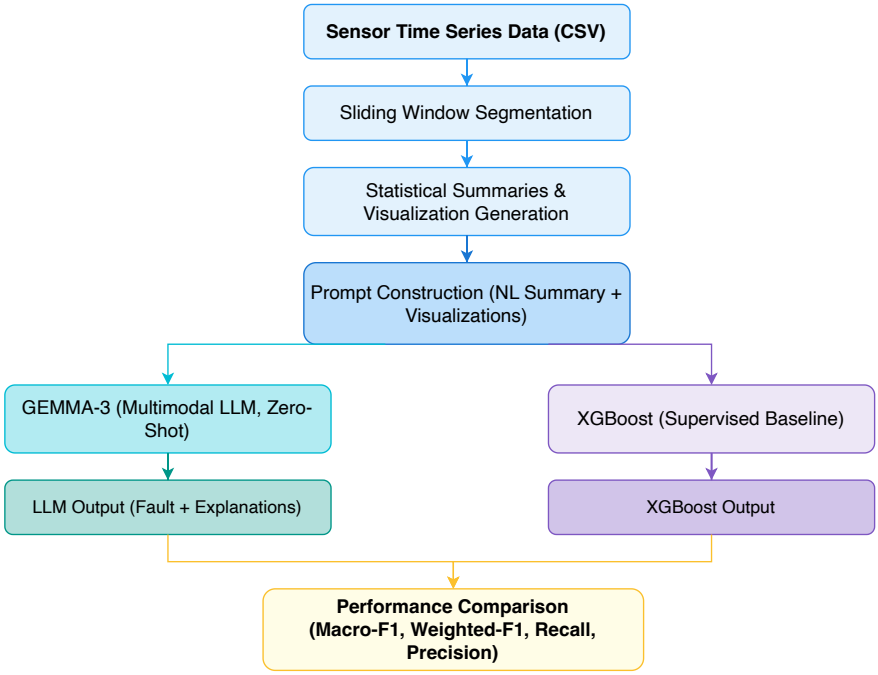


Figure 1 Overview of the GEMMA-FD pipeline. Sensor data is windowed and summarized with statistics and visualizations. Structured prompts are fed into GEMMA-3 for zero-shot inference, while the same text is used in a supervised XGBoost baseline. Outputs are compared using standard performance metrics.

To the best of our knowledge, this is the first application of an open-weight, zero-shot LLM-based method for multilingual, interpretable, and label-free fault detection in heat pump systems. We position GEMMA-FD as a diagnostic baseline, illuminating the strengths and limitations of current foundation models in real-world CPS deployments.

3 Methodology

We compare two contrasting diagnostic pipelines for fault detection in heat pump systems: a supervised baseline using XGBoost and a zero-shot multimodal baseline using GEMMA-3. Both pipelines consume the same structured prompt format but differ in knowledge acquisition: XGBoost is trained on labeled data, while GEMMA-3 infers labels in a zero-shot setting without any task-specific training.

- **Supervised baseline (XGBoost):** A gradient-boosted decision tree model trained on structured prompt-label pairs derived from sensor data.
- **Zero-shot baseline (GEMMA-3):** An open-weight multimodal large language model (LLM) that receives structured prompts and visual summaries as input and performs inference-only classification.

This setup allows us to assess the trade-off between supervised accuracy and zero-shot generalization in low-resource diagnostics. Figure 1 illustrates the overall fault detection pipeline (XGBoost had been trained before).

3.1 Dataset and Pipeline Overview

We use multivariate time series data logged from an Austrian heat pump system. Each row in the CSV logs represents a 1-minute snapshot of multiple synchronized sensor readings. The dataset contains operational data of the heat pump system (AnlageA_JKj), annotated into four categories: `betrieb_ok` (normal operation), `defrosting_issue`, `driver_temp_error`, and `overheating_control_issue`. The dataset contains in total 130,351 samples. For model training and evaluation we split the data into training (70%, 91,245 samples), validation (15%, 19,553), and test (15%, 19,553) sets using a stratified sampling strategy. We further follow two different data pre-processing paths:

- **XGBoost path:** Each row is treated as an individual sample for supervised training and testing.
- **GEMMA-3 path:** Data is segmented into overlapping 4-hour windows. Each window is converted into a textual prompt and three visualizations.

All samples are transformed into structured natural language prompts (see Section 3.2). For GEMMA-3, the prompt is enriched with composite images (heatmap, time series, and histogram) to support multimodal reasoning.

The GEMMA-FD pipeline processes multivariate time series data logged from a real-world heat pump system. Sensor readings are stored as CSV files, where each row represents a 1-minute interval of synchronized measurements across multiple physical variables (e.g., temperatures, flow rates, compressor speed). We implement two processing pathways:

- A **row-wise** path for supervised learning (XGBoost) that treats each minute independently.
- A **windowed** path for zero-shot prompting (GEMMA-3) that aggregates sensor readings over a 4-hour sliding window to provide temporal context.

As illustrated in Algorithm 1 each sample (row or window) is transformed into a **structured textual prompt**. For GEMMA-3, additional **multimodal visual summaries** (heatmap, time series, histogram) are also generated and jointly passed to the model.

Algorithm 1 GEMMA-FD Pipeline Overview.

Require: Multivariate time series $\mathbf{X}$; annotated row-wise labels (for supervised baseline)

if Supervised **then**
  for each row x_i in $\mathbf{X}$ **do**
    Construct structured prompt $P(x_i)$ from features
  end for
  Train XGBoost on TF–IDF representations of $P(x_i)$
else
  for each time window W_i in $\mathbf{X}$ **do**
    Extract statistical summaries and trends
    Generate time-series plot, heatmap, and histogram
    Construct structured natural language prompt $P(W_i)$
    Pass $P(W_i)$ and corresponding image to GEMMA-3 for zero-shot classification
  end for
end if
return Predicted class labels $\hat{y}$

3.2 Prompt Generation

To enable large language models (LLMs) to reason about system state, we convert raw heat pump sensor data into structured textual summaries, using a sliding-window approach. Each window corresponds to a fixed temporal segment (e.g., 4 hours) of multivariate time-series data, which is then transformed into a prompt as follows:

- **Sensor statistics**: For each relevant sensor (e.g., compressor speed, flow rate, inlet/outlet temperatures), we compute the mean, minimum, and maximum over the window. These are presented in natural language.
- **Trend detection**: For selected sensors, we detect increasing or decreasing trends using the first and last sample in the window.
- **Operating mode inference**: If available, the categorical "wp-app-state_38" field is used to infer the dominant operational mode (e.g., heating, standby, defrost).
- **Visual summary (multimodal input)**: A composite image is generated per window containing: (i) a z-score normalized heatmap, (ii) raw time-series plots, and (iii) sensor histograms. This image is passed to the LLM alongside the text.
- **Instructions for classification**: We append an instruction block asking the LLM to jointly analyze the image and text and classify the window into one of four predefined fault categories.

These design choices aim to emulate how human experts summarize system behavior and enable the LLM to leverage both symbolic and visual cues for anomaly detection. An example prompt is shown below:

Listing 1 Zero-shot user-prompt for GEMMA-3 with statistical summaries and classification instructions.

```
=== HEAT PUMP SENSOR WINDOW SUMMARY ===
Samples: 60 (2023-01-10 08:00 to 2023-01-10 12:00)
Most frequent operating mode: heating mode

Sensor statistics:
T2 Außen: mean=2.41, min=-1.00, max=5.20
wp-comp-speed: mean=2800.33, min=2600.00, max=3200.00
...

Trends:
Compressor speed increased over the window.

Instructions:
- The image below contains a Z-score normalized heatmap and time series plots for all sensors.
- Analyze the numeric data and the image.
- Look for abnormal periods, outliers, or persistent trends.
- Classify the window as one of: [betrieb_ok, defrosting_issue, driver_temp_error, overheating_control_issue].
- Output ONLY three function calls as specified. Reply in English and German.
```

Listing 1 presents a representative user prompt automatically generated using lightweight Python functions, while Listing 2 shows the corresponding system prompt provided to the model. This fully automated setup eliminates the need for manual annotation and enables scalable, LLM-based analysis of large, unlabeled datasets.

3.3 Model Configuration

We use a locally hosted, open-weight multimodal language model `gemma3:4b`, available through the Ollama framework. This variant supports both text and image input and is based on the GEMMA-3 architecture released by Google DeepMind in 2025. Table 1 summarizes key model characteristics.

Table 1 LLM configuration for windowed heat pump analysis.

Property	Value
Model	gemma3:4b (multimodal)
Parameters	4 billion
Weights	Open-source (Google DeepMind GEMMA-3)
Inference engine	Ollama v0.1.34
Modality	Multimodal (image + text)
Context window	8,192 tokens
Hardware	Apple MacBook Pro 14.2", M1 Pro, 10 Core CPU, 16 Core GPU
Average inference time	∼2 seconds per window
Batching	Single-sample (per window)
Prompt tuning	None (zero-shot only)

All predictions were performed using zero-shot prompting, without any fine-tuning or few-shot examples. Each prompt-image pair is sent independently to the model. The outputs are parsed to extract fault labels and free-text explanations.

3.4 Zero-Shot Fault Detection with GEMMA-3

We select `gemma3:4b`, an open-weight, multimodal LLM released by Google DeepMind [13] and served via the Ollama framework, to maximize reproducibility and local deployment feasibility. This 4B-parameter model supports both text and image inputs and is well-suited for vision-language inference tasks in constrained environments.

All experiments are conducted locally, with each sample consisting of a structured textual prompt and three visualizations as described in Section 3.2. The model receives a diagnostic system prompt specifying the output format; only the predicted label from the `report` function call is used for evaluation, parsed via regular expressions.

Evaluation is performed on a stratified sample of 300 test prompts. This setup reflects the label-scarce, zero-shot conditions of real-world fault detection. Our method is inspired in part by VisualTimeAnomaly [15], which highlights the value of multimodal visualizations for time series anomaly detection, and by Alnegheimish et al. [1], who emphasize the role of structured prompt engineering for LLM-based time series analysis.

To enrich the LLM's context, each inference window is described by a structured natural language prompt and three visualizations: a z-score normalized heatmap of all sensor signals (Figure 2a), a time series plot (Figure 2b), and histograms of value distributions for each sensor (Figure 2c). All visual inputs are directly derived from real-world heat pump sensor

Listing 2 System prompt provided to GEMMA-3 for multimodal zero-shot fault classification in heat pumps. The model outputs three Python function calls: `report()`, `diagnostics()`, and `diagnostics_de()`.

```
You are a diagnostic assistant for heat pump systems.
For each sample, you will receive:
- A plot (attached image) showing a 4-hour window of all sensor data (heatmap, time series, and histogram)
- A structured textual summary of that window

Your task:
- Use BOTH the plot and the textual summary to classify the system state during this period.
- Consider all short or persistent abnormalities, not just the overall trend.
- If you are unsure, explain your reasoning in the diagnostics output.
- All classes are equally likely.

Output format:
- Output exactly three Python function calls, in this order:
   1. report(fault_type: str)
   2. diagnostics(explanation: str) # (English)
   3. diagnostics_de(explanation: str) # (German)

Valid fault types:
- betrieb_ok: No abnormalities detected during the 24h period.
- defrosting_issue: Signs of persistent or recurring defrosting problems.
- driver_temp_error: Driver circuit temperature is out of expected range.
- overheating_control_issue: Evidence of overheating or failed thermal regulation.

Example output:
report('driver_temp_error')
diagnostics('Between 03:00 and 04:00, the driver temperature exceeded the normal range. The rest of the day appears normal.')
diagnostics_de('Zwischen 03:00 und 04:00 lag die Fahrertemperatur auß erhalb des Normalbereichs. Der Rest des Tages war unauffällig.')

IMPORTANT: Do not output any explanations or extra text outside of these three function calls. Do not include code blocks or define any functions.
```

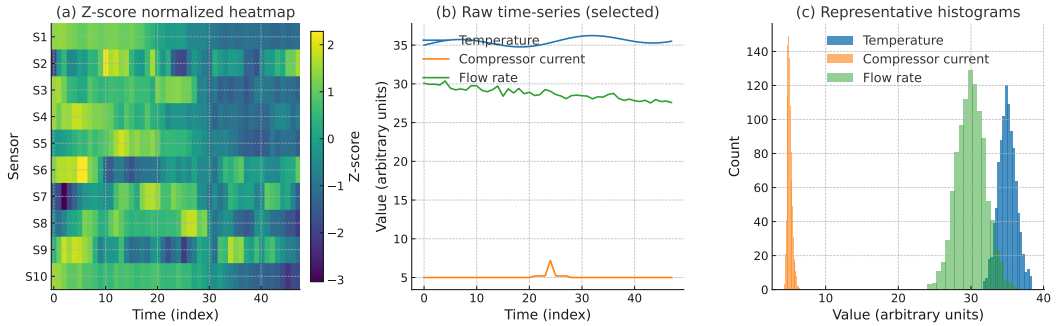


Figure 2 Visual encodings used as multimodal inputs to GEMMA-3. Each time window is represented as **(a)** a z-score normalized heatmap of sensor signals, **(b)** selected raw time-series plots, and **(c)** representative histograms of sensor value distributions. The values shown are exemplary and schematic for clarity; actual sensor values differ in scale and range but are visualized in the same encoding format. This schematic representation highlights the multimodal input structure (temporal, statistical, distributional) rather than domain-specific magnitudes.

data collected in Austria. The axis labels and variable names in the generated plots appear in German, as they reflect the naming conventions of the original control system and are passed to GEMMA-3 without translation to preserve semantic fidelity. These visuals capture temporal and statistical patterns, supporting robust zero-shot anomaly and fault detection.

For the zero-shot fault detection experiments, we evaluated GEMMA-3 using a stratified random sample of 300 test windows. To ensure class balance in the evaluation set, we performed stratified sampling across all annotated system states. Each sample consisted of a structured prompt and multimodal visualizations, as described previously.

The prediction workflow is as follows:

1. Stratify and sample $N = 300$ test windows from the full labeled dataset, preserving class proportions.
2. For each sampled window, generate a structured prompt and corresponding visualizations.
3. Run the `llm_predict` function, which queries GEMMA-3 via the Ollama inference framework with each prompt-image pair.
4. Collect and store all raw LLM outputs for further analysis and reproducibility.
5. Parse predicted fault labels from the structured LLM output using regular expressions.
6. Save all predictions and explanations to CSV and TXT files for downstream analysis.
7. Compute classification metrics (precision, recall, F1-score) and confusion matrix on the held-out samples.
8. Export results as LaTeX tables and plots for inclusion in the paper.

All LLM experiments are conducted with the model running locally to ensure full reproducibility and data privacy. We restrict the evaluation to 300 test samples per run to manage computational demands and facilitate interpretability of individual LLM predictions.

3.5 Supervised Baseline: XGBoost

For the supervised baseline, we train an XGBoost classifier using the full row-level dataset, where each row corresponds to a sensor snapshot and is labeled with one of the four classes: `betrieb_ok`, `defrosting_issue`, `driver_temp_error`, or `overheating_control_issue`. Individual CSV files for each class are parsed and transformed into prompt-label pairs using domain-informed prompt generation. After concatenating the class-specific data, we

split the combined dataset into training, validation, and test partitions. To address class imbalance, the training set is balanced using random oversampling. Each prompt is then vectorized using term frequency-inverse document frequency (TF–IDF) features, allowing up to 2,000 unigrams and bigrams.

The XGBoost classifier (`objective="multi:softprob"`) is trained on the balanced training data with hyperparameters set to 200 trees, a learning rate of 0.05, and a maximum tree depth of 6. After training, the model is evaluated on the held-out test partition, with classification metrics and confusion matrices generated to assess performance. This process is summarized in the following steps:

1. Load and concatenate row-level CSV data for each fault class.
2. Generate structured prompt-label pairs per sample.
3. Stratify into train/validation/test splits.
4. Apply random oversampling to balance the training set.
5. Vectorize prompts with TF–IDF.
6. Train the XGBoost classifier and evaluate on the test set.
7. Report class-wise precision, recall, F1-score, and confusion matrix.

This workflow describes our supervised baseline for comparison with zero-shot LLM-based fault detection.

3.6 Evaluation Metrics and Protocol

We report precision, recall, and F1-score for each class, as well as macro- and weighted averages to account for class imbalance and fault rarity, following Seliya et al. [10]. Confusion matrices visualize misclassifications.

Evaluation Protocols. For computational feasibility and qualitative inspection, GEMMA-3 was evaluated on 300 stratified, randomly-sampled windows from the test set, ensuring balanced class representation. XGBoost was evaluated on the full test set of 19,553 samples. Here are the details for the two settings:

- **Windowed Zero-Shot LLM (GEMMA-3):** We selected a stratified random sample of $N = 300$ test windows to ensure balanced class representation. Each sample was processed into a structured prompt and multimodal visualization. This sample size was chosen to balance computational feasibility and qualitative interpretability of individual LLM outputs.
- **Row-Wise and XGBoost (Full Test Set):** For the row-wise GEMMA-3 variant and the supervised XGBoost baseline, we evaluated on the complete held-out test set ($n \approx 19,553$ samples). No additional stratification was applied; all labeled samples were included for the XGBoost training.

For each protocol, predictions were parsed, performance metrics computed, and confusion matrices generated. This dual protocol quantifies both balanced and full-scale model performance for zero-shot and supervised approaches.

Metric definitions. Precision ($\frac{TP}{TP+FP}$) measures the proportion of correctly identified faults among flagged cases; recall ($\frac{TP}{TP+FN}$) measures the proportion of actual faults correctly detected; F1-score ($2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$) is their harmonic mean.

4 Results

This section presents a comparative analysis of fault detection in heat pump systems using three approaches: (1) GEMMA-3 in a zero-shot setting with windowed, multimodal prompts, (2) GEMMA-3 with row-wise, non-windowed prompts, and (3) a supervised XGBoost baseline. We evaluate all methods on the same structured dataset, reporting class-wise precision, recall, and F1-scores, as well as macro- and weighted averages to account for class imbalance. Quantitative results are complemented by qualitative analyses of LLM output.

4.1 Overall Performance

GEMMA-3, evaluated on a stratified random sample of 300 windowed test prompts, achieved a macro-F1 score of 0.19 and a weighted F1 of 0.39 (Table 2), indicating reliable detection of the dominant class but limited sensitivity to rare faults. XGBoost, in contrast, reached a macro-F1 of 0.69 and a weighted F1 of 0.83 on the full test set (19,553 samples; Table 4), demonstrating robust, balanced performance across all fault categories. These results underscore the advantage of supervised learning with labeled data for comprehensive fault detection, while highlighting the generalization challenge faced by zero-shot LLM-based approaches.

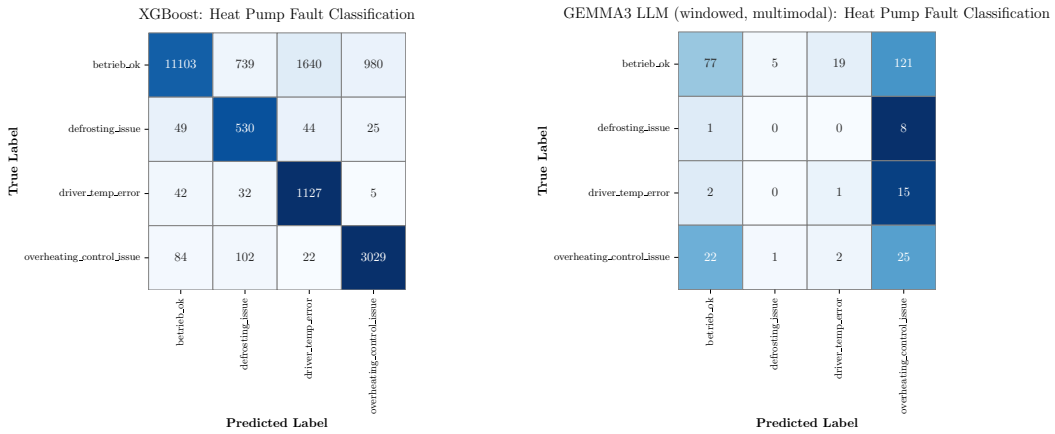
We observe that GEMMA-3 achieves its highest recall on the normal class (`betrieb_ok`), suggesting that it captures expected system behavior more reliably than anomalous regimes, consistent with prior findings on LLM robustness to typical inputs [1].

4.2 Class-Wise Performance Trends

A detailed examination of class-wise metrics reveals that GEMMA-3 achieves its highest recall for the majority class, `betrieb_ok`, but performs poorly on the minority fault classes. Specifically, GEMMA-3 attains a recall of 0.345 for `betrieb_ok`, while recall drops to zero for `defrosting_issue` and to 0.056 for `driver_temp_error`. The model achieves a moderate recall of 0.500 for `overheating_control_issue`, but with low precision. In contrast, the supervised XGBoost model shows high recall across all classes, including 0.818 for `defrosting_issue`, 0.934 for `driver_temp_error`, and 0.936 for `overheating_control_issue`. This demonstrates that XGBoost not only identifies the majority class but also maintains strong detection performance for rare faults.

4.3 Error Patterns and Limitations

Analysis of the confusion matrix for GEMMA-3 (Figure 3b) shows that the model predominantly assigns samples to the majority class, `betrieb_ok`, resulting in frequent misclassification of fault cases as normal operation. Both `defrosting_issue` and `driver_temp_error` are rarely, if ever, correctly identified, with most of these cases assigned to the normal class. For `overheating_control_issue`, GEMMA-3 correctly identifies some instances but also produces a high number of false positives. In contrast, the confusion matrix for XGBoost (Figure 3a) is dominated by high values along the diagonal, indicating correct classification for all classes and minimal confusion between normal and fault types.



(a) XGBoost on full test set.

(b) GEMMA-3 on 300 sampled prompts.

Figure 3 Confusion matrices comparing supervised and zero-shot diagnostic performance. **(a)** XGBoost achieves high accuracy across all fault classes with supervised training. **(b)** GEMMA-3, operating in a zero-shot setting, displays strong bias toward the majority class (`betrieb_ok`), with limited sensitivity to rare faults.

4.4 Performance Comparison

To further clarify the differences between approaches, we present a direct comparison of zero-shot LLM classification with GEMMA-3 and supervised learning with XGBoost. Both models are evaluated on identical input data and assessed using standard classification metrics. The following subsections provide detailed results for each method.

4.4.1 Zero-Shot Inference with Windowing

Table 2 shows the results for the zero-shot setting, where GEMMA-3 was evaluated on 300 randomly selected windows of test prompts. The model output was parsed to extract the predicted fault class. Figure 3b shows the confusion matrix for this evaluation.

Table 2 Classification performance of GEMMA-3 for zero-shot fault detection in heat pump systems using structured prompts and windowed multimodal data. The model shows strong recall for normal operation but limited accuracy for rare fault classes, underscoring the challenges of zero-shot detection without labeled data.

Class	Precision	Recall	F1	Support
betrieb_ok	0.755	0.345	0.474	223
defrosting_issue	0.000	0.000	0.000	9
driver_temp_error	0.045	0.056	0.050	18
overheating_control_issue	0.148	0.500	0.228	50
micro avg	0.344	0.343	0.344	300
macro avg	0.237	0.225	0.188	300
weighted avg	0.589	0.343	0.393	300

The model assigned the majority of samples to the `betrieb_ok` class, resulting in a recall of 0.345 for this category. For the rare fault classes, recall was near zero for `defrosting_issue` and 0.056 for `driver_temp_error`, while recall for `overheating_control_issue` reached 0.500 but with low precision. The macro-F1 score was 0.188, and the weighted-F1 was 0.393, indicating overall limited effectiveness in rare fault detection.

Table 3 Classification performance of GEMMA-3 for zero-shot fault detection in heat pump systems using row-wise, non-windowed prompts. Compared to the windowed setting, the model shows higher recall for some fault classes but still struggles to detect `overheating_control_issue`, indicating limitations in distinguishing complex fault patterns without temporal context.

Class	Precision	Recall	F1	Support
betrieb_ok	0.743	0.706	0.724	14,051
defrosting_issue	0.046	0.251	0.078	630
driver_temp_error	0.159	0.299	0.207	1,170
overheating_control_issue	0.000	0.000	0.000	0
micro avg	0.549	0.658	0.598	15,851
macro avg	0.237	0.314	0.252	15,851
weighted avg	0.672	0.658	0.660	15,851

4.4.2 Row-Wise Zero-Shot Inference (No Windowing)

Table 3 reports classification performance for GEMMA-3 in a row-wise zero-shot setting, where each time step is treated independently without temporal aggregation. The model was evaluated on the entire test set ($n = 15{,}851$) using non-windowed prompts.

The highest F1-score was achieved for the normal operating state `betrieb_ok` (F1 = 0.724), with precision and recall of 0.743 and 0.706, respectively. This indicates that the model performs reasonably well for detecting normal operation. Fault classes, however, exhibit significantly lower scores. For example, `defrosting_issue` achieved a recall of 0.251 but extremely low precision (0.046), indicating frequent false positives. Similarly, `driver_temp_error` reached a recall of 0.299 with low precision (0.159), suggesting limited fault discrimination ability.

Notably, the model completely failed to detect any instances of `overheating_control_issue`, as indicated by zero precision, recall, and F1 score likely due to lack of context in single-timestep prompts. This aligns with the observation that more complex temporal patterns are difficult to infer without windowed context.

The macro-averaged F1-score of 0.252 and the weighted average of 0.660 reflect the model's bias toward majority classes. Although the row-wise approach scales more easily to large datasets, it fails to capture time-dependent anomaly patterns essential for robust fault diagnosis. These findings highlight inherent challenges in employing multimodal LLMs directly for fault detection tasks, especially for minority classes. Potential strategies to enhance performance include targeted prompt-tuning, data augmentation techniques, or incorporating ensemble methods to improve robustness and accuracy.

4.4.3 Supervised XGBoost Baseline

The supervised XGBoost model was trained and evaluated on the same dataset as the LLM, using class-weighted loss to address class imbalance. It achieved, as shown in Figure 4, a macro-average F1 score of 0.693 and a weighted-average F1 score of 0.827. Recall exceeded 0.81 for all fault classes, with the highest values observed for `overheating_control_issue` (0.936) and `driver_temp_error` (0.934).

Table 4 Classification performance of XGBoost trained on supervised fault labels for heat pump fault detection. XGBoost demonstrates high precision and recall across all classes, including rare faults, highlighting the effectiveness of supervised learning when labeled data are available.

Class	Precision	Recall	F1	Support
betrieb_ok	0.984	0.768	0.863	14,462
defrosting_issue	0.378	0.818	0.517	648
driver_temp_error	0.398	0.934	0.558	1,206
overheating_control_issue	0.750	0.936	0.833	3,237
macro avg	0.627	0.864	0.693	19,553
weighted avg	0.889	0.807	0.827	19,553

These results demonstrate that supervised, class-aware training yields strong diagnostic performance, even when the input consists of structured natural language prompts identical to those used by the LLM.

4.5 Error Analysis

A detailed examination of the GEMMA-3 results (Tables 2 and 3) reveal three dominant error patterns:

- **Majority class bias:** GEMMA-3 disproportionately predicts the `betrieb_ok` (normal operation) class, even for samples labeled with faults. This leads to very low recall for rare fault types such as `defrosting_issue` (0.000) and `driver_temp_error` (0.056), reflecting a strong bias toward the majority class. This behavior is clearly visible in the confusion matrix (Figure 3b).
- **Limited fault-specific detection:** Fault samples are rarely classified with the correct fault type, even when statistical summaries contain values outside expected ranges. This suggests that GEMMA-3, operating under zero-shot conditions, struggles to associate specific numerical patterns with fault semantics.
- **Overprediction of certain faults:** For `overheating_control_issue`, the model exhibits moderate recall (0.500) but low precision (0.148). This pattern suggests that the model often predicts this fault incorrectly (false positives), indicating a tendency to overtrigger this class rather than true ambiguity at the decision boundary.

In contrast, the XGBoost baseline achieves high recall and precision for all classes, including minority faults (Table 4). Its confusion matrix (Figure 3a) is dominated by correct predictions, with minimal misclassification between fault types. This contrast highlights the challenge of zero-shot LLM-based fault detection – namely, that generalization from unlabeled normal data is insufficient for robust fault classification in complex cyber-physical systems.

These findings underscore the need for future work on:

- Enhancing prompt design with explicit fault indicators and temporal patterns,
- Applying few-shot learning to expose models to representative fault cases,
- Leveraging visual modality inputs (e.g., plots) more effectively for multimodal reasoning.

4.6 Comparison and Insights

A direct comparison between GEMMA-3 and XGBoost highlights the fundamental trade-off between label-free generalization and supervised precision in heat pump fault detection.

- **XGBoost consistently outperforms GEMMA-3**, particularly on rare faults such as `defrosting_issue` and `driver_temp_error`, achieving high F1-scores across all classes – even when trained on natural language prompts originally designed for LLMs. This underscores the effectiveness of supervised learning when annotated data is available, regardless of input format.
- **GEMMA-3 enables fully label-free fault detection**, operating in a zero-shot regime using structured prompts and visualizations. However, it struggles with class imbalance and underperforms on minority fault types, limiting its current practical utility.
- **LLM-based diagnostics remain promising**, especially for low-resource or rapidly evolving environments. Hybrid strategies that integrate LLMs with few-shot exemplars, domain knowledge, or statistical post-processing may help overcome zero-shot limitations.

These results emphasize that while foundation models enable flexible deployment without labeled data, they are not yet reliable for safety-critical diagnostics. Improving multimodal prompt design and combining statistical learning with LLM reasoning are key directions for future research.

4.7 Future Directions

Out current framework already incorporates multimodal prompts – combining structured text with visual representations such as heatmaps, time series, and histograms. Future work could focus on leveraging these features more effectively. Enhancing prompt engineering with explicit fault indicators and temporal patterns, as well as introducing few-shot learning with annotated fault cases, may help address the low recall for rare fault types. In addition, hybrid approaches that combine LLM-based reasoning with supervised or statistical models could further improve robustness and accuracy. These directions aim to bridge the remaining gap between zero-shot generalization and reliable fault detection in complex, real-world cyber-physical systems.

In summary, supervised learning with XGBoost yields balanced detection across all fault types, while GEMMA-3's zero-shot, prompt-based classification is largely limited to majority-class detection and fails to generalize to rare faults. These findings highlight the fundamental trade-off between the flexibility of zero-shot LLMs and the reliability of supervised models, motivating the improvements discussed in the next section.

5 Discussion and Limitations

Our results reveal a core trade-off in fault detection for cyber-physical systems: supervised models like XGBoost achieve high accuracy across all classes (macro-F1 = 0.69) by leveraging labeled data, while GEMMA-3 enables interpretable, multilingual zero-shot diagnostics (macro-F1 = 0.24) without requiring any labeled faults. However, GEMMA-3 suffers from majority-class bias and limited recall for rare anomalies.

This performance gap underscores the limitations of zero-shot prompting when fault-specific cues are weak or absent. Nonetheless, GEMMA-3 delivers natural language explanations and can be deployed in settings where labeled data is scarce or unavailable, making it a viable foundation for low-label diagnostic pipelines.

To improve zero-shot fault detection, we propose three extensions:

1. **Enhance prompt engineering** with explicit temporal and fault-indicative features.
2. **Introduce few-shot prompts** to provide representative fault exemplars during inference.
3. **Incorporate retrieval-augmented generation (RAG)** using domain-specific corpora (e.g., technical manuals, field logs) to support root cause explanation.

In this study, we excluded RAG and fine-tuning to isolate the effects of prompt structure and visual context. This allows us to establish GEMMA-FD as a reproducible zero-shot baseline and benchmark for future hybrid approaches.

We also acknowledge the limitation of evaluating on a proprietary dataset. Future work will expand to public HVAC datasets (e.g., UCI SECOM, AHU) and incorporate unsupervised methods (e.g., Isolation Forest, LSTM Autoencoders) to assess generalizability and complement LLM reasoning.

Our findings suggest that structured prompting with visual encodings provides a viable entry point for fault detection in low-label regimes – but bridging the gap to robust deployment will require hybrid approaches that combine LLMs with symbolic models, supervision, or retrieval.

6 Conclusion and Outlook

We introduced **GEMMA-FD**, a zero-shot, prompt-based framework for fault detection in heat pumps using open-weight, multimodal large language models. By converting sensor data windows into structured text and visual inputs, our approach enables interpretable fault classification without labeled anomalies or expert rules.

While GEMMA-3 shows promise as a flexible, label-free diagnostic tool, it underperforms supervised methods like XGBoost in precision and recall, particularly for rare fault types. This illustrates the core trade-off between generalization and accuracy in CPS fault detection.

Looking forward, we see strong potential in hybrid approaches that combine the interpretability and accessibility of LLMs with the precision of supervised models. Enhancements such as few-shot adaptation, prompt ensembling, and retrieval-augmented generation can bridge current gaps and improve fault isolation and explanation in real-world deployments.

Ethical Statement

This research does not involve any ethical concerns or conflicts of interest.

References

1 Sarah Alnegheimish, Linh Nguyen, Laure Berti-Equille, and Kalyan Veeramachaneni. Can Large Language Models be Anomaly Detectors for Time Series? In *2024 IEEE 11th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, October 2024. doi:10.1109/DSAA61799.2024.10722786.

2 Andreas Auer, Patrick Podest, Daniel Klotz, Sebastian Böck, Günter Klambauer, and Sepp Hochreiter. TiRex: Zero-Shot Forecasting Across Long and Short Horizons with Enhanced In-Context Learning, May 2025. doi:10.48550/arXiv.2505.23719.

3 Elaheh Bazdar, Fuzhan Nasiri, and Fariborz Haghighat. Optimal planning and configuration of adiabatic-compressed air energy storage for urban buildings application: Techno-economic and environmental assessment. *Journal of Energy Storage*, 76:109720, January 2024. doi:10.1016/j.est.2023.109720.

4 Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pages 10148–10167, Vienna, Austria, July 2024. JMLR.org.

5 Hongliang Fei, Younghun Kim, Sambit Sahu, Milind Naphade, Sanjay K. Mamidipalli, and John Hutchinson. Heat pump detection from coarse grained smart meter data with positive and unlabeled learning. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1330–1338, New York, NY, USA, August 2013. Association for Computing Machinery. doi:10.1145/2487575.2488203.

6 Birgit Hofer and Franz Wotawa. Detecting Soft Faults in Heat Pumps (Short Paper). *OASIcs, Volume 125, DX 2024*, 125:22:1–22:10, 2024. doi:10.4230/OASICS.DX.2024.22.

7 Herbert Muehlburger and Franz Wotawa. FaultLines - Evaluating the Efficacy of Open-Source Large Language Models for Fault Detection in Cyber-Physical Systems*. In *2024 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pages 47–54, July 2024. doi:10.1109/AITest62860.2024.00014.

8 Anjan Rao Puttige, Staffan Andersson, Ronny Östin, and Thomas Olofsson. Application of Regression and ANN Models for Heat Pumps with Field Measurements. *Energies*, 14(6):1750, January 2021. doi:10.3390/en14061750.

9 Md Mahbubur Rahman, Reza Malekian, and Vilhelm Akerstroem. Fault Detection On Heat Pump Operational Data Using Machine Learning Algorithms. In *2024 11th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 204–211, September 2024. doi:10.1109/IOTSMS62296.2024.10710259.

10 Naeem Seliya, Taghi M. Khoshgoftaar, and Jason Van Hulse. A Study on the Relationships of Classifier Performance Metrics. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 59–66, November 2009. doi:10.1109/ICTAI.2009.25.

11 Ji-Hyun Shin and Young-Hum Cho. Machine-Learning-Based Coefficient of Performance Prediction Model for Heat Pump Systems. *Applied Sciences*, 12(1):362, January 2022. doi:10.3390/app12010362.

12 Cem Ekin Sunal, Vladimir Dyo, and Vladan Velisavljevic. Review of Machine Learning Based Fault Detection for Centrifugal Pump Induction Motors. *IEEE Access*, 10:71344–71355, 2022. doi:10.1109/ACCESS.2022.3187718.

13 Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, and Ramona Merhej. Gemma 3 Technical Report, March 2025. doi:10.48550/arXiv.2503.19786.

14 Andreas Weigert, Konstantin Hopf, Nicolai Weinig, and Thorsten Staake. Detection of heat pumps from smart meter and open data. *Energy Informatics*, 3(1):21, October 2020. doi:10.1186/s42162-020-00124-6.

15 Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy, June 2022. doi:10.48550/arXiv.2110.02642.

16 Wei Yin, Guo-qing Wang, Wan-sheng Miao, Min Zhang, and Wei-guo Zhang. Semi-supervised learning of decision making for parts faults to system-level failures diagnosis in avionics system. In *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, pages 7C4–1–7C4–14, October 2012. doi:10.1109/DASC.2012.6382418.

Beyond Dynamic Bayesian Networks: Fusing Temporal Logic Monitors with Probabilistic Diagnosis

Chetan Kulkarni
KBR Inc, NASA Ames Research Center, Moffett Field, CA, USA

Johann Schumann
KBR Inc, NASA Ames Research Center, Moffett Field, CA, USA

Abstract

Conventional diagnostic systems often fail to account for temporal dynamics – such as duration, frequency, or sequence of events – which are critical for accurate fault assessment. Existing solutions that model time, like Dynamic Bayesian Networks (DBNs), typically suffer from computational complexity and scalability issues.

This paper introduces a hybrid diagnostic architecture that integrates a standard Bayesian Networks (BNs) with a powerful temporal reasoner R2U2 (Realizable Responsive Unobtrusive Unit). By decoupling temporal logic from probabilistic inference, our approach allows the specialized R2U2 engine to efficiently process complex time-dependent conditions and provide nuanced inputs to the BNs. The result is a more scalable, flexible, and robust framework for diagnosing failures in systems where temporal behavior is a key factor. The paper will detail this architecture, its generation from system models, and demonstrate its capabilities using a UAV electric powertrain example.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics; Computing methodologies → Temporal reasoning; Mathematics of computing → Probabilistic inference problems; Computing methodologies → Multiscale systems; Mathematics of computing → Bayesian networks

Keywords and phrases Bayesian diagnostic network, temporal logic, fault diagnosis, temporal reasoning, probabilistic inference, scalability

Digital Object Identifier 10.4230/OASIcs.DX.2025.13

Category Short Paper

Acknowledgements This work was authored by employees of KBR Wyle Services, LLC under Contract No. 80ARC020D0010 with the National Aeronautics and Space Administration. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, or allow others to do so, for United States Government purposes. All other rights are reserved by the copyright owner.

1 Introduction

The ability to diagnose and resolve system malfunctions at their source is fundamental to the reliability of any complex domain, a critical capability delivered by a robust diagnostics framework. Although the term is frequently used interchangeably with Fault Detection and Isolation (FDI), diagnostics is a distinct discipline; whereas FDI is concerned with the identification and localization of a fault, diagnostics endeavors to determine the precise nature and underlying cause of said failure. A comprehensive array of methodologies has been developed to this end, spanning from elementary D-matrices that map test outcomes to failure modes, to more sophisticated probabilistic models like Bayesian Networks (BNs)

and data-driven techniques such as Deep Neural Networks (DNNs). These approaches are broadly classifiable as either model-based, which depend on an explicit representation of the system, or model-free, which derive their logic from empirical data. Moreover, the field of diagnostics is intrinsically linked with system safety and reliability disciplines, such as Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA), frequently serving as the mechanism for the practical detection of failure modes identified therein.

A considerable limitation inherent in many conventional diagnostic methodologies is their static character; while effective in representing the direct causal relationships between signals and failure modes, they inadequately incorporate temporal dynamics. These temporal considerations are, however, often indispensable for an accurate diagnosis. For example, a transient voltage fluctuation may be considered insignificant, yet the same condition constitutes a valid fault if it persists for a specified minimum duration. Likewise, subsequent to a change in a system's operational mode, certain error conditions may be anticipated and must be disregarded for a defined interval to preclude erroneous fault indications. Other temporal patterns, such as the occurrence frequency of anomalous data packets or the specific sequence of events, can alter the diagnostic output. Although solutions like Dynamic Bayesian Networks (DBNs) attempt to model these time-dependent relationships, their application is often hindered by computational complexity and model size, which increases as the network is unrolled over time, and they exhibit limitations in efficiently managing disparate temporal intervals.

To surmount these challenges, a hybrid R2U2/BN diagnosis system is proposed for advanced model-based temporal diagnosis. This framework integrates three principal components: a standard diagnostic Bayesian Network, a high-capability temporal reasoner (R2U2), and the potential for incorporating requirements specified in natural language. The novelty of this approach resides in the symbiotic integration wherein the BN's inputs (test signals) and its outputs (health-state nodes) serve as inputs for the R2U2 reasoner. This reasoner, in turn, supplies temporally processed information to the BN. This architecture facilitates robust temporal pre-processing, enabling the formulation of complex logical conditions, or "temporal tests," such as "the voltage remained below a threshold for at least 10 seconds" or "the lidar system's health metric was below 0.7 for a minimum of 2 seconds." The R2U2 component employs synchronous observers for Future Time Logic (FTL) that utilize a three-valued logic (true, false, maybe). When propagated to the BN, this three-valued input provides more nuanced insights for diagnostic reasoning, thereby overcoming the scalability and interval-management deficiencies characteristic of DBNs.

The motivation for this hybrid architecture arises from the inherent limitations of DBNs. By embedding temporal relationships directly into the probabilistic graph, DBNs suffer from state-space explosion and computational intractability, which complicates modeling sophisticated temporal patterns. The proposed R2U2/BN system provides a more scalable solution by decoupling temporal reasoning from probabilistic inference. This separation of concerns allows the Bayesian Network to remain a compact model of causal failures, while the specialized R2U2 reasoner efficiently handles the full expressive power of temporal logic. The resulting framework is therefore more flexible and robust for diagnosing systems where complex temporal dynamics are a critical component of the process.

The rest of this paper systematically develops our proposed diagnostic architecture. Section 2 provides the necessary background on diagnostic Bayesian networks and the R2U2 monitoring engine. With this foundation, Section 3 introduces our novel methodology, explaining how we combine temporal monitors with Bayesian networks and how these models are generated from FMEA. We then provide context by comparing our work to existing research in Section 4, before offering a final summary and outlining future research avenues in Section 5.

2 Background

2.1 Diagnostic Bayesian Networks

As defined by Pearl [22], Bayesian Networks (BNs) are directed acyclic graphs that model causal relationships between variables. In this framework, the nodes represent the variables themselves, while the connecting arcs point from a cause to its direct effect. The probabilistic strength of these causal links is captured by conditional probabilities. The example below, also from [22], serves as a practical illustration of a BN in action.

Figure 1 shows a representative BN, where the complete joint probability distribution $p\left(x_1, x_2, x_3, x_4, x_5, x_6\right)$ is the product of the conditional probabilities of each proposition given its ancestors, Eq. (1).

$$p\left(x_1, \ldots, x_6\right) = p(x_6|x_5)\, p(x_5|x_2, x_3)\, p(x_4|x_1, x_2, x_3)\; p(x_3|x_1)\, p(x_2|x_1)\, p(x_1) \tag{1}$$

The joint probability distribution could also be expressed with the following notation:

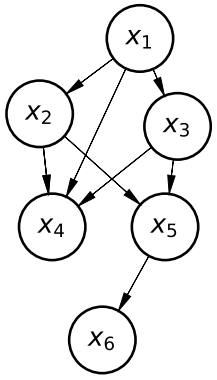


Figure 1 Example of directed acyclic graph used to create a Bayesian network.

$$p\left(\boldsymbol{x}\right) = \prod_{j=1}^{n} p\left(x_j|\boldsymbol{a}_j\right)\,, \tag{2}$$

where $\boldsymbol{a}_j$ represents the set of ancestors of variable x_j, and $\boldsymbol{x}$ is the random vector containing all variables $x_1, \ldots, x_n$ [23, 2]. For example, the term $p(x_4|x_1, x_2, x_3)$ becomes $p(x_4|\boldsymbol{a}_4)$.

Dependencies among propositions are described through the definition of sets of ancestors (or parents) and descendants (or children). For example, the set $\{x_1, x_2, x_3\}$ contains the ancestors of x_4, while $\{x_2, x_3\}$ contains the children of x_1. This structural model allows analysis over interventions, i.e., enable the computation of the joint probability density function (pdf) conditioned on some specific assumptions over a specific variable in the network [23]. Starting from the example in Figure 1, it is possible to evaluate the joint pdf given, e.g., x_2 has been defined *True*:

$$\begin{aligned} p_{X_2=1}\left(x_1, x_3, \ldots, x_6\right) = & p(x_6|x_5)\, p(x_5|X_2 = 1, x_3) \\ & p(x_4|x_1, X_2 = 1, x_3)\, p(x_3|x_1) \\ & p(x_1)\,. \end{aligned} \tag{3}$$

A key challenge in applying Bayesian Networks is the effort required to assess all conditional probabilities. Each node in the network needs a Conditional Probability Table (CPT) that defines its state based on every possible combination of its parents' values, meaning the

table's size grows combinatorially with the number of parents. For instance, while we can represent an intervention like forcing $X_2 = 1$ by simply removing the edge from its parent x_1 (as its state no longer depends on x_1, see Eq. (3)), the initial construction of such dependency tables for networks with many interconnected nodes remains a significant practical obstacle.

2.2 The R2U2 Monitoring Engine

The real-time R2U2 (Realizable, Responsive, and Unobtrusive Unit) has been developed to continuously monitor system and safety properties of an aerospace system. R2U2 has been implemented as an FPGA configuration [10]. and a software component. Hierarchical and modular models within this framework [27, 28] are defined using Metric Temporal Logic (MTL) [13] and mission-time Linear Temporal Logic (LTL) [25] for expressing Boolean formulas and temporal properties. In the following, we give a high-level overview over the R2U2 framework and its implementation. For details on temporal reasoning, its implementation, and semantics the reader is referred to [25, 10, 28].

2.2.1 Temporal Logic Observers

LTL and MTL formulas consist of propositional variables, the logic operators $\wedge$, $\vee$, $\neg$, or $\rightarrow$, and temporal operators to express temporal relationships between events.

R2U2 is capable of handling formulas for the past-time fragment of temporal logic as well as future time. Below, we briefly describe future-time operators and their informal semantics. For LTL formulas p, q, we have $\Box p$ (Always p), $\Diamond p$ (Eventually p), $\mathcal{X} p$ (NextTime p), $p \mathcal{U} q$ (p Until q), and $p \mathcal{R} q$ (p Releases q). Their formal definition and concise semantics is given in [25]. On an informal level, given Boolean variables p, q, the temporal operators have the following meaning (see also Figure 2):

- Always p ($\Box p$) means that p must be *true* at all times along the timeline.
- Eventually p ($\Diamond p$) means that p must be *true* at some time, either now or in the future.
- NextTime p ($\mathcal{X} p$) means that p must be *true* in the next time step; in this paper a time step is a tick of the system clock aboard the UAV.
- p Until q ($p \mathcal{U} q$) signifies that either q is *true* now, at the current time, or else p is *true* now and p will remain *true* consistently until a future time when q must be *true*. Note that q must be *true* sometime; p cannot simply be *true* forever.
- p Releases q ($p \mathcal{R} q$) signifies that either both p and q are *true* now or q is *true* now and remains *true* unless there comes a time in the future when p is also *true*, i.e., $p \wedge q$ is *true*. Note that in this case there is no requirement that p will ever become *true*; q could simply be *true* forever. The Release operator is often thought of as a "button push" operator: pushing button p triggers event $\neg q$.

For MTL, each of the temporal operators are accompanied by upper and lower time bounds that express the time period during which the operator must hold. Specifically, MTL includes the operators $\Box_{[i,j]}\ p$, $\Diamond_{[i,j]}\ p$, $p\ \mathcal{U}_{[i,j]}\ q$, and $p\ \mathcal{R}_{[i,j]}\ q$, where the temporal operator applies over the interval between time i and time j, inclusive (Figure 2).

Additionally, we use a *mission bounded* variant of LTL [25] where these time bounds are implied to be the start and end of the mission of a UAV. Throughout this paper, time steps corresponds to ticks of the system clock. So, a time bound of $[0, 7]$ would designate the time bound between 0 and 7 ticks of the system clock from now. Note that this bound is relative to "now" so that continuously monitoring a formula $\Diamond_{[0,7]}\ p$ would produce *true* at every time step t for which p holds anytime between 0 and 7 time steps after t, and *false* otherwise.

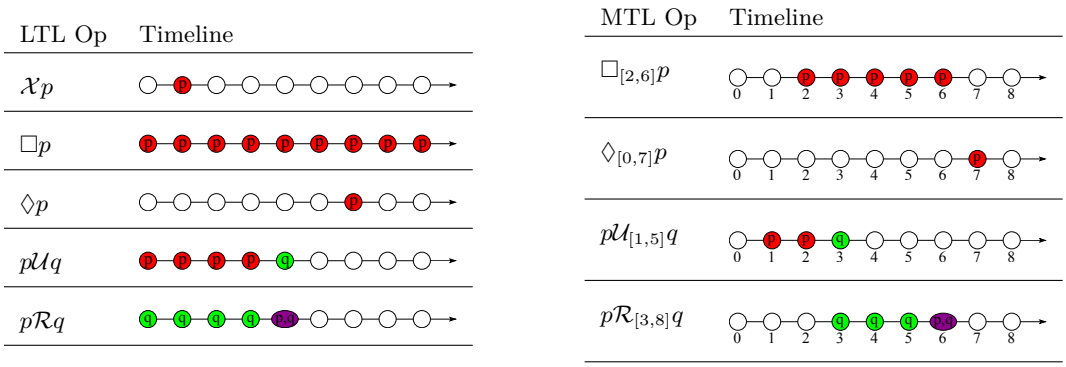


Figure 2 Pictorial representation of LTL temporal operators and MTL operators.

3 Temporal Bayesian Diagnosis with Temporal Logic Monitors

Our approach for diagnostic reasoning with temporal elements is based upon the synergistic combination of an efficient reasoning algorithm for Bayesian networks and the R2U2 temporal engine.

3.1 Tool Architecture and Modeling Process

Figure 3 gives a high-level overview of the architecture for our approach. Input signals S_t, which are obtained at time point t from sensors, software sensors, the system status, and outputs of our diagnostic BN are first going through a signal processing stage, where the values, usually floating point numbers, are scaled and subjected to thresholding to obtain Boolean values. In our architecture, the thresholds and range limits are model-based and assumed to be fixed. For example, a floating point signal U_{batt} might be thresholded using $U_{batt} < 18V$ to obtain the Boolean value `Ubatt_low`. Signal processing is carried out with a fixed basic rate, e.g., 10Hz.

The vector of Boolean values B_t for time t are now input to the R2U2 monitoring engine. Here, formulas in past-time and future-time logic are evaluated to yield Boolean or 3-valued (see Section 3.5.1) valuations V_t of each of the formulas at t. These outputs comprise the output of our diagnostic system and are also fed into the diagnostic BN.

For example, the formula $\Box_{[60s]}$`Ubatt_low` could correspond to a failure-mode "battery-too-weak-unusable". On the other hand, a formula $\Box_{[2s]}$`Ubatt_low` would filter out short drops or glitches of the battery voltage, and the result would be used as a value for an observable sensor node in the BN.

At each point in time t, the BN is evaluated and posterior probabilities are calculated. These probabilities for selected nodes correspond to the presence of failure modes or can correspond to component health; these values are direct outputs for our diagnosis system.

These posterior probabilities can also provide valuable information, when considered over time: for example, knowing if subsystem C has a poor health $H(C) < 0.3$ for an extended period of time. Therefore, selected values are fed back into the signal processing to be able to formalize such temporal properties.

Our proposed diagnostic architecture is purely model-driven and contains no machine-learning elements. The process steps for configuring and tailoring our tool is outlined in Figure 4. We derive diagnostic information from a comprehensive suite of models that are developed during the design phase of a complex and potentially autonomous system (see

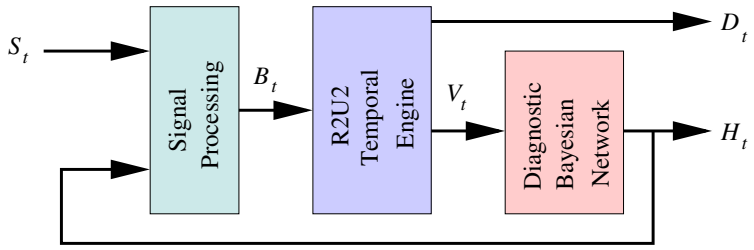


Figure 3 High-level representation of our diagnostics architecture.

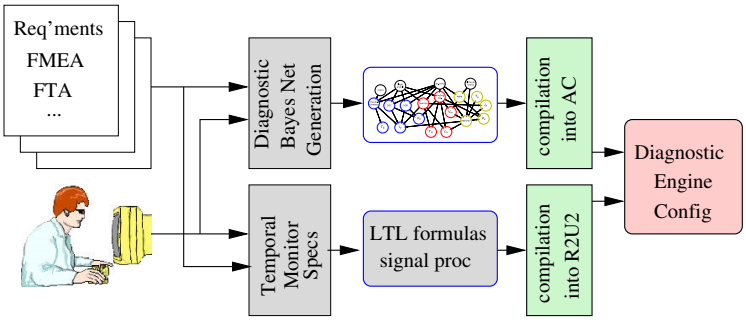


Figure 4 Development of the configuration for our hybrid diagnostic engine: The diagnostic BN and the temporal monitors are developed and generated from system requirements, FMEA, Fault trees, and other models and documentaton. After compilation for efficient execution both parts are merged into the machine-executable configuration.

Table 1 for an overview). Most notably, functional decomposition models, fault trees, and the outputs of a Failure Mode and Effects Analysis (FMEA) are used to automatically construct the diagnostic Bayesian Network. The probabilities for the Bayesian transitions are informed by a combination of metrics, including Mean Time To Failure (MTTF) and Mean Time Between Failures (MTBF), as well as operational conditions, system health, and subject matter expert (SME) expertise.

These models, together with system and component requirements, Concept of Operations (ConOPS), and safety/performance requirements are used to define the logic and temporal monitors for R2U2. In the following sections, we describe in detail how the BN and temporal monitors are constructed, before we discuss a realistic example.

3.2 BN Construction from Failure Mode and Effects Analysis (FMEA)

To develop an effective system-level diagnosis framework under uncertain conditions, we integrate information from a Failure Mode and Effects Analysis (FMEA) with a Bayesian Network (BN). This process unfolds in two main stages: first, we construct the graphical structure of the BN, and second, we define its conditional probability tables (CPTs) which quantify the relationships between different failure events.

The initial stage involves building the network's structure by translating the qualitative FMEA data into the BN's components. Then systematically convert the identified causes, failure modes, and effects from the FMEA into nodes within the network. This includes creating "observable" nodes that represent real-world sensor data – such as a high-temperature reading – which are critical for identifying and isolating the root cause of a failure. The

Table 1 Types of models typically used for an autonomous aerospace system like a UAV (from [26]).

Model	Context	Description
Effect	Fault Propagation Model	Functions cross referenced in fault propagation model. This is used to capture the functional degradation due to the presence of one or more fault(s).
Implementation	Functional Decomposition Model	Reference to blocks in the system model that implement one or more function(s) in the functional decomposition model.
Ambiguity Set	Fault Impact, Refinement Model	Represents a set of faults which cannot be distinguished based on the triggering Tests.
Operational Impact	Fault Impact, Recovery Plan	Reference to system variables and changes in their operating range.
Requirement	Active Diagnosis Procedures, Recovery Plan	Conditions on system variables in order to execute active diagnosis procedures or recovery plans.
Mode Requirement	Active Diagnosis Procedures, Recovery plan	Conditions on system modes in order to execute active diagnosis procedures or recovery plans.
TestRefs	Active Diagnosis Procedures	Additional tests that can be evaluated when an active diagnosis procedure is executed.
Trigger Condition	Recovery Plan	A set of faults related to triggering a recovery plan. Any of the faults in the triggering condition may be handled using the recovery plan.
Mode Change	Recovery Plan	Mode change introduced by executing a recovery plan.

network is then assembled by drawing directed arcs that reflect the causal logic from the FMEA, pointing from causes to their resulting failure modes and from failure modes to their ultimate effects. A simple example of a BN structure build from FMEA of an UAV electric powertrain is shown in Fig.5 for illustration.

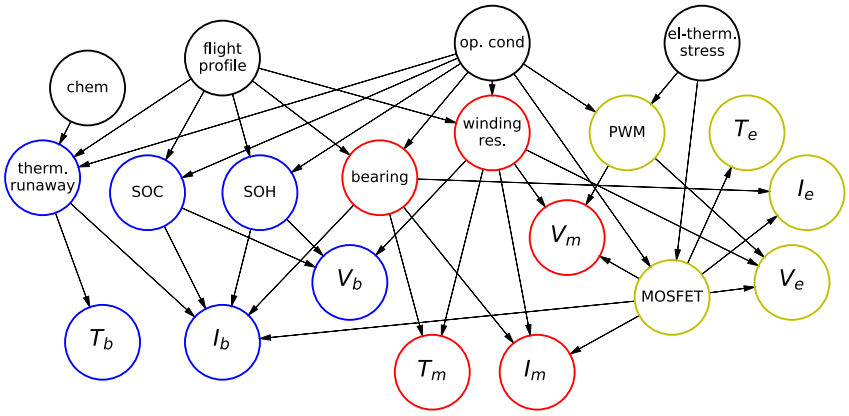


Figure 5 Example of a simple BN structure build from FMEA of UAV electric powertrain.

Table 2 Example of conditional probability table for a fault event with two known root causes.

x_1	x_2	f	
		0	1
0	0	$p_{0\|0\,0}$	$p_{1\|0\,0}$
0	1	$p_{0\|0\,1}$	$p_{1\|0\,1}$
1	0	$p_{0\|1\,0}$	$p_{1\|1\,0}$
1	1	$p_{0\|1\,1}$	$p_{1\|1\,1}$

With the Bayesian Network (BN) structure in place, we then assign probabilities to each node. First, root nodes (those with no parent nodes) are given prior marginal probabilities. Next, every other node is assigned a conditional probability based on its parent nodes, using information derived from the FMEA. To determine these values, we can draw from several sources, including historical failure data, the expertise of engineers, or established techniques such as maximum entropy theory [11].

Once the BN is fully defined, it can be used to detect and localize a fault within a complex system by turning observable nodes to *True* or *False*. The process starts when an observable evidence node, or "symptom," is triggered – for example, when a sensor value exceeds a set threshold. This new evidence is fed into the network, which then uses Bayesian inference to update the probability of every potential root cause. The cause that emerges with the highest failure probability is identified as the source of the fault. We will address potential complicating factors, such as environmental conditions and false alarms, in the next section.

3.3 Detailed Modeling Approach and Issues

The dependency among elements in FMEAs do not have to be restricted to deterministic relationships in BNs [2], and this property intrinsically enhances the modeling of the diagnostic system. Let us consider, for simplicity, a fault event f with two root causes, its ancestors, x_1 and x_2. Table 2 is the conditional probability table of the model, where probabilities are defined through three binary subscripts $i, j, k \in \{0, 1\}$. The term $p_{k|i\,j}$ defines the probability of the outcome k given values i, j, with k referring to the fault event f and i, j referring to its ancestors x_1 and x_2. For example, $p_{1|0\,0}$ is the probability that $f = 1$ given both ancestors x_1, x_2 are 0 (or *False*).

The fault event may happen, with low probability, because of external causes or unknown events not described by its ancestors. Such external forcing was called *Common Cause Failures* in [2], and following that idea, $p_{1|0\,0} \geq 0$, and so $p_{0|0\,0} = 1 - p_{1|0\,0}$. On the opposite side of the spectrum, the fault event may not happen even if both ancestors are activated (true). This option describes the ability of a system to work partially or reconfigure, [2], or describes a statistical relationship between the three elements, suggesting that root causes do not deterministically trigger the failure, so $p_{1|1\,1} < 1$. As a result, the two ancestors may occur without triggering the fault event, so $p_{0|1\,1} \geq 0$ and $p_{1|1\,1} = 1 - p_{0|1\,1}$. Different ancestors may influence the fault event in different ways, e.g. according to the severity of the root cause. This properties can be easily embedded in the network by assigning different values to the probabilities conditioned over $\{X_1 = 1, X_2 = 0\}$ and $\{X_1 = 0, X_2 = 1\}$.

In addition to the cases of failures induced by external variables or prevented system reconfiguration, the BN should also account for the performance of the measuring and/or detection system. In the proposed architecture, the evidence used to perform inference over the network is collected through sensors that measure variables connected (directly or

indirectly) to the fault event we aim to detect. The sensor performance or, similarly, the ability of the detection system to identify anomalous sensor data, should be embedded in the estimation of the CPT values. Reconnecting to the previous example, therefore, the element $p_{0|0\,0}$ in Table 2 should account for false alarm rates, and $p_{1|1\,1}$ should include, on top of any statistical relationship between the elements, the probability of mis-detection.

3.4 Efficient Evaluation of BN

Different BN inference algorithms can be used to compute a posterior probabilities. These algorithms include junction tree propagation [14, 12, 29] conditioning [8], variable elimination [17, 30], stochastic local search [21, 19], and arithmetic circuit evaluation [9, 5].

We select arithmetic circuit (AC) evaluation as our inference algorithm, which compiles our diagnostic BN into an arithmetic circuit. Especially for real-time aerospace systems, where there is a strong need to align the resource consumption of diagnostic computation to resource bounds [20, 18] algorithms based upon arithmetic circuit evaluation are powerful, as they provide predictable real-time performance [5]. An arithmetic circuit is a directed acyclic graph (DAG) in which the leaf nodes λ represent parameters and indicators while other nodes represent addition and multiplication operators. Figure 6 shows a small BN and its corresponding AC.

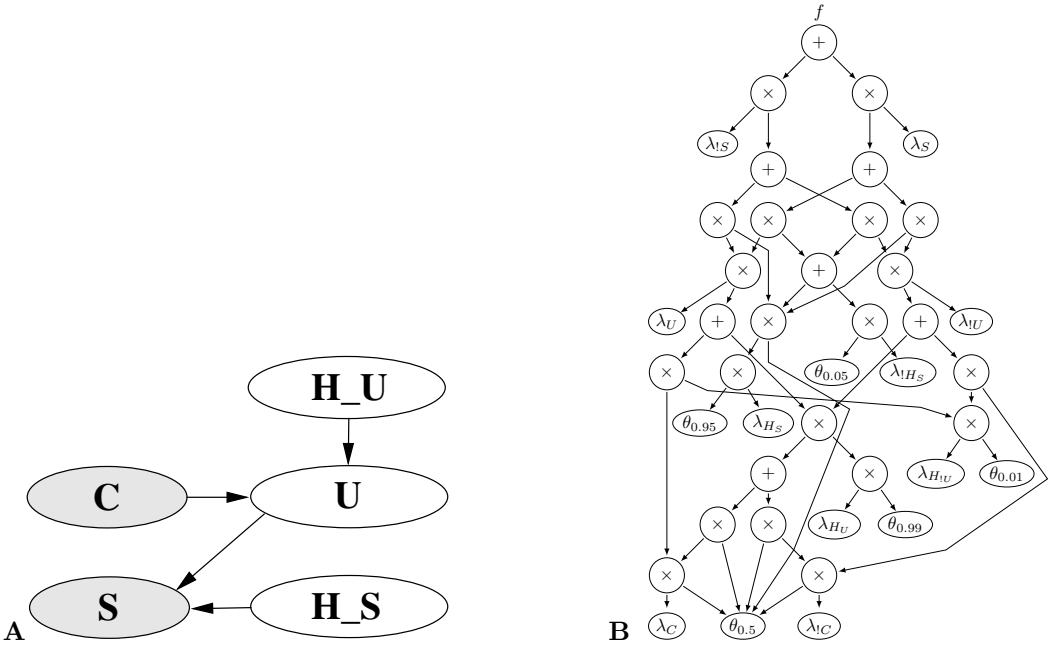


Figure 6 Small diagnostic Bayesian Network (A) and the corresponding arithmetic circuit (B) (from [3]).

Posterior marginals in a Bayesian Network can be computed from the joint distribution over all variables $X_i \in \mathcal{X}$:

$$p(X_1, X_2, \ldots) = \prod_{\lambda_x} \lambda_x \prod_{\theta x|\mathbf{u}} \theta_{x|\mathbf{u}}$$

where $\theta_{x|\mathbf{u}}$ are the parameters of the Bayesian network, i.e., the conditional probabilities that a variable X is in state x given that its parents $\mathbf{U}$ are in the joint state $\mathbf{u}$, i.e.,

$p(X = x|\mathbf{U} = \mathbf{u})$. Further, λ_x indicates whether or not state x is consistent with BN inputs or evidence. For efficient calculation, we rewrite the joint distribution into the corresponding network polynomial f [9]:

$$f = \sum_{\mathbf{x}} \prod_{\lambda_x} \lambda_x \prod_{\theta x|\mathbf{u}} \theta_{x|\mathbf{u}}$$

An arithmetic circuit is a compact representation of a network polynomial [7] which, in its in-compact form, is exponential in size and thus unrealistic in the general case. Hence, answers to probabilistic queries, including marginals and most probable explanations (MPEs), are computed using algorithms that operate directly on the arithmetic circuit. The marginal probability (see Corollary 1 in [9]) for x given evidence e is calculated as

$$Pr(x|\mathbf{e}) = \frac{1}{Pr(\mathbf{e})} \cdot \frac{\partial f}{\partial \lambda_x}(\mathbf{e})$$

where $Pr(\mathbf{e})$ is the probability of the evidence $\mathbf{e}$. In a bottom-up pass over the circuit, the probability of a particular evidence setting (or clamping of λ parameters) is evaluated. A subsequent top-down pass over the circuit computes the partial derivatives $\frac{\partial f}{\partial \lambda_x}$.

To evaluate the developed Bayesian Network (BN), we utilized the SamIam software package [6]. SamIam is a powerful, Java-based tool from UCLA that provides a comprehensive environment for modeling and reasoning with BNs. It includes a graphical user interface for building network models and a robust reasoning engine. This engine supports various critical functions, including classical inference, parameter estimation, sensitivity analysis (to assess how changes in one node affect the entire network), and the computation of Most Probable Explanations (MPE) [9, 7].

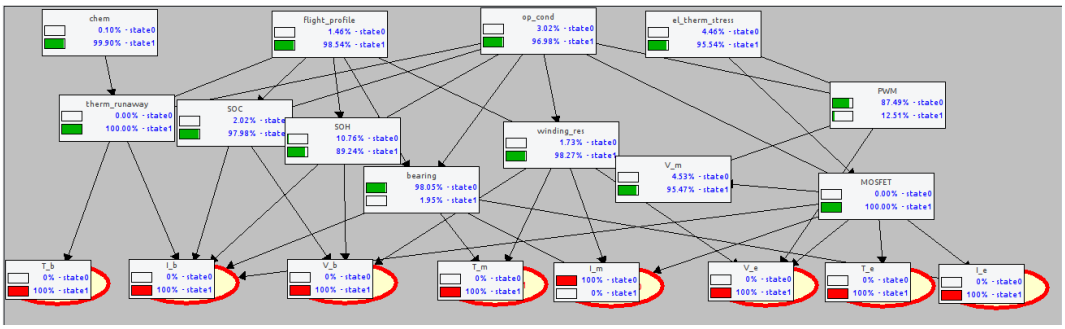


Figure 7 Evaluation of the BN using the tool SamIam [6].

We applied this framework to the powertrain BN, as shown in the schematic of Fig. 7. In our evaluation, the sensor nodes are observable and clamped to their current values. The resulting posterior probabilities of the other nodes are then calculated. For instance, in the scenario depicted, all sensor readings are nominal except for an abnormal motor current, I_m. The BN correctly infers a high posterior probability for the "bad motor bearing" fault mode, identifying it as the likely root cause. It is important to note that this specific example does not incorporate temporal monitors.

3.5 Defining Temporal Monitors

R2U2 monitors are used in our architecture to pre-process sensor signals (e.g., by thresholding) and to watch conditions in sensor signals and outputs of the BN over time. In order to allow simple signal processing, the input language for R2U2 can, in addition to Boolean Variables, contain arithmetic expressions, like $U_{batt} > 21.0V$.

For diagnostic purposes, there exist a number of different kinds of monitor patterns, including (see also Table 3):

Thresholding: such monitors perform simple signal processing tasks and determine if the current signal value is above or below a certain threshold, or is within a given range.

Persistency: a condition is considered to be persistent if it is consecutively true for at least n time steps: $\Box_n C$. Such formulas are used to filter out short dropout or nuisance signals.

Conditions: failure conditions might be only considered, when certain conditions hold, e.g., when the system is in a specific mode.

Transient blocking: upon a change in the system (e.g., a mode change), a failure condition is blocked during a certain temporal interval

Occurrence: these types of monitors can determine if a signal or event occurs more than n times within a given interval. This type of monitors can be used to trigger events based upon failure rates, e.g., there should not be more than 3 ill-formed data packets within a 10s interval.

Expectations: these kinds of formulas can be used to monitor if certain events occur before, during, or after a certain triggering event or condition. E.g., after touchdown, the engine RPM should be within 30 seconds below $10s^{-1}$.

Each of the variables of these formulas can be Boolean's obtained by thresholding sensor signals or the posterior probabilities of the failure-mode nodes of the diagnostic BN. This capability allows use to write monitors, which depend on diagnoses. E.g., if the health of a lidar sensor has been poor for at least the last minute, then only large measurement errors causes the triggering of a failure mode. Such formulas can be used to customize the diagnosis system based upon current diagnostic results and health status. If a subsystem has been diagnosed with a poor health, it might be necessary to tolerate larger error margins in order to avoid cascading of failure conditions.

Table 3 Typical temporal formulas for monitoring used in our example. Formulas are given in past time (PT) or future time logic (FT).

Formula	L	description
$V_b := \Box_{1min} U_{batt} < 12V$	PT	battery low voltage V_b test succeeds, if the battery voltage is lower than 12V continuously for at least the last minute
$MR := spin\_motor \rightarrow \Diamond_{10sec} \Delta_{RPM} > 100$	FT	when motor is started, we expect an increase in RPM of at least 100 within the next 10 seconds
$HT := \neg strong\_climb \wedge \Box_{1min} T > 200F$	PT	the motor should not be overly hot for a longer period of time except when in strong climb mode
$B := \neg batt\_overheat \wedge \neg\Diamond_{10min} \neg V_{batt} < 12V$	FT	the battery is not overheated and within the next 10 minutes, the battery voltage shall not fall below 12V
$LB := \Box_{1min} H(lidar) < 0.5$	PT	The LIDAR sensor is diagnosed as bad, if it had poor health for more than one consecutive minute
$crit := \Box_{10min} V_{batt} < 12V \wedge \Box_{1min} H(lidar) < 0.5$	PT	the battery voltage has been low for the last 10 minutes and the LIDAR sensor is unhealthy

3.5.1 Synchronous Observers

Besides the (asynchronous) temporal observers described in Section 3.5 for past time and future time linear temporal logic, R2U2 also provides *synchronous observers* for future time logic. Whereas the observers for ptLTL can be valuated at each point in time, future-time observers might need to be valuated at a later time, because R2U2 cannot look into the future. For example, the ftLTL formula $\Diamond_{[10s]}p$ cannot be valuated at the beginning of the interval, because it is not yet known if p will become true within the next 10 seconds. Thus, this formula can, in the worst case, only valuated after 10 seconds. R2U2 provides that information, but the use of ftLTL formulas for monitoring is not suitable for our application.

In contrast, *synchronous* observers can be valuated at each point in time. Introduced in [24], they return one of three possible values: `false, maybe, true`. Their highly efficient implementation in R2U2 makes them an ideal candidate for temporal monitors. In addition, the three-values logic values can be directly carried over to the diagnostic Bayesian network. Here, the observable sensor nodes now get a third state labeled "maybe".

Its usefulness for diagnostic reasoning with synchronous future temporal monitors becomes evident when we look at the following example. Encoding a monitor for "the UAV shall reach an altitude of 300ft within 20 seconds if the ESC status is OK" as

$$M := (ESC == ok) \wedge \Diamond_{[20s]}(alt > 300ft)$$

provides valuable information at any point in time. Whereas the asynchronous observer can only valuated after 20 seconds, the synchronous observer returns "maybe" even at the beginning of the interval, indicating that the UAV can still reach the required altitude. Only in the case, the ESC is not working, or the 20 second interval has passed, "false" is returned. This additional piece of instantaneous information can be used to improve the diagnosis and can also support autonomous decision-making: in the "maybe"-case, the flight might be continued if the additional risk can be justified, whereas the "false" will need to cause an abortion of the mission right away.

3.6 Practical Example

We illustrate our approach with a simple model of an electric powertrain for a UAS system. As shown in Figure 8A, the system consists of a li-ion battery B, the brushless dc motor M (only one shown here), and the electronic speed control ESC. For each of the components, we measure temperature T, voltage V and current I. In the high-fidelity simulator, which uses physical and electro-chemical models, measurements are obtained in 10 second intervals. Figure 8B shows the measurement signals which are typical for a nominal situation, where between $t = 900s$ and $t = 1800s$ the motor load is increased, leading to increased currents I and a slight rise in battery temperature. The battery voltage slowly decreases as the battery discharges as load decreases based on operational modes. Toward the end of the scenario, the battery is near-empty and discharges very quickly.

In the nominal scenario, voltages and currents at each component are the same. As shown in Figure 5, our diagnostic BN has input nodes for temperatures, currents, and voltages, and is capable of diagnosing battery-related issues (thermal runaway, low state-of-charge, low state-of-health)„ motor issues (bearing problems and change in the resistance of the motor winding), as well as in the electronic controller (power MOSFET electronics, or pulse-width-modulation (PWM) issues).

Figure 9 shows the original nominal signals for the battery, their discretizations with R2U2, as well as the posterior probabilities for the SOC and SOH nodes. Most significantly, a $V_{batt} < 21.0V$ is considered low battery voltage (`V_batt_low`) and `V_batt_nom` is a battery

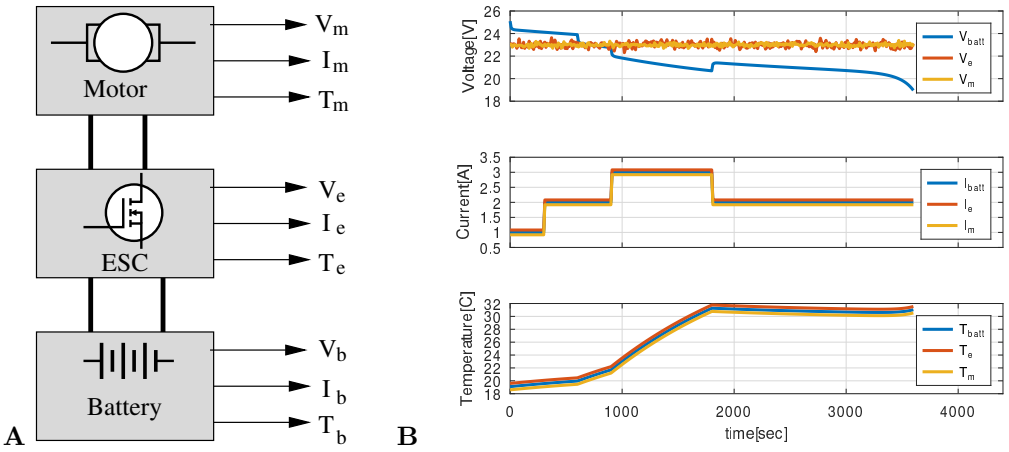


Figure 8 Electric Drive Train for a UAS: schematic (A), and sensor values for a nominal scenario (B).

voltage between 20V and 26V. The figure shows that with lower voltage through increased load, and through discharge, the BN nodes for SOC and SOH change, indicating a low state of charge. Toward the end of the scenario, the battery is drained and the voltage goes off-nominal. For this scenario, a simple BN is sufficient and no temporal operators are necessary. In the next scenario, however, we need to deal with noisy signals: the battery voltage signal can have drop-outs of up to 30 seconds. These dropouts, however, should not count toward a highly discharged battery. In our framework, we subject the battery voltage signal to the R2U2 formula

$$\texttt{V\_batt\_low} := \square_{0,30s}(V_{batt} < 21.0V)$$

The noise in Figure 9 has been eliminated properly. If we want to filter out longer-lasting periods of low voltage, like the time between $t = 900s$ and $t = 1800s$ in Figure 8B, the R2U2 formula just needs to be adjusted; the additional memory overhead for processing the temporal formula is minimal (just a ring buffer for 90 integer variables). This is in stark contrast to using a dynamic BN, which would require to replicate the basic network 90 times.

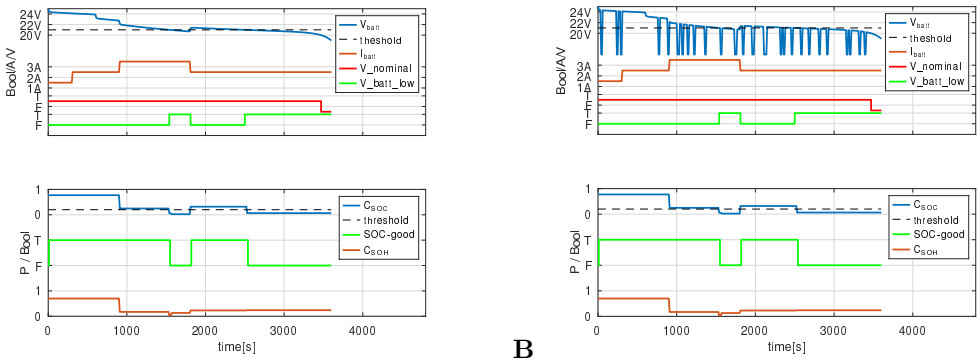


Figure 9 Nominal scenario without and with noisy V_{batt} signal. Temperature development (not shown) is identical to the curve shown in Figure 8B.

Figure 10 illustrates a diagnostic scenario in which the order of events is important. Initially, a flight is operating under nominal conditions. After some time, the motor current (I_m) begins to increase while the voltage (V_m) stays constant. In response, the Bayesian Network (BN) diagnoses a potential issue with the motor's bearing or winding. The resulting increase in friction and current then causes the motor temperature (T_m) to rise. Once T_m surpasses a critical threshold, the posterior probability for both "bearing" and "winding" failures becomes very high. This clearly points to a motor malfunction, but a static BN that fails to consider the sequence of events cannot differentiate between these two failure modes, leading to an ambiguous diagnosis.

In our framework, we add a past-time temporal formula to the output of the BN:

$$\text{Bearing} := (\square_{[0,10s]}(P_{bearing} > 0.7 \wedge P_{winding} > 0.7)\mathcal{S}_{[0,10]}(\square_{[0,20s]}P_{bearing} > 0.7)$$

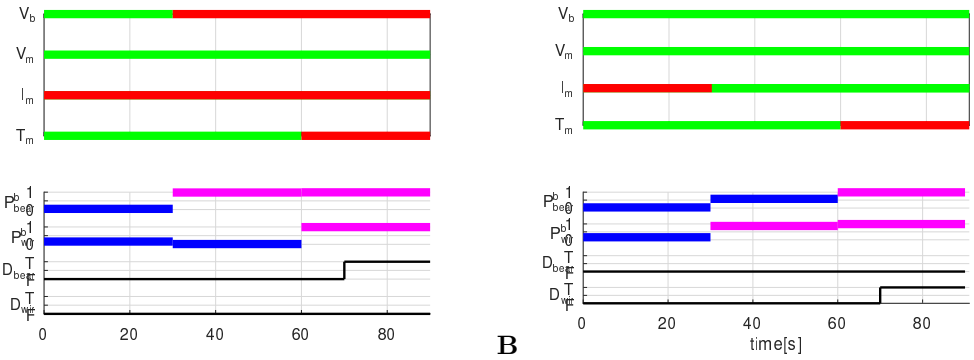


Figure 10 Failure scenarios (starting at t=30s). A: A faulty bearing causes an extended power draw (I_m high) and a drop in battery voltage. At t=60s, an overheating occurs (T_m too high). Probabilities for issues with the bearing and winding are high, making it impossible for the BN to find the correct cause. The R2U2 result shows the correct result; the delay caused by the $\square$ operator. B: failure of the winding. Again, the temporal formula is used to disambiguate the failure situation.

Only if an issue with the motor bearing has been flagged continuously for at least 100s and then, within 10 seconds, the bearing and the winding issues are flagged at the same time for at least 20 seconds, we can disambiguate the situation and infer that the bearing issue is the root cause. The 10 second grace time of the temporal "S" (Since) operator is used to minimize transient effects observed due to change in operational modes.

The symmetric R2U2 formula for the winding issue would look like:

$$\text{Winding} := (\square_{[0,10s]}(P_{bearing} > 0.7 \wedge P_{winding} > 0.7)\mathcal{S}_{[0,10]}(\square_{[0,20s]}P_{winding} > 0.7)$$

Figure 10 shows traces for both scenarios. Although a single DBN would be capable of modeling this situation, the complex temporal dependencies would require a large and complex DBN.

4 Related Work

Previous work has extensively utilized dynamic Bayesian networks (DBNs) for system diagnosis and reliability assessment. Addressing the shortcomings of traditional model-based approaches, Lerner et al. [15] proposed using a temporal causal graph (TCG) to structure a DBN for representing dynamic variable relationships. This DBN framework has been applied

to several domains. In the field of reliability engineering, Lewis et al. [16] demonstrated the use of DBNs for risk assessment and highlighted the importance of modeling the health state of complex systems. Other related research includes the work of Arocha et al. [1], who developed a method for identifying reasoning strategies in medical applications through cognitive-semantic analysis.

In contrast to our approach, which prioritizes network size and efficiency, the methodology presented by Cai et al [4]. illustrates the trade-offs inherent in using Dynamic Bayesian Networks (DBNs). Their work effectively addresses the challenge of diagnosing complex temporal faults – including transient, intermittent, and permanent failures – by explicitly modeling a system's dynamic degradation over time. To achieve this, their DBN employs Markov chains, which replicate the network structure across multiple time slices. While this allows them to classify fault types based on evolving posterior probabilities, this replication is precisely what leads to significantly larger and more computationally intensive models – a complexity our architecture is designed to avoid.

5 Conclusions

In this paper, we have introduced a powerful diagnostic system that successfully untangles temporal dynamics from probabilistic reasoning. Our key innovation – decoupling the R2U2 temporal monitoring engine from a static Bayesian Network (BN) – offers the best of both worlds: highly efficient evaluation of complex temporal conditions without the exponential complexity and network replication inherent in traditional Dynamic BNs. This ensures the core diagnostic model remains compact, transparent, and easy to maintain.

Looking ahead, our work will focus on making the developed methodology more accessible and robust for additional complex systems. The most critical next step is to streamline the modeling process itself. We will achieve this by integrating FRET, an open-source tool that automatically translates requirements written in structured natural language to formal temporal logic. This will dramatically accelerate development and eliminate the error-prone task of manual formula definition. Simultaneously, we will refine the automated generation of the Bayesian Network, deepening its integration with industry-standard modeling and analysis tools to create a seamless, end-to-end diagnostic framework.

References

1 Jose F Arocha, Dongwen Wang, and Vimla L Patel. Identifying reasoning strategies in medical decision making: a methodological guide. *Journal of biomedical informatics*, 38(2):154–171, 2005. doi:10.1016/J.JBI.2005.02.001.

2 A Bobbio, L Portinale, M Minichino, and E. Ciancamerla. Improving the analysis of dependable systems by mappping fault trees into bayesian networks. In *Reliability Engineering and Systems Safety*, volume 71, pages 249–260, 2001. doi:10.1016/S0951-8320(00)00077-6.

3 Borzoo Bonakdarpour and Scott A. Smolka, editors. *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, volume 8734 of *Lecture Notes in Computer Science*. Springer, 2014. doi:10.1007/978-3-319-11164-3.

4 Baoping Cai, Yu Liu, and Min Xie. A dynamic-bayesian-network-based fault diagnosis methodology considering transient and intermittent faults. *IEEE Transactions on Automation Science and Engineering*, 14(1):276–285, 2016. doi:10.1109/TASE.2016.2574875.

5 Mark Chavira and Adnan Darwiche. Compiling Bayesian networks using variable elimination. In *IJCAI*, pages 2443–2449. Morgan Kaufmann Publishers Inc., 2007. URL: http://ijcai.org/Proceedings/07/Papers/393.pdf.

6 A. Darwiche. SamIam: Sensitivity analysis, modeling, inference, and more. URL: http://reasoning.cs.ucla.edu/samiam/.

7 A. Darwiche. Modeling and reasoning with bayesian networks. In *Modeling and Reasoning with Bayesian Networks*, 2009.

8 Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001. doi:10.1016/S0004-3702(00)00069-2.

9 Adnan Darwiche. A differential approach to inference in Bayesian networks. *J. ACM*, 50(3):280–305, 2003. doi:10.1145/765568.765570.

10 Johannes Geist, Kristin Y. Rozier, and Johann Schumann. Runtime observer pairs and bayesian network reasoners on-board FPGAs: Flight-certifiable system health management for embedded systems. In *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, pages 215–230, 2014. doi:10.1007/978-3-319-11164-3_18.

11 Alvaro García Eduardo Gilabert. Mapping fmea into bayesian networks. *International Journal of Performability Engineering*, 7(6):525–537, 2011.

12 F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic network by local computations. *Computational Statistics*, Quarterly 4:269–282, 1990.

13 Ron Koymans. Specifying real-time properties with Metric Temporal Logic. *Real-time systems*, 2(4):255–299, 1990. doi:10.1007/BF01995674.

14 S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–224, 1988.

15 U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. In *Proc. of the Seventeenth national Conference on Artificial Intelligence (AAAI-00)*, pages 531–537, 2000. URL: citeseer.ist.psu.edu/lerner00bayesian.html.

16 Austin D Lewis and Katrina M Groth. A dynamic bayesian network structure for joint diagnostics and prognostics of complex engineering systems. *Algorithms*, 13(3):64, 2020. doi:10.3390/A13030064.

17 Zhaoyu Li and Bruce D'Ambrosio. Efficient inference in bayes networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11(1):55–81, 1994. doi:10.1016/0888-613X(94)90019-1.

18 O. J. Mengshoel. Designing resource-bounded reasoners using Bayesian networks: System health monitoring and diagnosis. In *DX*, pages 330–337, 2007.

19 Ole J. Mengshoel, Dan Roth, and David C. Wilkins. Portfolios in stochastic local search: Efficiently computing most probable explanations in Bayesian networks. *J. Autom. Reason.*, 46(2):103–160, 2011. doi:10.1007/S10817-010-9170-5.

20 D.J. Musliner, J.A. Hendler, A.K. Agrawala, E.H. Durfee, J.K. Strosnider, and C. J. Paul. The challenges of real-time AI. *Computer*, 28(1):58–66, 1995.

21 James D. Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. *J. Artif. Int. Res.*, 21(1):101–133, 2004. doi:10.1613/JAIR.1236.

22 J. Pearl. Bayesian networks: A model cf self-activated memory for evidential reasoning. In *7th Conference of the Cognitive Science Society*, 1985.

23 J. Pearl. *Causality: models, reasoning and inference*. MIT Press Cambridge, MA, 2000.

24 Thomas Reinbacher, Jörg Brauer, Martin Horauer, Andreas Steininger, and Stefan Kowalewski. Runtime verification of microcontroller binary code. *Sci. Comput. Program.*, 80:109–129, February 2014. doi:10.1016/j.scico.2012.10.015.

25 Thomas Reinbacher, Kristin Yvonne Rozier, and Johann Schumann. Temporal-logic based runtime observer pairs for system health management of real-time systems. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13,*

2014. Proceedings, volume 8413 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2014. doi:10.1007/978-3-642-54862-8_24.

26 Johann Schumann, Nagabhushan Mahadevan, Adam Sweet, Anupa R. Bajwa, Michael Lowry, and Gabor Karsai. Model-based System Health Management and Contingency Planning for Autonomous UAS. In *AIAA Scitech Forum*, 2019. URL: https://arc.aiaa.org/doi/pdf/10.2514/6.2019-1961.

27 Johann Schumann, Kristin Y. Rozier, Thomas Reinbacher, Ole J. Mengshoel, Timmy Mbaya, and Corey Ippolito. Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. In *Proceedings of the 2013 Annual Conference of the Prognostics and Health Management Society (PHM2013)*, October 2013.

28 Johann Schumann, Kristin Y. Rozier, Thomas Reinbacher, Ole J. Mengshoel, Timmy Mbaya, and Corey Ippolito. Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. *International Journal of Prognostics and Health Management*, 6(21):1–27, 2015.

29 Prakash P. Shenoy. A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, 3(5):383–411, 1989. doi:10.1016/0888-613X(89)90009-1.

30 Nevin Zhang and David Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178. Morgan Kaufmann, 1994.

The DX Competition 2025 and Its Benchmarks

Ingo Pill[1]
Institute of Software Engineering and Artificial Intelligence, TU Graz, Austria

Daniel Jung[2]
Department of Electrical Engineering, Linköping University, Sweden

Eldin Kurudzija[3]
Institute of Space Propulsion, German Aerospace Center (DLR), Köln, Germany

Anna Sztyber-Betley[4]
Warsaw University of Technology, Poland

Michał Syfert
Warsaw University of Technology, Poland

Kai Dresia
Institute of Space Propulsion, German Aerospace Center (DLR), Lampoldhausen, Germany

Günther Waxenegger-Wilfing
Institute of Space Propulsion, German Aerospace Center (DLR), Hardthausen am Kocher, Germany
Institute of Computer Science, University of Würzburg, Germany

Johan de Kleer[5]
c-infinity, Mountain View, CA, USA

Abstract

Fault diagnosis has been addressed in many research communities, leading to a variety of available fault diagnosis techniques. Deciding as a user which fault diagnosis methods are suitable for a specific application is thus a nontrivial task. Benchmarks can provide the community with a holistic understanding of the landscape of newly developed and available fault diagnosis methods when making this decision. After a long hiatus, we revived the DX Competition with three fault diagnosis benchmarks: SLIDe, LUMEN, and LiU-ICE. The purpose of the benchmarks is to inspire fault diagnosis research with challenging problems in cyber-physical systems relevant for industry. The benchmarks share a common code structure and we used similar performance metrics in order to simplify the adaptation of diagnosis system solutions to the different case studies.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics

Keywords and phrases Diagnosis, Algorithms, Evaluation

Digital Object Identifier 10.4230/OASIcs.DX.2025.14

Category DX Competition

Supplementary Material
Other (DXC'25 Homepage): `https://conf.researchr.org/home/dx-2025#Competition`
Other (DXC'25 Benchmarks, incl. Datasets and Instructions): `https://vehsys.gitlab-pages.liu.se/dx25benchmarks/`

[1] Chair DX Competition 2025
[2] Chair LiU-ICE Benchmark
[3] Chair LUMEN Benchmark
[4] Chair SLIDe Benchmark
[5] Co-Chair DX Competition 2025

36th International Conference on Principles of Diagnosis and Resilient Systems (DX 2025).
Editors: Marcos Quinones-Grueiro, Gautam Biswas, and Ingo Pill; Article No. 14; pp. 14:1–14:19
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We would like to thank all of our colleagues who contributed to making the DX Competition 2025 happen and who worked with us on the benchmarks. This includes in particular Erik Frisk, Mattias Krysander, Tobias Lindell, Tobias Traudt, Jan Deeken, Justin Hardi, Stefan Schlechtriem, Michael Börner, Dmitry Suslov, Robson dos Santos Hahn, Sebastian Klein, Wolfgang Armbruster, Jan Haemisch, Christopher Groll, Max Axel Müller, and Vincent Bareiß.

1 Introduction

Reasoning about the root causes for an encountered problem is a common task. Whenever an order is not delivered, our car does not start, a program does not work, our multimedia system stops working, when we feel ill, and in many other situations we are interested in the reasons why something was or is not working as expected. Only once we know the source of the problem can we begin to effectively address and solve it to mitigate the issue.

Diagnosis algorithms address this need, in that they tell us exactly which sets of malfunctioning parts in a system can explain the unexpected behavior. The corresponding approaches are sometimes dedicated to very specific scenarios [33, 38] and exploit specific aspects of a diagnostic problem [35], but most concepts are general enough to be applicable to a wide variety of systems. Whenever we refer to systems, we do so in the most abstract sense in that we mean actually any artifact that we can reason about. The system targeted by a diagnostic process might thus be digital, logical, analog, mechanical, cyber-physical, biological, ecological, ethical, and economical, or it could also refer to, for instance, a social system, a supply chain, or a process.

Thus, many research communities have been working on concepts and algorithms for fault diagnosis, and they have been doing so based on a diverse set of underlying techniques. This led to a large variety of available approaches that range from symbolic [42, 34, 7] to sub-symbolic [25, 1], hybrid [28] and statistical [5, 36] ones, and which potentially aim at diagnosing a single [34] or multiple [39] scenarios. In rare cases, they target even the isolation of all faults that are present in a system [37] by generating and considering data that are hopefully representative enough. Deciding as a user which fault diagnosis methods are suitable for a specific application is thus a nontrivial task, and it is certainly not as straightforward as it might look at first glance. In particular, we have to take into account that all methods come with their own individual ramifications in terms of resource expenditure, required knowledge, and achievable performance. Each solution is based on hidden assumptions, see, *e.g.* [20], which affect the quality of the results computed in a given scenario.

One solution for providing the community with a holistic understanding of the landscape of available and newly developed methods is the use of well-formulated benchmarks. They allow the scientific community to propose different solutions to diagnostic systems and draw on a well-founded comparison option to evaluate their performance. As we shall discuss in Section 2, there is a variety of such benchmarks available. Individual papers tend to use only a subset of those, usually in combination with paper-specific benchmarks. Furthermore, we have to take into account that the computation hardware as well as available tools (like SMT/SAT/constraint solvers or simulators) change significantly over time. All of these aspects make it hard to maintain an accurate picture of old and new proposals. So, when Reiter argued in his seminal paper [42] that solvers are too slow to search for diagnoses directly (so not taking conflicts into account), he did not anticipate the technological evolution that we have experienced since then and that allowed the advent of corresponding solutions [27] with very competitive performance [30].

After a long hiatus, we thus revived the DX Competition[6], which is an important source for evaluating new and old algorithms and putting their performance into perspective. In 2025, we started DXC'25 with a set of three benchmarks that focus on three individual cyber-physical systems. As we explain in individual sections, the three case studies[7] combine different diagnosis tasks, and they feature different properties, as summarized in Table 1.

Table 1 Characteristics of the DXC'25 benchmarks.

	SLIDe	LUMEN	LiU-ICE
application	steam line	rocket engine	combustion engine
docker container available	Y	Y	Y
real/artificial data	A	A	R
natural/injected faults	I	I	I
attacks	Y	N	N
intermittent faults	N	Y	N
discrete/continuous	C	C	C
fault data/system data/ simulator	FD, SD	FD, SD, SIM	FD, SD
challenges	diag, nonlinear	diag, nonlinear, sim2real	diag, nonlinear

As we can see from the table, the benchmarks focus on persistent faults and cyber-attacks in a variety of continuous nonlinear systems. For some, a simulator is available so that a user can also create their own behavioral samples. For others, real and/or artificial data are provided. All the technical details are available from our DXC'25 benchmark repository, so that a reader may test their own solutions for all the benchmarks described in this paper.

The outline of this paper is as follows. First, related research and other fault diagnosis benchmarks are discussed in Section 2. Then, presentations of each of the three DX benchmarks are given: including a system description, a presentation of considered fault scenarios, and provided resources. The case study SLIDe is presented in Section 3, LUMEN in Section 4, and LiU-ICE in Section 5. A description of the benchmark implementation environments is presented in Section 6 and the evaluation metrics used in the benchmarks are summarized in Section 7. Finally, a summary is given in Section 8.

2 Related research

Various fault diagnosis benchmarks have been proposed. In contrast to text or vision processing, technical fault diagnosis research still suffers due to data scarcity. This is mainly attributed to two causes. First, industrial datasets often cannot be shared due to confidentiality concerns. Second, fault diagnosis is a field of anomaly detection, where the number of normal samples is significantly larger than the number of faulty samples. Therefore, the community can still largely benefit from developing new benchmarks.

DX Community is grounded in logical and model-based approaches [42]. With the advances in machine learning, the community is integrating data-driven approaches. As pointed out in [56], industrial machine learning research (including fault diagnosis) must carefully follow principles to achieve reliable results. One of the crucial aspects is the use of

[6] `https://conf.researchr.org/home/dx-2025#Competition`
[7] available from `https://vehsys.gitlab-pages.liu.se/dx25benchmarks/`

held-out test sets. It makes competitions particularly useful for the evaluation of the proposed algorithms. The study in [19] showed that the performance of the solutions in the competition had a clear connection to the assumptions made about different faults in the design of the diagnostic system. Existing benchmarks, while helpful, carry the risk of overfitting to the test set. This effect was observed in the recent Safeprocess competition [17, 18] based on an internal combustion engine, where the results on the training data were generally overestimating the results on the held-out evaluation set.

There are several benchmarks that have been widely adopted in the fault diagnosis community. The Tennessee Eastman Process (TEP) [8] dataset is a simulation of a chemical process widely used for control and diagnosis research. DAMADICS benchmark [3] is a study of the intelligent industrial actuator. The CWRU dataset [45] serves for the comparison of fault diagnosis of rolling bearings. NASA Ames developed the Advanced Diagnostics and Prognostics Testbed [41] that has been used for benchmarks and competitions; see, e.g., [22]. NASA's Prognostics Data Repository[8] is a collection of datasets for prognostics and health management. It is a valuable resource for remaining useful life (RUL) prediction benchmarking. A simulation-based wind turbine benchmark is proposed for fault diagnosis and fault-tolerant control in [32]. The results of six participants in a competition using the wind turbine benchmark are summarized in [31].

Recently, a few benchmarks were proposed inside the DX community. A leak detection and localisation benchmark, including structural model of water distribution network, and simulated dataset, was proposed in [51, 50]. An ensemble of benchmarks based on simulated tank systems was presented in [2]. The set requirements for AI benchmarks in the domain of Cyber-Physical Production Systems were formulated in [11], additionally introducing a comprehensive benchmark, offering applicability on diagnosis, reconfiguration, and planning approaches. A TuLAUT (Theory and Teaching of Automation Technology) website[9] provides a curated collection of industrial datasets, including many fault diagnosis datasets.

The DX Competition has a history of successful editions [22, 21, 40, 47, 13], including synthetic track based on faults injected into ISCAS85 circuits, industrial tracks ADAPT and ADAPT-Lite, based on the Electrical Power System (EPS) testbed, software track, and thermal fluid track, which presented problems in a building's heating, ventilation, and air conditioning (HVAC) domain. DXC competitions gave rise to or helped evaluate numerous diagnostic algorithms, including FACT [43], HyDE [29], LYDIA [14], ProADAPT [26] RODON [24], and Wizards of Oz [16].

The range of problems covered in fault diagnosis benchmarks is extensive (from software and digital circuits to continuous processes from various domains), but it is still far from exhaustive. The benchmarks vary in complexity and the task (diagnosis, prognosis, planning). Due to data scarcity, primarily covering data with faults, many of the benchmarks rely on simulated data. There is a lack of benchmarks offering data and a structured process description.

3 SLIDe

SLIDe (Steam Line Intrusion Detection Benchmark) benchmark is devoted to the analysis of diagnostic algorithms for the detection and isolation of process faults and the detection of cyberattacks on a simulated fragment of the steam line of a fluidized bed boiler including

[8] https://www.nasa.gov/intelligent-systems-division/discovery-and-systems-health/pcoe/pcoe-data-set-repository/
[9] https://tulaut.github.io/

the third and fourth stage of superheaters. It includes challenging scenarios that exhibit sensor, actuator, and technological component faults as well as cyberattacks. To reflect the industrial nature of the benchmark, we provide only a qualitative description of the process with a list of measurements and a few prepared datasets representing different operating conditions, but only for fault-free and attack-free states.

The 2-stages steam line superheaters simulator models the processes within the boiler of a power unit. In each of these sections, there is an attemperator, a superheater, and a cascade controller – the main controller controls the temperature of the steam after the superheater, while the auxiliary controller, which controls the injection water valve, controls the temperature after the cooler. The schematic diagram of the process is shown in Figure 1.

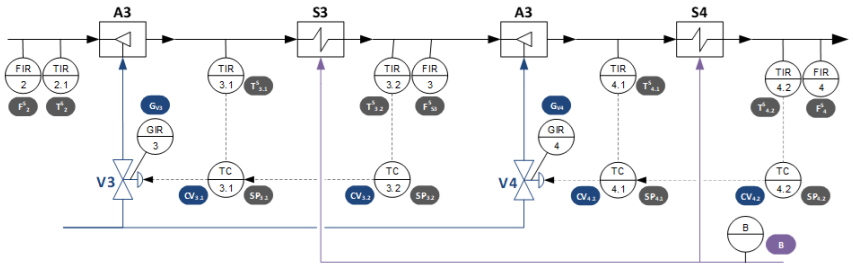
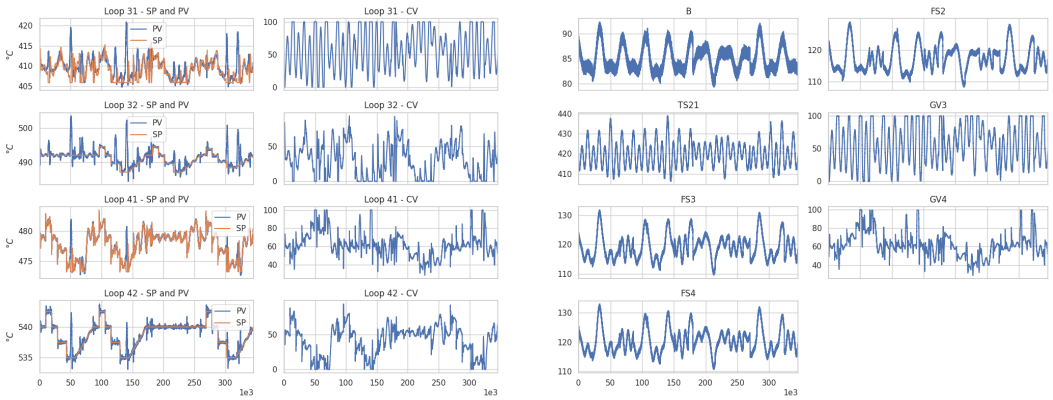


Figure 1 Process block diagram.

The benchmark simulator is implemented in Matlab. Control and measured variables are shown in Figure 2. B denotes the fuel inflow to the boiler, F steam flows, T temperatures, G positions of the injection valves, SP set points, and CV control signals. Figure 2 shows traces of control loops and process variables.



(a) Control loops.

(b) Process variables.

Figure 2 Control loops and process variables.

3.1 Process faults

The benchmark includes 16 process and sensor faults. The symbolic locations where process faults can be introduced are shown in Figure 3.

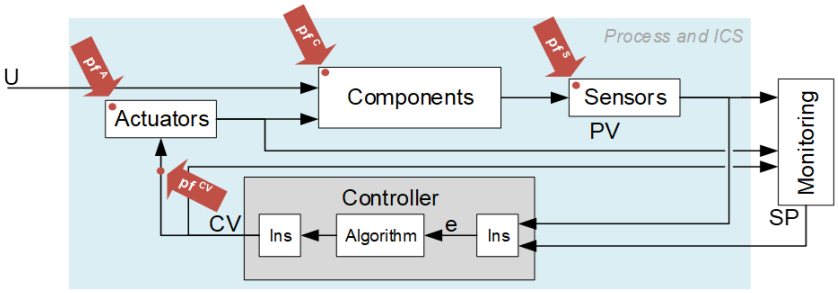


Figure 3 Symbolic designation of types and places of introduction of process faults.

Process faults are divided into the following types according to the entry points:

- pf^S: Incorrect operation of the measurement signal path.
- pf^A: Faulty operation of the actuator.
- pf^{CV}: Control signal path malfunction.
- pf^C: Technological component fault.

3.2 Cyber-attacks

Each cyber attack is carried out according to a designed scenario – a specific method of attack. Such a scenario consists of elementary impacts on individual system elements and signals in communication channels called cyber faults. The symbolic locations for introducing cyber faults in the simulator are shown in Figure 4.

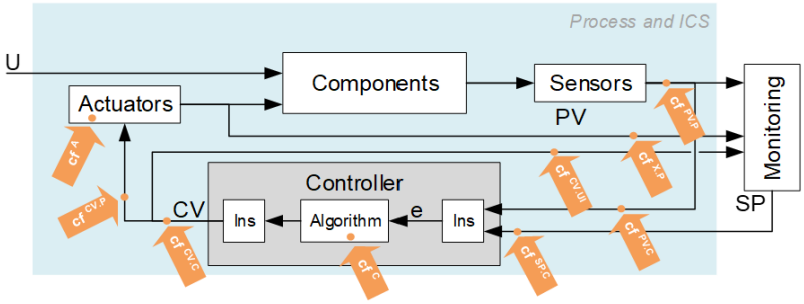


Figure 4 Symbolic designation of types and places of introduction of cyber faults.

We consider the following types of cyberattacks:

- cf^C – attack on the controller (change of operating mode, change of parameters),
- cf^SP – modification of set-points,
- cf^{CV} – modification of control variables,
- cf^{PV} – modification of controlled variables,
- cf^A – attack on the actuator (blockage, modification of operation, changes of operating parameters).

Cyber faults should be isolated to the specific control loop, *i.e.* the competitor's task is to detect cyber faults and say which control loop is affected. It is possible for more than one control loop to be affected by the same cyber-attack scenario. It is not necessary to isolate the cyber fault to the specific component.

3.3 Additional resources

Training datasets are available at the competition website[10]. Training and evaluation data from the previous version of the benchmark are available [48]. Exemplary algorithms for fault and cyber-attack detection and isolation can be found in [49, 53, 52].

4 LUMEN

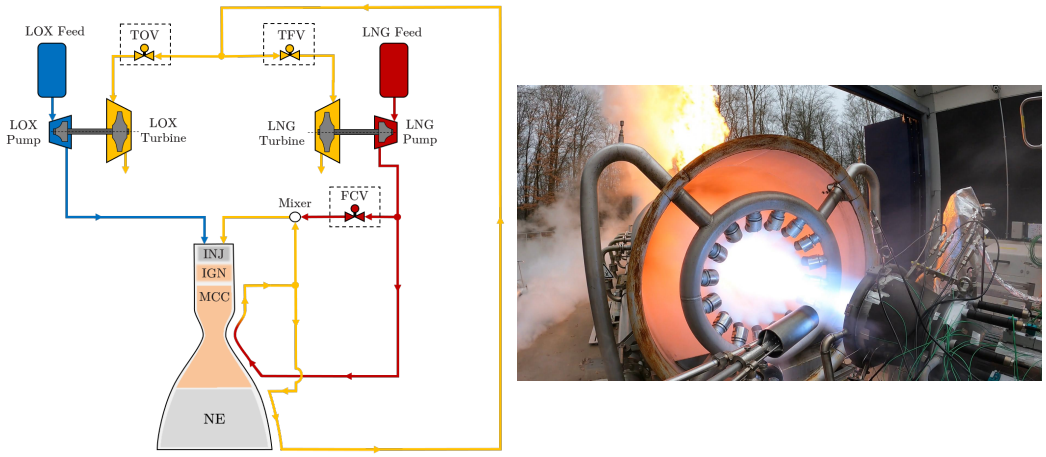
Early launch vehicles such as the American *Saturn 5* or the European rocket family *Ariane*, were expendable. Diagnostics therefore focused on pre-flight tests and post-test evaluations while no sophisticated system was used on-board. However, as the space industry shifts towards reusability and cost reduction, on-board diagnostic systems for health monitoring are essential for next-generation rocket engines. The *RS-25*, Space Shuttle's Main Engine (SSME), was the first reusable liquid rocket engine (LRE). For on-board diagnostics, dynamic limit-checks (redlines) for critical engine parameters, *e.g.* rotational speed of the turbopumps or combustion chamber pressure, were performed [6]. Those redlines were defined based on engineering judgement and experience. Although this simple method works well for most component faults, there are still severe problems: Sensor faults can cause unnecessary engine shutdowns and component faults can remain undetected. During the operation of rocket engines, sensors, such as pressure transducers and thermocouples, are subjected to high thermal and mechanical stresses, which make them susceptible to failure. This is underlined by the history of Space Shuttle ground test aborts, launch delays and the launch abort of flight STS-51-F caused by faulty sensors [4]. In addition, undetected component faults can result in catastrophic events [54]. For a more sophisticated diagnosis of the engine's health, vibration data was used on-board of the SSME to detect faults in the turbopumps which are the most common source of failure in rocket engines [6]. Newer reusable launchers such as SpaceX's *Falcon 9* also use sophisticated systems for detecting off-nominal conditions and initiating autonomous safe shutdowns [46]. The effectiveness of their health monitoring system was demonstrated on various flights, *e.g.*, during flight 84 where one of the nine engines on the first stage of the rocket was shutdown due to an anomaly and the mission was still successful. Following SpaceX's lead, Europe and other nations are actively researching and developing reusable rockets. The European rocket engine *Prometheus*, for example, is being optimised for reusability and cost reduction. The diagnosis of the the engine's health plays a vital role in achieving this goal [44]. The development and testing of these diagnostic systems involves component-level testing and ground tests, with the ultimate goal of ensuring their suitability in-flight.

LUMEN (**L**iquid **U**pper stage de**M**onstrator **EN**gine) is a modular pump-fed liquid oxygen (LOX) and liquid methane (LNG) rocket engine with 25 kN thrust, developed by the Institute of Space Propulsion of the German Aerospace Center (DLR). The operational envelope of LUMEN covers combustion chamber pressures from 40 bar to 80 bar and mixture ratios between 3.0 to 3.8. DLR has successfully completed two hot-fire test campaigns of LUMEN, demonstrating key capabilities such as stable combustion over a wide throttling range of 38 bar to 78 bar [10]. With this achievement LUMEN is now fully operational and available as a testbed for the development of intelligent control and diagnosis systems for health monitoring.

[10] `https://conf.researchr.org/home/dx-2025`

4.1 System description

This benchmark is proposed as a challenging problem for fault diagnosis of safety-critical technical systems with focus on the LUMEN engine. LUMEN, as shown schematically in Figure 5, is operated in an expander-bleed cycle. Both propellants are pressurized by separate turbopump units. While LOX is injected directly into the combustion chamber (MCC), LNG is first used for the regenerative cooling of the combustion chamber in a counterflow arrangement. The heated coolant flow is partially remixed with LNG to actively control the fuel injection temperature. The remaining cooling mass flow is further heated within the nozzle extension (NEM) and is then used to drive LOX and LNG turbines. Afterwards, the turbine exhaust is vented without being combusted. The generated thrust and therefore the operating point of LUMEN is defined by the combustion chamber pressure, the mixture ratio and the cooling channel mass flow. LUMEN's operating point is set by the position of the control valves TFV, TOV and FCV in open-loop. A more detailed description of LUMEN can be found in [9, 23, 55].



(a) Flow scheme of LUMEN.

(b) Thrust chamber of LUMEN during a hot-fire test.

Figure 5 The LUMEN benchmark.

4.2 Challenges

The development of fault diagnosis systems for LUMEN is complicated for various reasons, *e.g.* a limited amount of experimental data, measurement inaccuracies, nonlinear dynamics, the wide throttling range and strong coupling of all components. The placement of sensors is also constrained by extreme temperatures, high pressures and vibrations within the engine, which can damage instrumentation and lead to unreliable measurements. In addition, experimental data for fault scenarios can not be intentionally collected due to the inherent risk of a catastrophic failure. As a result of the extreme operating conditions close to the physical limits, the engine may also fail for unpredictable reasons. A number of challenges in developing a diagnosis system are defined by these difficulties.

The required robustness to unknown faults due to the lack of data for failure scenarios is one of the main challenges. The diagnosis system should detect any deviation from nominal operation as fast as possible but also reason based on the symptoms whether a detected fault can be isolated to known fault scenarios or is unknown. Another challenge poses the

scarcity of experimental data. Accurate reduced-order simulation models offer the possibility of generating representative data in controlled environments for both nominal operation and fault scenarios. However, resulting from unavoidable modeling errors in reduced-order models, the simulator is only an approximation of the real system. Closing this simulation-reality gap is not trivial and needs to be addressed in the development of diagnosis systems for rocket engines.

4.3 Provided resources

For developing the diagnosis system, a transient simulation model of LUMEN is provided. The simulator accurately reproduces experimental results in both transient and steady-state conditions with errors below 10 %. The simulator is based on differential-algebraic equations (DAE) and built with *EcosimPro*, the state-of-the-art modeling tool for space applications, and the European Space Propulsion System Simulation (ESPSS) library. The behavior of each component is defined by a set of geometrical and physical parameters such as the length, diameter and wall roughness of a pipe which cannot be changed in the simulator. For performing transient simulations, a Python interface is provided which can be used to adjust the position of the control valves and therefore change the operating point of LUMEN at each time step. The output of the simulator consists of noisy measurements at positions in which sensors are commonly placed within a flight-like rocket engine. In total, the output of the simulator consists of eight pressure measurements, *e.g.* the combustion chamber pressure, seven temperature measurements, *e.g.* the turbine inlet temperature, the rotational speed of the two turbopumps, the command and position signal of each valve as well as five mass flows at different positions, *e.g.* the injection fuel mass flow. As flowmeters are not used on board of rocket engines, the provided mass flows are calculated based on other measurements. The simulator can be used to generate both nominal trajectories and trajectories in which a fault is introduced at an user-defined time. In total, 15 sensor faults, three actuator faults and three components faults can be simulated.

To replicate the challenge of the adaption to the real system in this benchmark, we introduce another simulation model that is not provided to the participants. The modified simulator has a slightly different set of physical parameters and is a representation of the real system in this benchmark. This real system simulator is used for evaluating the performance of the diagnosis systems. To mimic the scarcity of available experimental data, a limited set of nominal trajectories which is generated with the real system simulator is also provided. In addition to the known fault scenarios, we use the real system simulator to generate trajectories for fault scenarios which are unknown to the participants *a priori*. If the symptoms of these faults differ from known faults, the diagnosis system should classify this fault as unknown. An overview of the provided resources and the evaluation process is given in Figure 6. The evaluation metrics are described in Section 7. For evaluating the diagnosis system solution, a Docker container with evaluation code is provided as described in Section 6.

4.4 Fault scenarios

Component, sensor, and actuator faults can be introduced in the simulation model at each time step. Sensor faults can be injected into all measurements by multiplying the measured variable by a fault factor $f \in [0.8, 1.2]$. As the simulation is performed open loop, sensor faults affect the measured signal and downstream calculations of the mass flows. Component and actuator faults, on the other hand, influence the operation of the entire engine as a result of strong coupling. The actuator fault is modeled as a stuck valve that does not change position according to the command signal. This fault can be introduced in TFV, TOV and FCV. In addition, three component faults with different magnitudes can be simulated:

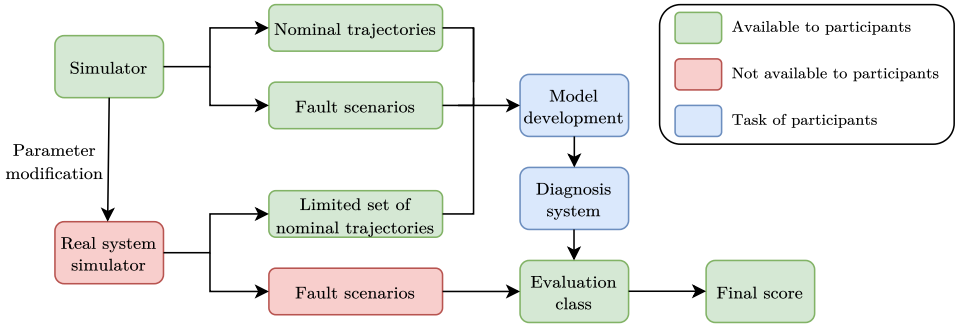


Figure 6 Overview of the evaluation process for the LUMEN benchmark.

- **Blockage of the turbine inlet nozzle**: This fault models a geometrical change in the flow area of the inlet turbine nozzles that can result from stuck particles.
- **Leakage**: This fault simulates a leakage mass flow downstream of the pump. It is modeled with an additional valve that can be partially opened, and thus introduce a leakage mass flow.
- **Increased pressure drop**: This fault simulates an additional pressure drop within the feedlines of the system. It is modeled with an additional valve that can be partially closed to increase the pressure drop.

To illustrate the described effects of each type of fault, Figure 7 shows the normalized sensor signals for the valve command for FCV, TFV and TOV, the fuel injection mass flow, the combustion chamber pressure and the rotational speed of the fuel turbopump (FTP).

5 LiU-ICE

The first version of the Linköping University Internal Combustion Engine (LiU-ICE) industrial benchmark was initially presented in [17]. The benchmark is proposed as a challenging industrial-relevant case study to support fault diagnosis research. Engine fault diagnosis is a nontrivial task that is complicated by nonlinear dynamic behavior, with slow and fast dynamics, a wide operating range, and both stationary and transient operation.

Developing a diagnosis system is complicated by system model inaccuracies, measurement uncertainties, and limited training data from relevant fault scenarios. The objective of the competition is to address these challenges by designing a diagnosis system for the air path of an internal combustion engine.

5.1 System description

The benchmark consists of operational data collected from an internal combustion engine test bench, see Figure 8a and a mathematical model, where the model parameters are unknown. Figure 8b shows a schematic of the modeled part of the system, which is the air path through the engine. The available sensor signals are as follows:

- y_{pic} – Intercooler pressure
- y_{Tic} – Intercooler temperature
- y_{pim} – Intake manifold pressure
- y_{waf} – Mass flow through the air filter
- y_{xpos} – Throttle actuator position

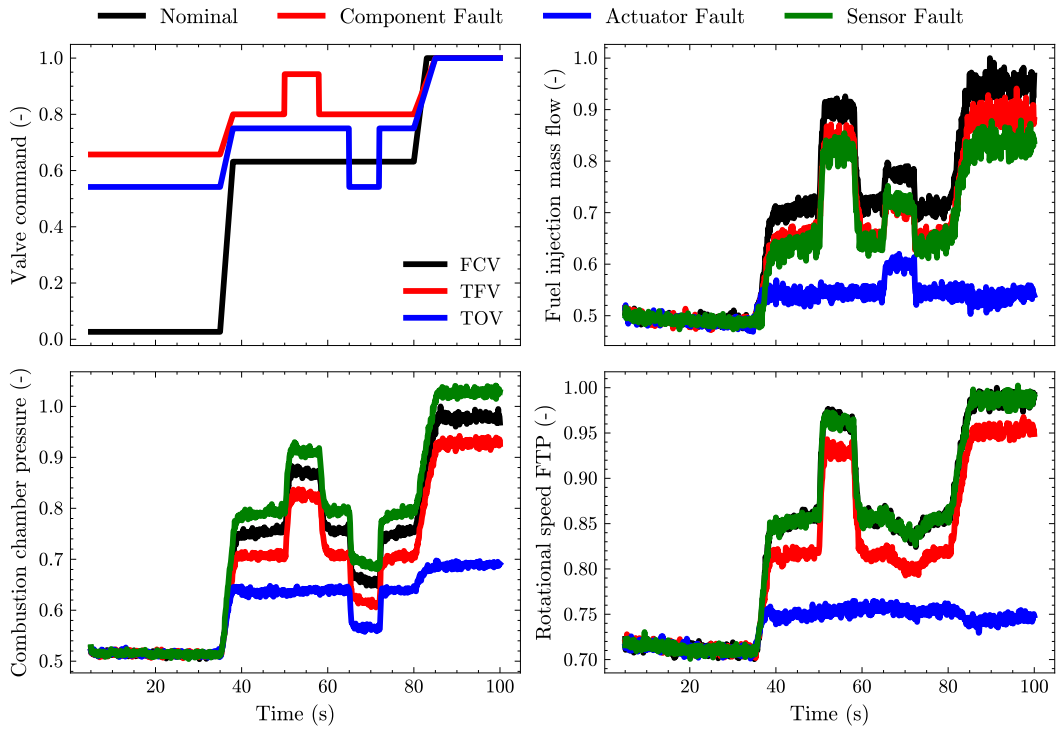


Figure 7 Examples of each fault type on normalized sensor signals. The fault is injected at different $t = 36$ s. The valve commands are identical for each fault.

- y_ω – Engine speed
- y_{pamb} – Ambient pressure
- y_{Tamb} – Ambient temperature

The known actuator signals are as follows:

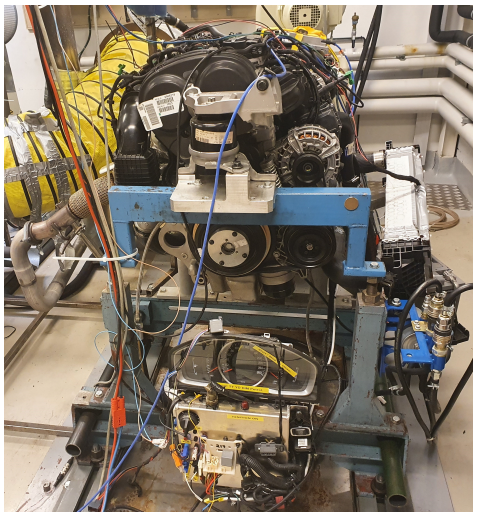
- u_{mf} – Requested injected fuel mass
- u_{wg} – Requested wastegate actuator position

The available signals represent a set of standard signals that are available in a production engine. Note that most signals are available in the air intake of the engine, see Figure 8b.

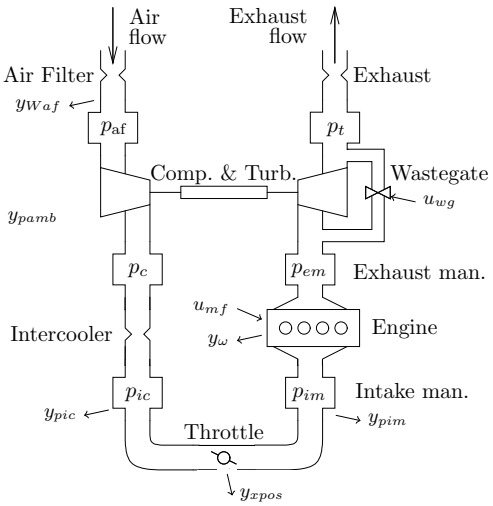
The airflow passes through an air filter before the compressor and the intercooler. A throttle is used to control the pressure in the intake manifold where air enters the cylinders, where it is mixed with fuel and ignited to generate torque. The exhaust gases pass through the exhaust manifold and the turbo that drives the compressor before leaving the exhaust. The wastegate is used to control how much of the exhaust gases pass through the turbo. The engine control unit makes sure that the engine provides the desired torque while controlling the stoichiometry of the air and fuel in the cylinder to optimize combustion and reduce emissions.

5.2 Fault scenarios

Faults are introduced during operation, either by opening a valve that represents a leak or by modifying a measurement signal in the engine control unit, representing a sensor fault. The faults considered are the following:



(a) The LiU-ICE test bench.



(b) A schematic of the air path of an IC engine.

Figure 8 The LiU-ICE benchmark.

- f_{ypic} – A fault in the inter-cooler pressure sensor y_{pic}
- f_{ypim} – A fault in the intake manifold pressure sensor y_{pim}
- f_{ywaf} – A fault in the air mass flow sensor y_{waf}
- f_{iml} – A leakage in the intake manifold

Sensor faults are injected as multiplicative as $y = (1 + f)x$ where y is the measurement signal, x is the measured variable, and $f \neq 0$ represents a fault that scales the measured signal. Since a sensor fault is introduced during operation, it can affect the operating conditions of the system through feedback loops. The magnitude of the leakage fault f_{iml} is defined by the diameter of the valve orifice.

5.3 Provided resources

In the benchmark, a mathematical model of the system and training data from various fault scenarios are provided. The mathematical model is in the form of semi-explicit differential algebraic equations (DAE) of index 1. The component models are similar to what is described in [12]. The provided model is implemented in the Fault Diagnosis Toolbox [15]. A structural representation of the provided model is shown in Figure 9 where the blue dots represent unknown variables, the red dots are fault signals, and the black dots are known signals.

5.3.1 Training data

The training data for this version of the LiU-ICE benchmark consist of 26 datasets and include different magnitudes of each fault. Each data set is sampled at 20 Hz. All training data sets in the benchmark have been collected using the Worldwide Harmonized Light Vehicle Test Procedure (WLTP). The WLTP cycle is approximately 30 minutes long and covers varying operating conditions and transient behavior that represent both urban and highway driving. Each fault is introduced after approximately two minutes into each corresponding dataset and is present for the rest of the cycle. A summary of the fault realizations is shown in Table 2. For sensor faults, the fault signal f can be both positive and negative, that is, the faulty signal is scaled up or down with respect to the true signal. Each data set starts with nominal operation, and the fault is injected after approximately two minutes.

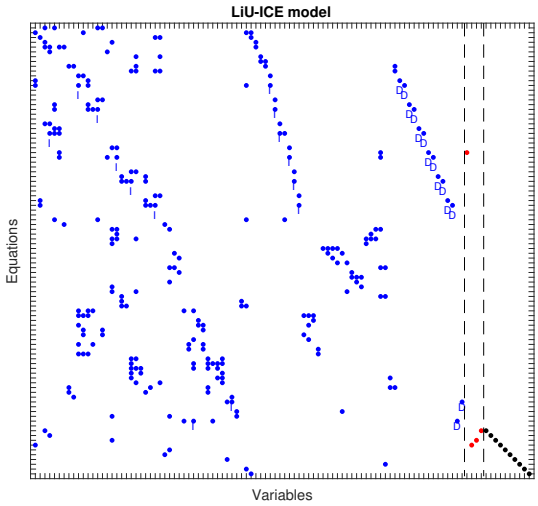


Figure 9 A structural representation of the engine model.

To illustrate the effects of each fault, Figure 10 shows the sensor signals y_{pic}, y_{pim}, and y_{waf} for one realization of each fault. The sensor faults in the plots are +10% and the leakage diameter is 6 mm. The injection of each fault is marked in the corresponding subplot where the fault is most visible. The sensor faults are marked in the corresponding signal and the leakage is highlighted in signal y_{pim} which measures close to the location of the leakage. In the figure, the signals have been translated in time, so they are synchronized in the drive cycles. Note that since the sensor faults are multiplicative, the same fault size for the different sensor faults results in different excitation in the signals. Since the fault f_{ywaf} is not visible in the plot, a zoomed in figure is shown in Figure 11.

For the final evaluation, a set of secret test data will be used to evaluate all participating solutions. Note that the fault scenarios in the test data can be other driving cycles than the WLTP cycle.

Training data sets are available on the competition website. The previous version of the benchmark is described in [17].

6 Benchmark implementation environment

To simplify the implementation of diagnostic system solutions to the different benchmarks, a standardized data format and code structure in Python are used. Each benchmark provides a Docker container with a similar evaluation code and a template for the implemented diagnosis system.

Table 2 Summary of training datasets with fault scenarios.

Fault	Magnitudes
f_{ypic}	-15%, -10%, -5%, 5%, 10%, 15%
f_{ypim}	-15%,-10%, -5%, 5%, 10%, 15%
f_{yWaf}	-15%,-10%, -5%, 5%, 10%, 15%
f_{iml}	4 mm, 6 mm

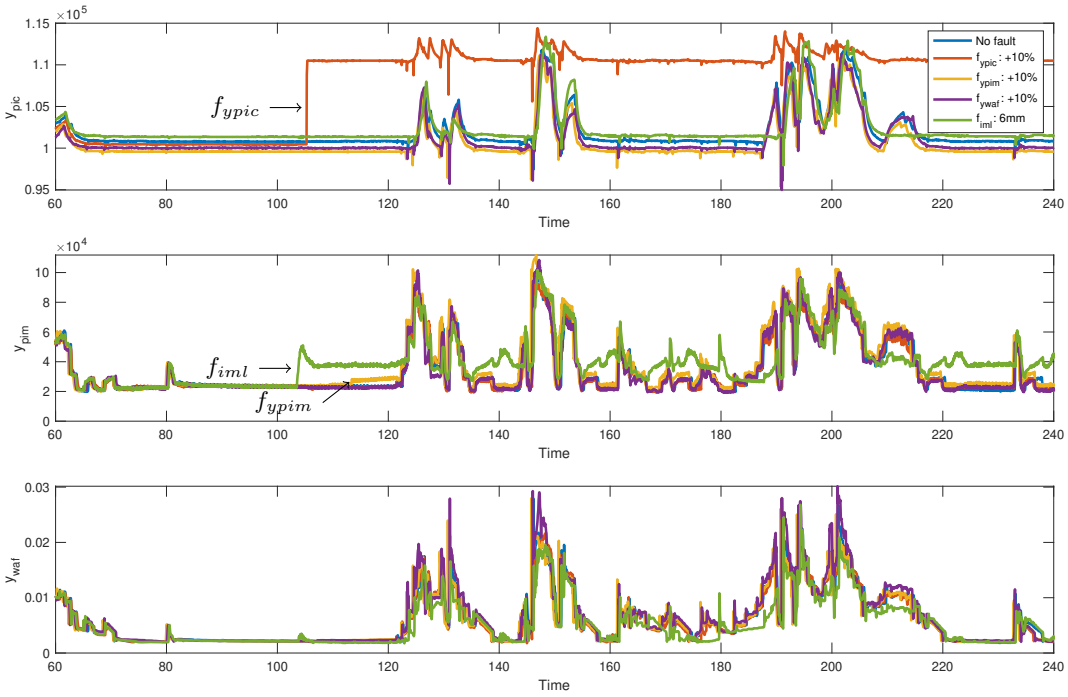


Figure 10 Examples of signals from different fault scenarios.

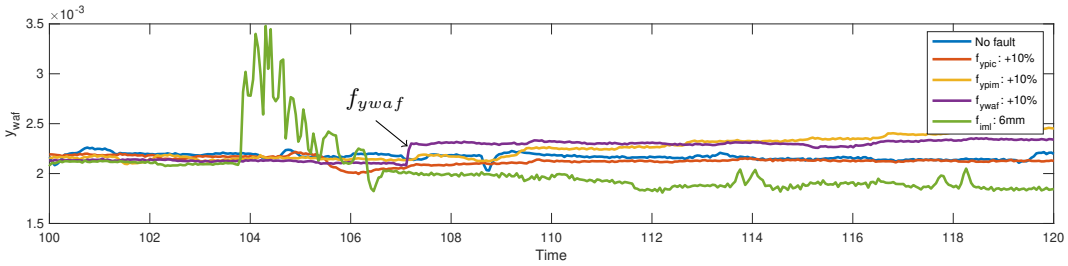


Figure 11 A zoom in of the y_{waf} signal from Figure 10 to show the sensor fault f_{ywaf}.

The diagnosis system solution should be implemented in Python in a class using the following template:

```
class DiagnosisSystemClass:
   def __init__(self):
       pass
    def Initialize(self):
      #initialize diagnosis system here
       pass
    def Input(self,sample):
      #Update diagnosis using new sample
       detection = [] # Set flag if fault is detected
       isolation = [] # Ranking of diagnoses
       return(detection,isolation)
```

found in `DiagnosisSystemClass.py`. Note that it is possible to update the class with functionality needed for the solution. It is important that the inputs and outputs to the above functions are not changed.

The evaluation code provided is found in the file `evaluate_diagnosis_system.py` as part of the benchmarks and calls the diagnosis system every time a new sample of data is available. The diagnosis system requires to return if a fault is detected and a ranking of the diagnoses is obtained. The evaluation of each fault scenario is stored in a csv file in the results folder in the container.

7 Evaluation metrics

There are various performance metrics that can be used for evaluation of diagnosis systems, where we outline in the following those chosen for DXC'25.

7.1 Diagnosis of faults

For each new sample, the diagnosis system should return a fault detection flag. When a fault is detected, the system provides a ranked list of diagnoses with decreasing posterior probability. The diagnosis system should have high fault detection accuracy and a low false alarm rate. At the same time, it is important to isolate the true fault to select a suitable countermeasure. The diagnosis system solutions are evaluated based on the following performance metrics:

- False alarm rate (FAR) – the percentage of samples in which the diagnosis system states that a fault is detected when there is no fault in the system.
- True detection rate (TDR) – the percentage of samples in which the diagnosis system states that a fault is detected when there is a fault in the system.
- Fault isolation accuracy (FIA) – the average probability given to the true diagnosis for all samples when a fault is correctly detected.

All performance metrics are between 0 and 1. The metrics are kept simple to simplify the comparison of different fault diagnosis solutions. The total score is calculated as a weighted sum of these performance metrics as follows:

$$\text{total score} = (\ (1\text{-FAR}) + \text{TDR} + \text{FIA}\ )/3.$$

where a higher value represents better performance. Note that a naive solution can achieve at least 0.3, *e.g.*, by not triggering any alarm.)

7.2 Diagnosis of cyberattacks

In the SLIDe benchmark, diagnosis of cybernetic faults is also evaluated based on the following performance metrics:

- False alarm rate (FAR) – the percentage of samples in which the diagnosis system states that a cybernetic attack is detected when there is no cyber attack in the system. We use 1 - FAR as a metric.
- True detection rate (TDR) – the percentage of samples in which the diagnosis system states that a cyber attack is detected when there is a cyber attack in the system.
- Cyber attacks isolation accuracy (CIA)

The isolation accuracy of cybernetic faults (CIA) is divided into two parts:

- True isolation rate (TIR) – the average probability assigned to the simulated attack vector,
- False isolation rate (FIR) – the average probability assigned to the loops that are not attacked. We use $1 - FIR$ as the isolation accuracy score.

The isolation accuracy score is computed as the harmonic mean of TIR and $1 - FIR$:

$$CIA = \frac{2 \cdot TIR \cdot (1 - FIR)}{TIR + (1 - FIR)} \tag{1}$$

The metrics used for cyber attacks differ in their approach to isolation accuracy because we also consider scenarios when multiple loops are attacked. The proposed metric better evaluates the cases where only some of the attacked loops are isolated correctly in contrast to only considering the probability of a correct diagnosis.

8 Summary

The three benchmarks described in this paper (and which are available from the DXC'25 benchmark repository[11]) served as a starting point for reviving the DX competition after a long hiatus. As we can easily deduce from the characteristics listed in Table 1, our aim was to provide an initial set of challenges that is diverse but also close enough to foster the testing of an approach for all benchmarks.

The benchmarks are continuously updated. Thus, we encourage prospective participants and interested readers to reach out to us for the latest versions. We also plan to complement the current benchmark set with additional ones that cover other types of system, such as discrete ones. Of particular interest will be extensions that cover additional diagnostic problems. This could include intermittent fault scenarios or the evaluation of a system's long-term performance (and degradation).

At the same time, we intend to expand the competition with challenges for approaches that integrate diagnosis with control, repair, and potentially prognosis. Evaluating these integrated approaches will, in particular, allow us to investigate the effectiveness of various diagnosis concepts regarding their integration into design approaches for intelligent systems.

Being able to analyze and, in turn, anticipate the exact needs for diagnostic support in the decision making of an intelligent autonomous system shall provide the community with the background to make educated design decisions towards enabling resilience in a system. That is, the ability to reasoning about problems and their mitigation at run-time enables resilient systems to maintain their functionality not only for anticipated fault scenarios, but also in situations and circumstances that could not be anticipated at design time.

References

1 J. L. Augustin and O. Niggemann. Graph Structural Residuals: A Learning Approach to Diagnosis, 2023. doi:10.48550/arXiv.2308.06961.

2 K. Balzereit, A. Diedrich, J. Ginster, S. Windmann, and O. Niggemann. An ensemble of benchmarks for the evaluation of AI methods for fault handling in CPPS. In *2021 IEEE 19th Int. Conf. on Industrial Informatics (INDIN)*, pages 1–6. IEEE, 2021.

3 M. Bartyś, R. Patton, M. Syfert, S. de las Heras, and J. Quevedo. Introduction to the DAMADICS actuator FDI benchmark study. *Control Engineering Practice*, 14(6):577–596, 2006.

4 T. W. Bickmore. Real-Time Sensor Data Validation. *Contractor Report, NASA-CR-195295*, 1994.

5 P. Chatterjee, J. Campos, R. Abreu, and S. Roy. Augmenting Automated Spectrum Based Fault Localization for Multiple Faults. In *32nd Int. Joint Conf. on Artificial Intelligence (IJCAI-23)*, pages 3140–3148, August 2023.

[11] https://vehsys.gitlab-pages.liu.se/dx25benchmarks/

6 M. Davidson and J. Stephens. Advanced health management system for the space shuttle main engine. In *40th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 2004.

7 J. de Kleer and B. C. Williams. Diagnosis with Behavioral Modes. In *11th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1324–1330, 1989.

8 J. Downs and E. Vogel. A plant-wide industrial process control problem. *Computers & chemical engineering*, 17(3):245–255, 1993.

9 K. Dresia, M. Börner, W. Armbruster, S. Klein, T. Traudt, D. Suslov, J. Hardi, G. Waxenegger-Wilfing, and J. C. Deeken. Design and control challenges for the LUMEN LOX/LNG expander-bleed rocket engine. In *34th Int. Symposium on Space Technology and Science (ISTS)*, 2023.

10 K. Dresia, T. Traudt, M. Börner, D. Suslov, W. Armbruster, R. dos Santos Hahn, E. Kurudzija, C. Groll, M. A. Müller, S. Klein, J. Hämisch, J. Deeken, J. Hardi, and S. Schlechtriem. Hot-fire testing and system analysis of the LUMEN liquid upper stage demonstrator engine. In *3rd Int. Conf. on Flight Vehicles, Aerothermodynamics and Re-entry (FAR)*, 2025.

11 J. Ehrhardt, M. Ramonat, R. Heesch, K. Balzereit, A. Diedrich, and O. Niggemann. An AI benchmark for diagnosis, reconfiguration & planning. In *2022 IEEE 27th Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2022.

12 L. Eriksson. Modeling and control of turbocharged SI and DI engines. *Oil & Gas Science and Technology-Revue de l'IFP*, 62(4):523–538, 2007.

13 A. Feldman, J. de Kleer, T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, and A. van Gemund. The diagnostic competitions. *AI Magazine*, 35(2):49–54, 2014. `doi:10.1609/AIMAG.V35I2.2532`.

14 A. Feldman, G. Provan, and A. van Gemund. The Lydia approach to combinational model-based diagnosis. *Proc. Int. Workshop on Principles of Diagnosis*, 9:403–408, 2009.

15 E. Frisk, M. Krysander, and D. Jung. A toolbox for analysis and design of model based diagnosis systems for large scale models. *IFAC-PapersOnLine*, 50(1):3287–3293, 2017.

16 A. Grastien and P. Kan-John. Wizards of Oz description of the 2009 DXC entry. *Proc. Int. Workshop on Principles of Diagnosis*, 9:409–413, 2009.

17 D. Jung, E. Frisk, and M. Krysander. The LiU-ICE benchmark–an industrial fault diagnosis case study. *arXiv preprint*, 2024. `arXiv:2408.13269`.

18 D. Jung, E. Frisk, M.s Krysander, A. Sztyber-Betley, F. Corrini, A. Arici, N. Anselmi, M. Mazzoleni, J. Xu, S. Mo, Z. Xu, C. Yang, Z. Du, H. Safaeipour, M. Forouzanfar, V. Mirahi, A. Pinnarelli, V. Puig, Q. Deng, Y. Liu, J. Liu, H. Ke, W. Zhu, S. Merkelbach, M. Ahang, and H. Najjaran. A fault diagnosis benchmark of technical systems with incomplete data – six solutions. *Control Engineering Practice*, 2025. to appear.

19 D. Jung, H. Khorasgani, E. Frisk, M. Krysander, and G. Biswas. Analysis of fault isolation assumptions when comparing model-based design approaches of diagnosis systems. *IFAC-PapersOnLine*, 48(21):1289–1296, 2015.

20 D. Jung and M. Krysander. Assumption-based Design of Hybrid Diagnosis Systems: Analyzing Model-based and Data-driven Principles. In *Annual Conf. of the PHM Society*, 2024.

21 T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, and A. Feldman. Second international diagnostic competition (dxc'10), 2010.

22 T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, and A. Feldman. First international diagnosis competition-DXC'09. *Proc. Int. Workshop on Principles of Diagnosis DX*, 9:383–396, 2009.

23 E. Kurudzija, K. Dresia, J. Martin, T. Traudt, J. C. Deeken, and G. Waxenegger-Wilfing. Virtual sensing for fault detection within the LUMEN fuel turbopump test campaign. In *9th Edition of the Space Propulsion Conference, Glasgow, Scotland.*, May 2024.

24 K. Lunde, R. Lunde, and B. Münker. Model-based failure analysis with rodon. In *ECAI 2006*, pages 647–651. IOS Press, 2006.

25 I. Matei, M. Zhenirovskyy, J. de Kleer, and A. Feldman. Classification-based Diagnosis Using Synthetic Data from Uncertain Models. *Annual Conference of the PHM Society*, 10(1), 2018.

26 O. J. Mengshoel. Designing resource-bounded reasoners using bayesian networks: System health monitoring and diagnosis. In *Proc. of the 18th Int. Workshop on Principles of Diagnosis (dx-07)*, pages 330–337, 2007.

27 A. Metodi, R. Stern, M. Kalech, and M. Codish. Compiling Model-Based Diagnosis to Boolean Satisfaction. In *26th AAAI Conf. on Artificial Intelligence*, pages 793–799, 2012.

28 L. Moddemann, H. Steude, A. Diedrich, I. Pill, and O. Niggemann. Extracting Knowledge using Machine Learning for Anomaly Detection and Root-Cause Diagnosis. In *29th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2024. to appear.

29 S. Narasimhan and L. Brownston. HyDE – A general framework for stochastic and hybrid modelbased diagnosis. *Proc. Int. Workshop on Principles of Diagnosis*, 7:162–169, 2007.

30 I. Nica, I. Pill, T. Quaritsch, and F. Wotawa. The Route to Success – A Performance Comparison of Diagnosis Algorithms. In *23rd International Joint Conference on Artificial Intelligence*, pages 1039–1045, 2013.

31 P. Odgaard and J. Stoustrup. Results of a wind turbine FDI competition. *IFAC Proceedings Volumes*, 45(20):102–107, 2012.

32 P. Odgaard, J. Stoustrup, and M. Kinnaert. Fault-tolerant control of wind turbines: A benchmark model. *IEEE Transactions on control systems Technology*, 21(4):1168–1182, 2013. doi:10.1109/TCST.2013.2259235.

33 I. Pill and T. Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *23rd Int. Joint Conf. on Artificial Intelligence*, pages 1053–1059, 2013.

34 I. Pill and T. Quaritsch. RC-Tree: A variant avoiding all the redundancy in Reiter's minimal hitting set algorithm. In *IEEE Int. Symp. on Software Reliability Engineering Workshops (ISSREW)*, pages 78–84, 2015.

35 I. Pill, T. Quaritsch, and F. Wotawa. Parse tree structure in LTL requirements diagnosis. In *2015 IEEE Int. Symp. on Software Reliability Engineering Workshops*, pages 100–107, 2015.

36 I. Pill and F. Wotawa. Spectrum-Based Fault Localization for Logic-Based Reasoning. In *2018 IEEE Int. Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 192–199, 2018.

37 I. Pill and F. Wotawa. Exploiting observations from combinatorial testing for diagnostic reasoning. In *30th Int. Workshop on Principles of Diagnosis*, 2019.

38 I. Pill and F. Wotawa. Extending Automated FLTL Test Oracles with Diagnostic Support. In *IEEE Int. Symp.on Software Reliability Engineering Workshops*, pages 354–361, 2019.

39 I. Pill and F. Wotawa. Computing Multi-Scenario Diagnoses. In *31st Int. Workshop on Principles of Diagnosis*, 2020.

40 S. Poll, J. de Kleer, R. Abreau, M. Daigle, A. Feldman, D. Garcia, and A Sweet. Third international diagnostics competition–DXC'11. In *Proc. of the 22nd Int. Workshop on Principles of Diagnosis*, pages 267–278, 2011.

41 S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, et al. Advanced diagnostics and prognostics testbed. In *DX Int. Workshop on Principles of Diagnosis*, pages 178–185, 2007.

42 R. Reiter. A Theory of Diagnosis from First Principles. *Art. Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

43 I. Roychoudhury, G. Biswas, and X. Koutsoukos. Designing distributed diagnosers for complex continuous systems. *IEEE Trans. on Automation Science and Engineering*, 6(2):277–290, 2009. doi:10.1109/TASE.2008.2009094.

44 P. Simontacchia, R. Blasi, Edeline E., S. Sagnier, , A. Espinosa-Ramos, J. Breteau, and P. Altenhöfer. PROMETHEUS: Precursor of new low-cost rocket engine family. In *Proc. of the 8th European Conf. for Aeronatuics and Space Sciences (EUCASS)*, 2019.

45 W. A. Smith and R. B. Randall. Rolling element bearing diagnostics using the case western reserve university data: A benchmark study. *Mechanical Systems and Signal Processing*, 64-65:100–131, 2015.

46 SpaceX. *Falcon User's Guide*. Space Exploration Technologies Corp., September 2021. (visited on 05/09/2025). URL: `https://www.spacex.com/media/falcon-users-guide-2021-09.pdf`.

47 A. Sweet, A. Feldman, S. Narasimhan, M. Daigle, and S. Poll. Fourth international diagnostic competition–DXC'13. In *Proc. of the 24th Int. Workshop on Principles of Diagnosis*, pages 224–229, 2013.

48 M. Syfert. Cyber-attack scenarios for super-heaters system, March 2023. `doi:10.5281/zenodo.7612269`.

49 M. Syfert, P. Wnuk, A. Sztyber-Betley, and M. Pobocha. The Model of Ongoing Diagnosis of Process Faults and Detection of Cybernetic Attacks for a Steam Line. *Acta Physica Polonica A*, 146(4):438, 2024.

50 A. Sztyber, E. Chanthery, and L. Travé-Massuyès. Benchmark for fault diagnosis of water distribution network. In *34rd Int. Workshop on Principle of Diagnosis – DX 2023*, pages 1–8, 2023.

51 A. Sztyber, E. Chanthery, L. Travé-Massuyès, and C. G. Pérez-Zuñiga. Water network benchmarks for structural analysis algorithms in fault diagnosis. In *33rd Int. Workshop on Principle of Diagnosis – DX 2022*, 2022.

52 A. Sztyber, Z. Górecka, J. M. Kościelny, and M. Syfert. Controller modelling as a tool for cyber-attacks detection. In Z. Kowalczuk, editor, *Intelligent and Safe Computer Systems in Control and Diagnostics*, pages 100–111, Cham, 2023. Springer International Publishing.

53 A. Sztyber-Betley, M. Syfert, J. M. Kościelny, and Z. Górecka. Controller Cyber-Attack Detection and Isolation. *Sensors*, 23(5), 2023. `doi:10.3390/S23052778`.

54 A. E. Tischer and R. C. Glover. Studies and Analyses of the Space Shuttle Main Engine. *Contractor Report, NASA-CR-183593*, 1987.

55 T. Traudt, W. Armbruster, C.r Groll, R. H. Dos Santos Hahn, K. Dresia, M. Börner, S. Klein, D. Suslov, E. Kurudzija, J. Haemisch, M. A. Müller, J. C. Deeken, J. Hardi, and S. Schlechtriem. LUMEN, the test bed for rocket engine components: Results of the acceptance tests and overview on the engine test preparation. In *9th Edition of the Space Propulsion Conference*, May 2024.

56 D. Vranješ, J. Ehrhardt, R. Heesch, L. Moddemann, H. S. Steude, and O. Niggemann. Design Principles for Falsifiable, Replicable and Reproducible Empirical Machine Learning Research. In I. Pill, A. Natan, and F. Wotawa, editors, *35th Int. Conf. on Principles of Diagnosis and Resilient Systems (DX 2024)*, volume 125 of *Open Access Series in Informatics (OASIcs)*, pages 7:1–7:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/OASICS.DX.2024.7`.

Data-Driven Fault Detection and Isolation Enhanced with System Structural Relationships

Austin Coursey
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Abel Diaz-Gonzalez
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Marcos Quinones-Grueiro
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Gautam Biswas
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Abstract

Fault detection and isolation are becoming increasingly important as modern systems become more complex. To encourage the development of new fault detection solutions that can operate with limited noisy data and an incomplete mathematical model, the DX 2025 LiU-ICE competition for diagnosis of the air path of an internal combustion engine was introduced. In this paper, we present our winning solution to this competition. Our fault detection architecture starts with a semi-supervised Transformer Autoencoder trained to reconstruct nominal data. Detected faults are then passed through a rule-based fault persistence filter that aims to suppress false positives. Once a fault is detected, we use four neural networks trained to estimate features determined from structural analysis of a partial system model. The residuals of these networks are fed to a supervised fault classification network that estimates the fault probabilities. With this architecture, we achieved an 87% detection rate with a 0% false alarm rate on the provided competition data. Additionally, our isolation architecture assigned the correct fault 73.8% probabilty on average. On unseen competition data from a new driving cycle, we achieved a 100% detection rate and assigned the correct fault 66.2% probability on average. On the other hand, the Transformer Autoencoder failed to transfer to the new driving conditions, causing many false alarms. We discuss ways future work can reduce this.

2012 ACM Subject Classification Computing methodologies → Anomaly detection; Applied computing → Engineering; Computing methodologies → Neural networks

Keywords and phrases fault detection, fault isolation, autoencoder

Digital Object Identifier 10.4230/OASIcs.DX.2025.15

Category DX Competition

Supplementary Material *Software*: `https://github.com/MACS-Research-Lab/vandy-dx2025`

Funding This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 2444112. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

Modern systems, such as those in transportation, manufacturing, and energy, are becoming increasingly complex and critical to daily life. As the complexity of these systems increases, the number of potential faults often increases. These faults can lead to performance degradation, safety risks, or even catastrophic failure [9]. Therefore, fault detection and isolation (FDI) methods are essential to ensure the reliability, efficiency [5], and safety [26] of complex systems.

Traditional FDI approaches are model-based, developing system-specific methods [1]. Model-based methods have historically relied on analyzing the patterns of the difference between observed sensor values and the output predicted by a mathematical model of the system, known as residuals [10]. This framework serves as the foundation for many modern model-based approaches, such as [25, 6]. Although model-based methods can be successful in detecting and isolating faults, they are limited in scenarios without an accurate and complete model of the system. In these scenarios, data-driven models can be applied if behavioral data has been collected for the system. Data-driven fault detection and diagnosis methods can be classified as supervised or unsupervised [1]. Supervised approaches leverage labeled training data to discriminate between faulty and nominal data or to classify the fault category, e.g., [24]. In many domains, obtaining labeled fault data is infeasible or prohibitively expensive. In these cases, unsupervised or semi-supervised data-driven approaches have been used requiring partially or no labeled data , e.g, [4]. While data-driven FDI approaches offer promising solutions when a complete system model is unavailable, they can be combined with model-based approaches when some knowledge of the system is available, such as in [22]. Despite the increasing number of FDI solutions, inaccuracies in the system model, limited training data, and measurement noise complicate the task, and existing solutions do not fully address these issues [14].

With these challenges in mind, the LiU-ICE benchmark was introduced as a DX 2025 competition [14]. This benchmark focuses on fault detection and isolation of the air path of an internal combustion engine. It consists of an analytical model with unknown parameters and data from nominal and four faulty scenarios with varying magnitudes. In this paper, we present a solution to the LiU-ICE DX 2025 competition. Our solution consists of two steps: fault detection and isolation. To detect faults, we use a semi-supervised Transformer Autoencoder that aims to reconstruct the current data sample given a window of recent samples. When the reconstruction error is higher than a threshold determined from nominal data, the sample is flagged as a potential fault. Potential faults are filtered by a rule-based persistence filter designed to suppress false alarms. After detection, we isolate faults. Using structural analysis, we obtain Minimally Structurally Over-determined (MSO) sets. For each MSO set, we train a feature estimation neural network that learns system relationships. The residuals of each feature estimation network are fed to a supervised fault classifier that estimates the probabilities of each fault.

The contributions of this paper are as follows.

- We present a data-driven fault detection and isolation architecture enhanced with system structural relationships that achieved first place in the LiU-ICE DX 2025 competition.
- We perform an in-depth analysis of the strengths and limitations of our approach on the provided competition data. On the evaluation data, we discuss the performance, including autoencoder transfer issues that cause a high false alarm rate and potential solutions.

The remainder of the paper is structured as follows. In Section 2, we give necessary background about the competition, dataset, and performance metrics. In Section 3, we describe our fault detection and isolation architecture. In Section 4.2, we describe the results on the provided competition data and unseen evaluation data. In Section 5, we discuss transfer issues on the unseen competition data and how future work can resolve this problem. Finally, in Section 6, we conclude the paper.

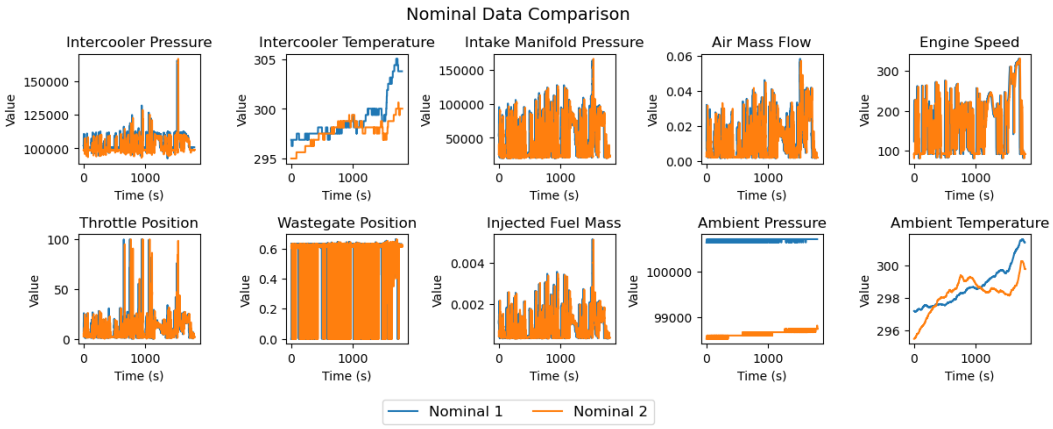


Figure 1 Comparison of two nominal driving cycles across all available signals.

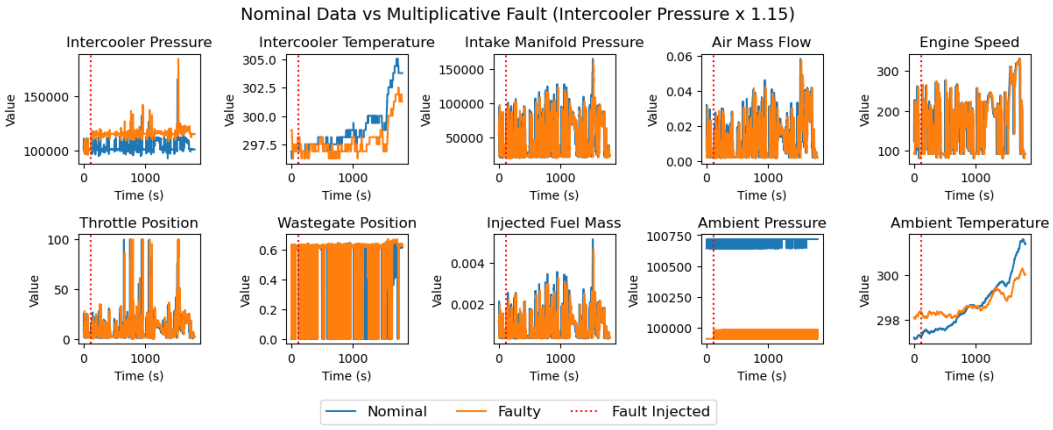


Figure 2 Comparison of a nominal driving cycle with a faulty driving cycle where are 15% higher sensor fault is introduced in the intercooler pressure sensor around 120 seconds into the cycle.

2 Competition and Dataset Preliminaries

This paper describes a solution to the 2025 International Conference on Principles of Diagnosis and Resilient Systems (DX'25) LiU-ICE industrial fault diagnosis benchmark competition [14] first described in [17]. This competition calls for the development of a diagnosis system for the air path of an internal combustion engine.

The data consists of ten signals measured on a real engine test bed at 20Hz. The ten signals consist of eight sensor signals: the intercooler pressure, intercooler temperature, intake manifold pressure, mass flow through the air filter, throttle actuator position, engine speed, ambient pressure, and ambient temperature. There are also two actuator signals: the requested injected fuel mass and the requested wastegate actuator position. Of these, we removed the ambient pressure, ambient temperature, and intercooler temperature, as they were difficult to reconstruct and led to worse fault detection performance. Figure 1 shows the ten available signals on two nominal driving cycles. The intercooler temperature, ambient pressure, and ambient temperature are different despite both being from nominal driving cycles, further justifying their removal as a preprocessing step.

In this challenge, there are four faults of varying magnitudes. There are three sensor faults and one leakage fault. The three sensor faults are multiplicative faults in the sensors measuring the air mass flow (WAF), intercooler pressure (PIC), and intake manifold pressure (PIM). These were introduced by multiplying the sensor measurement by a constant value of 0.8, 0.85, 0.9, 0.95, 1.05, 1.1, or 1.15 for the remaining duration of the driving cycle. An example showing the PIC fault multiplied by 1.15 (denoted PIC 115 in the dataset) is shown in Figure 2. After the fault was introduced around 120 seconds into the driving cycle, the intercooler pressure sensor measurement increased. The leakage fault was in the intake manifold (IML). A 4mm or 6mm leak was introduced by opening a valve.

The dataset for the competition consists of 25 driving cycles, 1 for each magnitude of sensor fault (7 per fault), 1 for each of the two magnitudes of leakage fault, and 2 nominal trajectories. In the faulty datasets, the fault was introduced around 120 seconds into the cycle and persists until the end of the trajectory. For our experiments, we split the data into sub-datasets used for different tasks in the remainder of the paper. These are as follows.

- **Nominal Dataset:** the two full nominal trajectories representing the nominal state of the system. This was used for training most models.
- **Validation Dataset:** the full trajectories (nominal and faulty data) for the 4mm IML, 080 PIC, 090 PIC, 095 PIC, 105 PIC, 115 PIC, 085 PIM, 090 PIM, 095 PIM, 110 PIM, 115 PIM, 080 WAF, 090 WAF, 095 WAF, 105 WAF, and 115 WAF faults. This totals 16/23 faulty trajectories. This was used for calibrating parameters, calibrating hyperparameters, and evaluating performance while developing our methodology.
- **Testing Dataset:** the full trajectories (nominal and faulty data) for the 6mm IML, 085 PIC, 110 PIC, 080 PIM, 105 PIM, 085 WAF, and 110 WAF faults. This totals 7/23 faulty trajectories. This was used for the evaluation of the generalization performance of our methodology to new fault magnitudes.
- **Full Faulty Dataset:** the full trajectories (nominal and faulty data) of all 23 fault magnitudes. This was used for the full evaluation of the methodology.

The competition is scored using three evaluation metrics, defined as follows.

- **False Alarm Rate (FAR):** the percentage of faults detected when there is no fault in the system.
- **True Detection Rate (TDR):** the percentage of faults detected when there is a fault in the system.
- **Fault Isolation Accuracy (FIA):** the average probability given to the actual fault when a fault is correctly detected. Note that this is not the fault classification accuracy.

The final score is determined by averaging the three evaluation metrics above (taking 1-FAR instead of the FAR).

3 Method

In this Section, we detail our unsupervised data-driven fault detection and supervised fault isolation methodology. We include details about the general methodology, model training process, and hyperparameter tuning process.

3.1 Fault Detection

First, we perform fault detection. Our fault detection architecture, shown in Figure 3, consists of an autoencoding step to obtain an initial unsupervised fault label and a rule-based filtering step to reduce false positives caused by noisy samples.

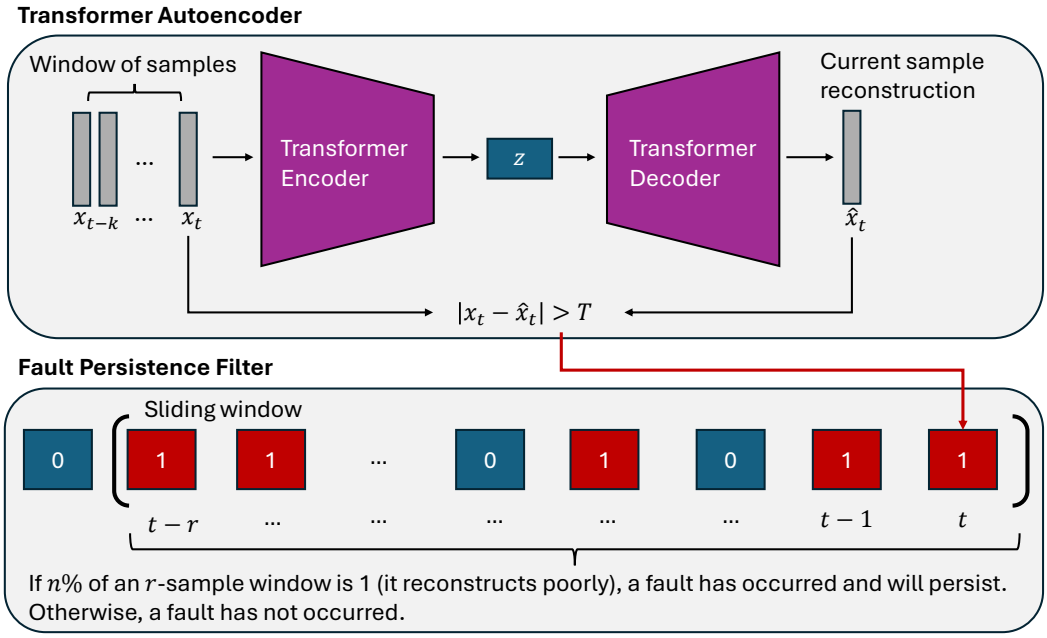


Figure 3 Fault detection architecture.

3.1.1 Transformer Autoencoder

To detect faults, we train a Transformer Autoencoder model on the nominal training trajectories. Autoencoder models are commonly used for unsupervised anomaly detection [20]. Autoencoders compress data to a lower dimensional space then reconstruct the original data from the lower-dimensional representation. By learning this task on nominal data in a semi-supervised way, an autoencoder can learn a representation of the nominal behavior of the system. It operates on the key assumption that the it will not generalize to unseen data distributions and will therefore only reconstruct nominal data well. Then, if anomalous (likely faulty) data is fed to the autoencoder, it will reconstruct it poorly. A threshold can be set on the reconstruction error to assign binary fault detection classifications.

More formally, we have an encoder neural network $\phi : X \to Z$ where $X \in \mathbb{R}^n$ is the space of the measured data and $Z \in \mathbb{R}^k$ is the space of compressed representations and $k < n$. We also have a decoder neural network $\psi : Z \to X$ that reconstructs the input data. Then, the autoencoder operation can be formulated as follows

$$\hat{x} = \psi(\phi(x)), \tag{1}$$

where $x \in X$ is an input sample and $\hat{x}$ is its reconstruction. An autoencoder is typically trained to minimize the reconstruction error on nominal data, quantified using the mean-squared error loss as follows

$$\mathcal{L}_{\text{AE}}(x) = ||x - \hat{x}||_2^2. \tag{2}$$

To determine whether the input x is faulty, its reconstruction error can be compared against a threshold T determined by applying the autoencoder to nominal data, i.e., it is faulty if $|x - \hat{x}| > T$.

The autoencoder framework allows for freedom in choosing the encoder and decoder networks. For the system studied in this paper, we use Transformer neural networks. Transformers [21] are sequence-based neural networks that use self-attention mechanisms

Table 1 Transformer Autoencoder optimized hyperparameters.

Batch Size	Latent Dim.	Trans. Dim.	Feedforward Dim.	Δ Thresh.	Dropout %
128	4	32	128	-0.02	28.1
Learn. Rate	Num. Epochs	Num. Heads	Num. Layers	Sequence Len.	-
0.003	8	2	3	5	

instead of recurrence. Self-attention allows each data point in the sequence to update its representation using every other point in the sequence in parallel. Using this mechanism Transformers have been shown to outperform Recurrent neural network-based approaches in time series domains like speech [15]. By using a powerful sequential model like a Transformer as the encoder and decoder of the autoencoder, we can learn potential temporal relationships in the data and leverage them for fault detection. The architecture of this model, taking a window of inputs and reconstructing the current input, is shown at the top of Figure 3.

As the Transformer Autoencoder model is the first step in our fault detection and isolation pipeline, it is important that it is properly calibrated. To do so, we started by tuning the hyperparameters. We tuned the hyperparameters using the Tree-Structured Parzen Estimator algorithm [3] included in Optuna [2], allowing for an intelligent optimization of both discrete and continuous hyperparameters. We optimized the hyperparameters that maximized the detection score, a modification of the competition score from Section 2 and quantified as follows

$$\text{Detection Score} = \frac{1}{2}\text{TDR} + \frac{1}{2}(1 - \text{FAR}), \tag{3}$$

where TDR is the true detection rate and FAR is the false alarm rate. This score measures the balance of true fault detections and false alarms. During hyperparameter optimization, the Transformer Autoencoder was trained on the two full nominal trajectories. Data was normalized between 0 and 1 using the minimums and maximums of the training data. An initial threshold vector was chosen as the maximum reconstruction error for each feature across the training sets after convergence. We evaluated the detection score on the 16 validation trajectories, described in Section 2. The optimal hyperparameters after 100 trials are shown in Table 1.

From Table 1, we can see that the Δ threshold, a modification of the reconstruction error threshold chosen as the maximum nominal reconstruction error for each feature, was chosen to be -0.02. To maximize the Detection Score, the hyperparameter optimization decreased the threshold, allowing some false positives. Since this hyperparameter controls the tradeoff between true positive and false positive rate, we performed an additional hyperparameter search over the threshold for each feature independently. This led to the following Δ thresholds. Intercooler pressure $= -0.061$, intercooler temperature $= 0.091$, intake manifold pressure $= 0.086$, air mass flow $= -0.023$, engine speed $= -0.091$, throttle position $= -0.018$, and injected fuel mass $= -0.017$. With different thresholds and Δ thresholds for each, the detection does not have to be sensitive to any one reconstructed feature.

3.1.2 Fault Persistence Filter

While the Transformer Autoencoder trained with optimized hyperparameters detects faults reasonably well (Detection Score$= 0.7$ on validation data), it is sensitive to noise in the data, causing false positives. Additionally, it may miss some detections after the fault occurs. To

address this issue, we develop a rule-based fault filtering approach. This approach assumes that once a fault is introduced, it will persist until the end of the trajectory. In other words, there are no intermittent faults.

The rule-based fault persistence filtering, shown in the bottom block of Figure 3, keeps track of the fault detection predictions by the Transformer Autoencoder. If $n\%$ of the last r predictions are a 1 (there is a fault), then the filter deems the system faulty and outputs 1 for the remainder of the trajectory. Otherwise, the system is not considered faulty, so the filter outputs 0. More formally, assume the detections are being tracked in a boolean queue $detects \in \{0,1\}^r$, filled with 0s at the start of each trajectory, and $detects^t$ is the state of the queue at time t. Then, the filtering logic is as follows

$$\text{Persist}(detects^t) = \begin{cases} 1, & \text{if } (\frac{100}{r}\sum_{i=0}^{r} detects_i^t) \geq n \text{ or } \text{Persist}(detects^{t-1}) = 1 \\ 0, & \text{else.} \end{cases} \tag{4}$$

We tuned the parameters n and r manually on the validation set using the Transformer Autoencoder model trained with optimized hyperparameters. We found that $n = 80\%$ and $r = 140$ (7 seconds) worked best.

3.2 Fault Isolation

Once a fault is persistently detected, we isolate the fault. Our fault isolation architecture, shown in Figure 4, consists of two phases. First, we train neural networks that perform a semi-supervised feature estimation based on relationships determined from structural analysis of the provided partial model. Then, we classify the fault probabilities using a supervised fault classification neural network that takes the feature estimation model errors as input to use the system structure information for fault isolation.

3.2.1 Data-Driven Residual Generation Through Structural Analysis

To isolate faults, we first determine relationships between features by performing a structural analysis of the partial system model provided in the competition.

Structural analysis defines a set of methods that allow to characterize the analytical redundancy in a system that can be exploited for fault detection and diagnosis. Obtaining data-driven residuals through structural analysis begins with defining the analytical model of the dynamic system that describes the underlying physics that drive the system's behavior, typically through differential algebraic equations. A structural model is represented as a bipartite graph or a logical/incidence matrix that describes the qualitative relationship between model equations and variables [13]. The Dulmage-Mendelsohn decomposition is applied to the structural model to partition it into under-determined, exactly determined, and over-determined parts. The over-determined part contains analytical redundancy – meaning there are more equations than unknown variables.

Within the over-determined part of the model, structural analysis methods identify redundant equation sets. Of particular interest are the Minimally Structurally Over-determined (MSO) sets. MSO sets represent the minimal redundant equation sets that can be used for residual generation, corresponding to the smallest parts of the system that can be monitored separately [17, 18]. From these MSO sets, computational graphs are derived by designating one equation as the residual and establishing a causal computation sequence for other variables from known signals. In Figure 4, the green boxes in the MSO matrix indicate the variable that can be represented using the other (blue) known signals. This derived structure is used to design a data-driven model, where non-linear functions and dynamic

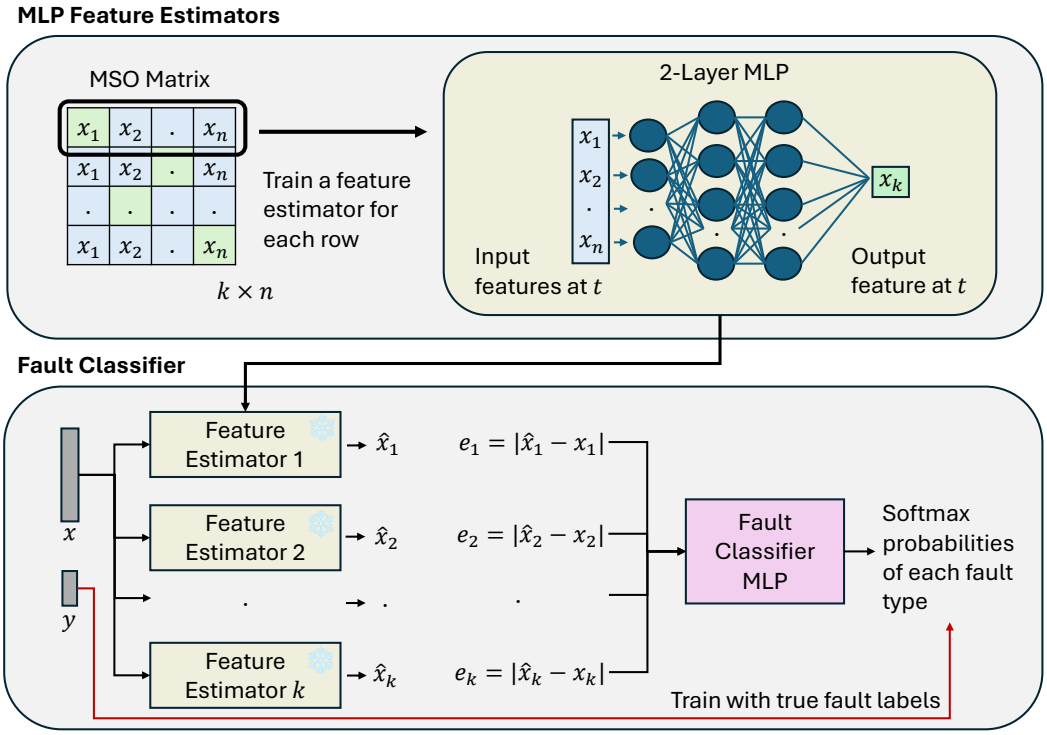


Figure 4 Fault isolation architecture. The snowflake indicates the model weights are frozen.

states from the computational graph are mapped to components like Multi-layer Perceptrons (MLPs) within the network [13, 17, 18]. These models are intentionally designed to resemble the physical system's structure, embedding physical insights into the data-driven approach. These models are then trained exclusively on nominal (fault-free) data to accurately learn the system's expected behavior under normal conditions. Training aims to minimize the model's prediction error for a target signal.

Finally, the data-driven residual is generated by calculating the difference between the actual measured value of the target signal and the output predicted by the trained data-driven model. This process results in residuals that act as anomaly detectors, highlighting deviations from expected nominal behavior, which is particularly useful when data from faulty scenarios is limited.

3.2.2 MLP Feature Estimators

Based on the MSO matrix, we use a simple test selection strategy that finds sets of tests that, ideally, fulfills specified isolability performance specifications [16]. For each MSO in the set $MSO_1 : 383, MSO_2 : 385, MSO_3 : 512, MSO_4 : 519$, we can use the system relationships to generate feature estimator models. According to the structural analysis, these four can be used to isolate any of the known faults, in theory (see Figure 5 for the isolability matrix and fault signature matrix). As shown at the top of Figure 4, we train one multi-layer perceptron neural network model for each row in the MSO matrix. These estimate the target feature using the predictor features and are small, 2-layer 64-neuron networks. The hyperparameters of these networks were empirically determined and kept fixed. They are trained using the

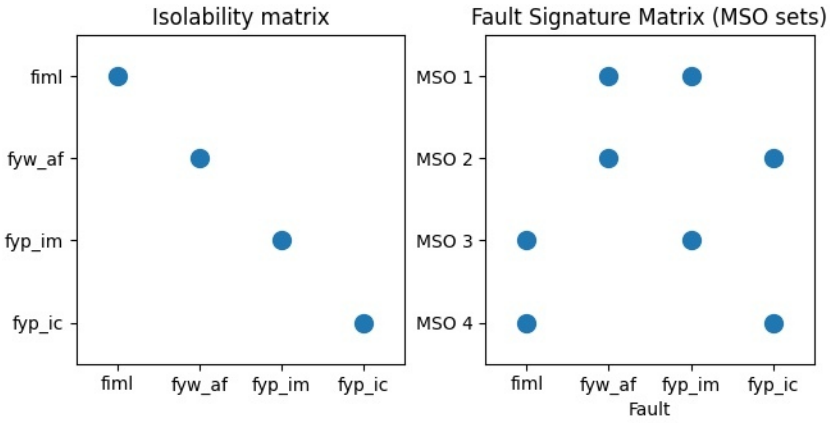


Figure 5 Result of the structural analysis.

mean-squared error. These models can be used to isolate a fault by checking whether their estimation error is above a threshold. When this happens, we say a model "triggers". This can be expressed for a model f_i as follows,

$$\text{Triggered} = |f(x_{1,\dots,n}) - x_k| > T, \tag{5}$$

where $x_{1,\dots,n}$ are the predictive features, x_k is the target feature, and T is some threshold. By comparing to the threshold, we can quantify the model's absolute estimation error. In the future, we will explore more advanced architectures like grey-box Recurrent Neural Networks [13] or Neural Ordinary Differential Equations [18].

3.2.3 Fault Classification

With the trained feature estimators for each MSO set, we can theoretically isolate faults. See Figure 5; all faults are isolable. To isolate the IML fault (column 1 of the rightmost plot), for example, we can check whether the feature estimators for MSO 3 and 4 trigger while the other two do not. The other faults could be isolated following the same pattern.

We could isolate faults with moderate success following this method. However, the reality of the noisy, limited data made consistently isolating all faults impossible, even with added rule-based filtering and threshold tuning. To address this issue, we leveraged the labeled faulty data to train a supervised fault classifier.

The fault classifier, as shown at the bottom of Figure 4, replaces the error thresholding and fault signature matrix lookup with a neural network. This allows us to represent complex relationships between the feature estimation errors while maintaining the structure of the MSO set models. The classifier takes the feature estimation error as input and outputs the softmax probabilities of each of the four fault types. We trained this model on all available anomalous data to ensure it was accurate on all fault magnitudes. During training, the weights of the feature estimators are frozen. We found that training this fault classifier allows to better characterize the feature residual space than thresholding techniques or statistical tests like CUSUM.

In evaluation mode, we process the softmax probabilities of the classifier to fit the competition format. First, we check whether the fault is unknown. If none of the four fault types has a softmax probability above 0.35, determined by analyzing the average probabilities across the anomalous data, we classify the fault as unknown. Otherwise, we classify the fault as the fault with maximum probability. We isolate these faults with 100% confidence for the competition, as the scoring encourages confident results.

Table 2 Fault detection performance broken down by fault type and magnitude.

Type	Magnitude	TDR	FAR	Detection Score
IML	4mm	0.14	0.00	0.57
IML	6mm	0.95	0.00	0.98
PIC	080	0.96	0.00	0.98
PIC	085	1.00	0.00	1.00
PIC	090	1.00	0.00	1.00
PIC	095	1.00	0.00	1.00
PIC	105	1.00	0.00	1.00
PIC	110	0.99	0.00	1.00
PIC	115	1.00	0.00	1.00
PIM	080	0.97	0.00	0.99
PIM	085	0.97	0.00	0.99
PIM	090	0.95	0.00	0.97
PIM	095	0.94	0.00	0.97
PIM	105	0.61	0.00	0.81
PIM	110	0.62	0.00	0.81
PIM	115	0.92	0.00	0.96
WAF	080	0.98	0.00	0.99
WAF	085	0.85	0.00	0.93
WAF	090	0.46	0.00	0.73
WAF	095	0.91	0.00	0.96
WAF	105	0.85	0.00	0.93
WAF	110	0.94	0.00	0.97
WAF	115	0.98	0.00	0.99
Average:		**0.87**	**0.00**	**0.93**

4 Results

To asses our approach, we evaluated the fault detection and fault isolation components on two sets of data. The first was the data given for the competition. The second was an unseen dataset used to evaluate competition submissions.

4.1 Results on Competition Data

Using the labeled faulty and nominal data from the competition, we evaluated both the fault detection and isolation performance of our model using the competition's evaluation metrics defined in Section 2.

4.1.1 Fault Detection

The results for the fault detection performance on all faulty data are shown in Table 2. This table shows the True Detection Rate (TDR), False Alarm Rate (FAR), and Detection Score (from Equation (3)) broken down by fault type and magnitude. From this, we can see that the proposed fault detection architecture detected 87% of faults with a 0% False Alarm Rate. This led to a Detection Score of 0.93. While some number of false positives may have led to a higher Detection Score, minimizing false positives is necessary for real applications.

The low FAR can be attributed to the fault persistence filter (see the bottom of Figure 3). This filter divides the current trajectory into nominal and faulty states. When 80% of the last 7 seconds have been flagged as anomalous by the Transformer Autoencoder, it considers the trajectory to be faulty. Otherwise, any detections by the autoencoder are attributed to noise or smaller anomalies and are suppressed.

While the fault persistence filter reduced the FAR, it also caused some faults to be detected late. For example, consider the 4mm IML fault, shown in Table 2. This fault was only detected during the last 14% of the faulty trajectory. Without the persistence filter, fault detections may have appeared earlier, at the cost of additional false alarms. With a more sensitive persistence filter, e.g., 50% of the last 3 seconds, this fault may have been consistently detected earlier. However, the chosen filter parameters maximized the overall Detection Score.

Next, we can analyze which fault types and magnitudes were more difficult to detect. From Table 2, we can see that the PIC faults were nearly always perfectly detected. All PIM faults where the sensor reading was reduced (magnitude ≤ 095) were detected with more than 90% accuracy. When the sensor reading was increased by 5 and 10% (magnitudes 105 and 110), the detection rate dropped to around 60%. The only WAF fault detected with a rate less than 85% was when the sensor reading decreased by 10%. In this case, the fault was not detected consistently enough when it was introduced around 120 seconds into the trajectory to satisfy the persistence filter. A consistent anomalous behavior, according to the Transformer Autoencoder reconstructions, did not appear until around 54% into the trajectory. Since WAF faults of lower magnitude were consistently detected, this can likely be attributed to the driving behavior of this specific trajectory. For example, the impact of a WAF fault may not be clear during long periods of consistent driving speeds. Finally, the 4mm IML fault was detected much later than the 6mm IML fault. The smaller leak had a much more subtle impact on the system, causing it to be harder to detect and isolate.

4.1.2 Fault Isolation

Next, we evaluated our fault isolation architecture on the faulty datasets. The average probabilities assigned to each fault type on each dataset are shown in Table 3. The overall Fault Isolation Accuracy (FIA), as defined by the competition description in Section 2, was 0.738. This means the true fault was assigned a 73.8% probability on average across the fault types and magnitudes.

For the sensor faults, this was higher. For example, the PIC fault, which was the most consistently detected, had an average FIA of 82.8%. On the other hand, the IML fault was not isolated with high probability. For the 4mm magnitude IML fault, the model assigned nearly equal average probabilities to the WAF and IML faults. At 6mm magnitude, the IML fault was assigned the highest probability, but still below 35% on average. This highlights the difficulty of detecting and isolating the leakage faults over the sensor faults in this system.

From Table 3, we can see that the average probability of an unknown fault was always low, below 5%. Since all this data was used for training the final fault classifier and none of the faults are unknown to the model, this is an expected and positive result. Future work should evaluate the model on new fault types to determine the effectiveness of the architecture in identifying unknown faults.

Next, we can analyze the impact of sensor fault magnitude on the isolation performance. From Table 3, we can see that the WAF fault was isolated consistently for all magnitudes. However, the PIC and PIM faults were correctly isolated with a higher average probability as the fault magnitude increased. Figure 6 shows the average probabilities assigned to these

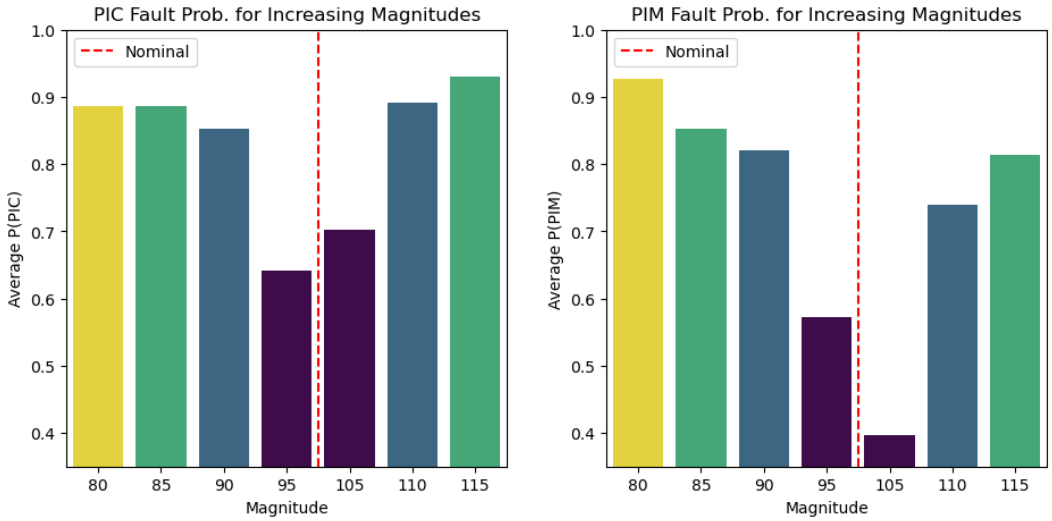
Table 3 Average fault isolation probabilities broken down by fault type and magnitude. Overall performance is measured through the Fault Isolation Accuracy, shown in the bottom row.

Type	Magnitude	P(PIC)	P(PIM)	P(WAF)	P(IML)	P(Unknown)
IML	4mm	0.086	0.193	0.337	**0.336**	0.048
IML	6mm	0.059	0.245	0.310	**0.347**	0.039
PIC	080	**0.887**	0.026	0.085	0.001	0.001
PIC	085	**0.887**	0.036	0.075	0.001	0.001
PIC	090	**0.853**	0.029	0.111	0.002	0.004
PIC	095	**0.641**	0.093	0.214	0.019	0.034
PIC	105	**0.703**	0.071	0.170	0.026	0.030
PIC	110	**0.892**	0.033	0.055	0.008	0.012
PIC	115	**0.930**	0.021	0.047	0.001	0.001
PIM	080	0.003	**0.927**	0.057	0.011	0.002
PIM	085	0.005	**0.853**	0.124	0.014	0.003
PIM	090	0.010	**0.821**	0.141	0.022	0.005
PIM	095	0.044	**0.573**	0.303	0.044	0.035
PIM	105	0.028	**0.396**	0.556	0.011	0.010
PIM	110	0.030	**0.740**	0.196	0.022	0.013
PIM	115	0.014	**0.813**	0.137	0.029	0.007
WAF	080	0.027	0.213	**0.752**	0.005	0.003
WAF	085	0.016	0.119	**0.821**	0.025	0.019
WAF	090	0.033	0.154	**0.789**	0.011	0.013
WAF	095	0.042	0.213	**0.708**	0.020	0.017
WAF	105	0.038	0.181	**0.759**	0.011	0.011
WAF	110	0.018	0.213	**0.755**	0.005	0.008
WAF	115	0.032	0.142	**0.788**	0.023	0.015
Fault Isolation Accuracy:			**0.738**			

fault types as the fault magnitude increased (the sensor reading was increased or decreased by 5, 10, 15, then 20%). As the magnitudes got further away from their nominal value, which would be represented by 100, the fault was better isolated. This result is consistent with intuition, as more extreme faults should have a more dramatic impact on the system and should therefore be easier to isolate.

Since the competition metrics included in Table 3 aggregate the isolation probabilities across the driving trajectories, we cannot analyze the behavior of the fault isolation across time. To better understand this, we plotted the isolation probabilities over time for the 4mm IML fault, 10% lower magnitude WAF fault, and 15% higher magnitude PIC fault in Figure 7. Since the fault isolation architecture assigns 100% confidence in a fault when its softmax probability is above 0.35, Figure 7 shows a running average of 20 of these probabilities to show the average probability trend.

First, we can consider the most difficult fault to detect and isolate, the 4mm IML fault shown in Figure 7a. Throughout the trajectory, this fault was consistently misidentified as the WAF and PIM faults. However, there are long periods where the IML fault is properly isolated, such as around 450s and 1000s. Even though the IML fault was not assigned the highest average probability, this consistency may lead us to classify the fault type of the trajectory as an IML fault, pointing maintainers to the intake manifold valve first.



(a) PIC Fault. **(b)** PIM Fault.

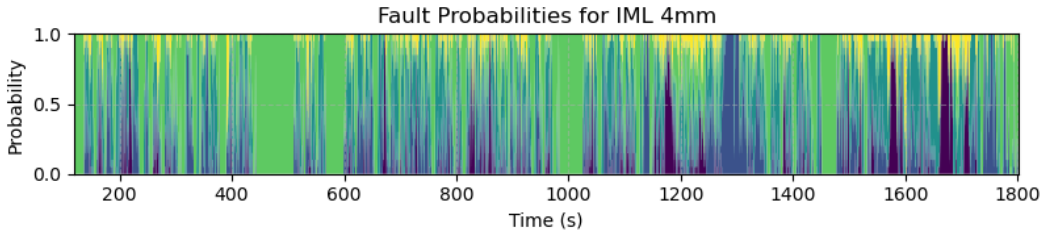
Figure 6 Isolation probabilities for increasing magnitudes of PIM and PIC faults. The red dotted line is the nominal magnitude. As the fault magnitudes increase, getting further from the red line, the average probabilities assigned by the isolation algorithm increase.

Table 4 Performance on competition evaluation data. This data was unseen during the model development phase and used for final competition scoring. The framework was evaluated on data from the same fault magnitudes but generated with a different driving cycle, city driving.

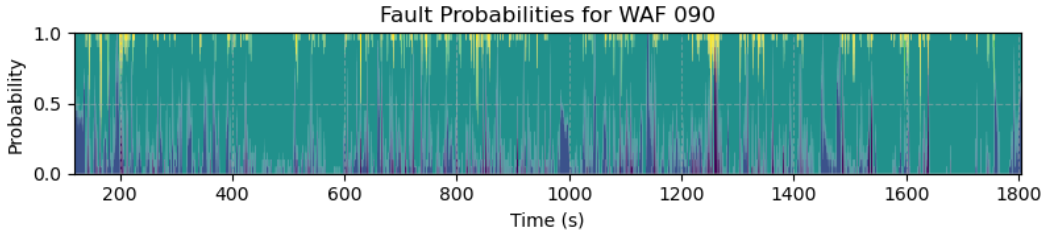
Fault Type	TDR	FAR	FIA	Time (ms)
Nominal	–	0.913	–	2.1
PIC	1.0	0.393	0.894	4.0
PIM	1.0	0.388	0.495	3.5
WAF	1.0	0.731	0.607	5.0
Average:	**1.0**	**0.526**	**0.662**	**3.65**
Score:	**71.2**			

Next, we can consider a sensor fault that was isolated well but detected poorly, the 10% lower magnitude WAF fault shown in Figure 7b. Even though this fault was detected late, over halfway through the trajectory, the isolation algorithm consistently isolated the WAF fault with high probability before it was detected. This indicates that the impact of the fault was present in the data and that the late detection was likely due to the detection not being consistent enough early to pass the persistence filter. Additionally, when the WAF fault was not correctly isolated, it was largely classified as a PIM fault.

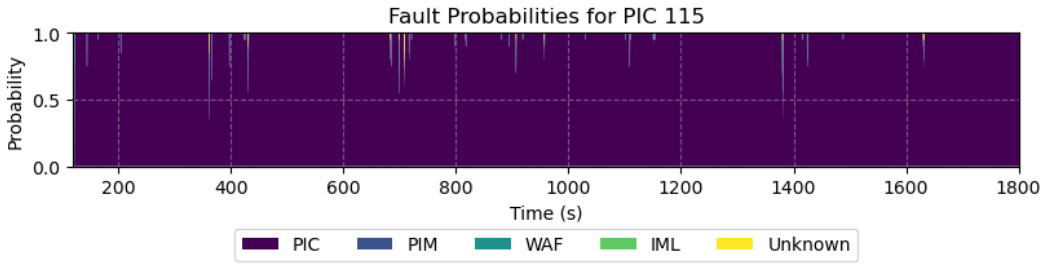
Finally, we can analyze the probabilities for the fault that was isolated best, the 15% higher PIC fault shown in Figure 7c. This fault was nearly perfectly isolated and was given 100% probability for most of the trajectory.



(a) 4mm IML Fault.



(b) 10% Lower WAF Fault (090).



(c) 15% Higher PIC Fault (115).

Figure 7 Stacked isolation probabilities over time for 4mm IML fault (a), 10% lower WAF fault (b), and 15% higher PIC fault (c). The fault probabilities are averaged over the last 20 predictions, starting after the fault occurs. The stacked bars have a fixed order, so the probability is denoted by the size of the bar at each time, not its proximity to the top of the y-axis.

4.2 Results on Unseen Data (Competition Results)

After the evaluation on the given competition data, the full fault detection and isolation architecture was combined and submitted. In the combined architecture, the fault isolation was only used when a fault was detected after the persistence filter. For the final competition scoring, the architecture was evaluated on data from the same fault types and magnitudes, without IML faults, generated using a different driving cycle representing city driving.

The results on this unseen data are shown in Table 4. The FAR was much higher, at 91.3% on nominal data and 73.1% on data from WAF faults. On PIC and PIM faults, the FAR was around 39%. In real applications, this many false alarms may be unacceptable. The Transformer Autoencoder needs to be recalibrated to the new distribution of nominal driving data. An in-depth discussion into why unsupervised autoencoders are so sensitive in situations like this and how to address this problem is provided in Section 5.

In terms of the other metrics, the architecture's performance was much closer to the performance on the provided competition data. It achieved a perfect detection rate (at the expense of false alarms). It also made its predictions in less than 5 milliseconds on average, well within the data rate of 50 milliseconds. Therefore, the predictions could be

used for real-time decisions. Finally, the FIA was 66.2% on average, close to the 77.6% average isolation probability for the same fault types on the training data. Overall, the final competition score was 71.2% achieving first place, with second place scoring 60.5%.

5 Discussion on Reducing Autoencoder False Alarms

While autoencoders can be very powerful for unsupervised anomaly detection, they are limited in their generalization to unseen data distributions. In a way, this is by design. Recall from Section 3 that the Transformer Autoencoder was trained to reconstruct nominal data. Then, its failure to reconstruct data from unseen distributions (anomalous data) can be used for fault detection. However, when nominal data is also out of distribution, the autoencoder may fail to reconstruct it well and classify it as faulty. This is why there was a high False Alarm Rate on the competition data generated using a different driving cycle (Section 4.2).

A simple way to reduce false alarms would be to fine-tune the autoencoder on data from the new driving cycle, ensuring it learns a more complete representation of the system's nominal behavior, like in [11]. However, this would not permanently solve the problem, as any new out-of-distribution nominal data, such as from new driving cycles or environmental factors, might be considered anomalous. In some cases, it may be reasonable to assume the system will not encounter new nominal conditions, such as in controlled environments and testbeds. However, in many real-world applications, the environment and system usage may be nonstationary [7]. Therefore, the autoencoder must constantly adapt to new conditions.

A promising potential solution to reducing the false alarms introduced in this architecture by the Transformer Autoencoder lies in *continual learning* [23]. In continual learning, a model is continually adapted throughout the lifetime of its deployment. In other words, it never stops learning. This could be used to adapt the Transformer Autoencoder model to all experienced nominal conditions, reducing false alarms. This adaptation could be done online by identifying data that is assumed to be nominal or immediately after a trajectory by identifying nominal periods of data.

Aside from improving the performance of the autoencoder for fault detection on new data distributions, it could be replaced with different fault detection approaches. Many anomaly and fault detection approaches may work well and should be considered in future studies. For example, contrastive learning [12], density estimation [19], and clustering [27] have all been used for unsupervised anomaly or fault detection. However, it may not be reasonable to expect any purely data-driven approach to transfer to new data distributions without adaptation. Instead, leveraging physical information, such as by using structural analysis in our isolation model or with other hybrid approaches like [8], may be required.

6 Conclusion

In this paper, we introduced and evaluated a data-driven fault detection and isolation architecture for the 2025 DX LiU-ICE competition that leveraged system structural information. Our detection model started with a Transformer Autoencoder that used time series information to reconstruct nominal data. To reduce false alarms caused by noise, we added a rule-based filtering to only consider a trajectory to be in a faulty state when a large percentage of the last seconds was predicted to be faulty. Once the filter detected a fault, the fault was isolated. The isolation model used four Multi-layer Perceptron (MLP) neural network models that estimated features according to rows of an MSO matrix determined from structural analysis. The errors of these feature estimation models were used as an enhanced feature space by an MLP supervised fault classifier that predicted the probabilities of each fault.

Using our architecture, we achieved an 87% fault detection rate with a 0% false alarm rate on the provided competition data. The sensor faults were confidently and consistently isolated, leading to an average of 73.8% confidence applied to the correct fault across the provided dataset. Intake manifold leakage faults were harder to detect and isolate, especially at the smaller magnitude of 4mm with it being detected in the last 14% of the trajectory and only being given 33.6% probability during isolation. We found that as the PIM and PIC sensor fault magnitudes increased, so did the confidence in their isolation. Finally, we visualized the isolation probabilities throughout some trajectories to concretely show how the isolation architecture performed.

On data from a new distribution from a different driving cycle used for the final competition scoring, our architecture had an average false alarm rate of 52.6%, much higher than on the provided data. We hypothesize that this was due to the dependence of the Transformer Autoencoder on a complete representation of nominal data and explained ways to improve this in future work. Despite the higher false alarm rate, our model achieved a 100% detection rate and an average of 66.2% confidence in the correct fault type within 3.65ms on the unseen competition data. This led to a score of 71.2%, achieving first place. Future work should focus on ways to reduce the false alarm rate on the unseen competition data, such as by fine-tuning or incorporating continual learning into the detection autoencoder. Future work should also explore the application of this architecture to systems with different complexities and data characteristics and perform further ablation studies to understand what components of the proposed approach contributed toward its success.

References

1 Anam Abid, Muhammad Tahir Khan, and Javaid Iqbal. A review on fault detection and diagnosis techniques: basics and beyond. *Artificial Intelligence Review*, 54(5):3639–3664, 2021. doi:10.1007/S10462-020-09934-2.

2 Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019. doi:10.1145/3292500.3330701.

3 James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

4 Lucas C Brito, Gian Antonio Susto, Jorge N Brito, and Marcus AV Duarte. An explainable artificial intelligence approach for unsupervised fault detection and diagnosis in rotating machinery. *Mechanical Systems and Signal Processing*, 163:108105, 2022.

5 Zhelun Chen, Zheng O'Neill, Jin Wen, Ojas Pradhan, Tao Yang, Xing Lu, Guanjing Lin, Shohei Miyata, Seungjae Lee, Chou Shen, et al. A review of data-driven fault detection and diagnostics for building hvac systems. *Applied Energy*, 339:121030, 2023.

6 Seongpil Cho, Zhen Gao, and Torgeir Moan. Model-based fault detection, fault isolation and fault-tolerant control of a blade pitch system in floating wind turbines. *Renewable energy*, 120:306–321, 2018.

7 Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015. doi:10.1109/MCI.2015.2471196.

8 Stephen Frank, Michael Heaney, Xin Jin, Joseph Robertson, Howard Cheung, Ryan Elmore, and Gregor Henze. Hybrid model-based and data-driven fault detection and diagnostics for commercial buildings. Technical report, National Renewable Energy Lab.(NREL), Golden, CO (United States), 2016.

9 Zhiwei Gao, Carlo Cecati, and Steven X Ding. A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. *IEEE transactions on industrial electronics*, 62(6):3757–3767, 2015. doi:10.1109/TIE.2015.2417501.

10 Janos J Gertler. Survey of model-based failure detection and isolation in complex plants. *IEEE Control systems magazine*, 8(6):3–11, 1988.

11 Yuxin Huang, Austin Coursey, Marcos Quinones-Grueiro, and Gautam Biswas. Time-series few shot anomaly detection for hvac systems. *IFAC-PapersOnLine*, 58(4):426–431, 2024.

12 Yang Jiao, Kai Yang, Dongjing Song, and Dacheng Tao. Timeautoad: Autonomous anomaly detection with self-supervised contrastive loss for multivariate time series. *IEEE Transactions on Network Science and Engineering*, 9(3):1604–1619, 2022. doi:10.1109/TNSE.2022.3148276.

13 Daniel Jung. Residual generation using physically-based grey-box recurrent neural networks for engine fault diagnosis, 2020. arXiv:2008.04644.

14 Daniel Jung, Erik Frisk, and Mattias Krysander. LiU-ICE Industrial Fault Diagnosis Benchmark - DeluXe 2025. https://vehsys.gitlab-pages.liu.se/dx25benchmarks/liuice/liuice_index, 2025. Accessed: 2025-06-03.

15 Shigeki Karita, Nanxin Chen, Tomoki Hayashi, Takaaki Hori, Hirofumi Inaguma, Ziyan Jiang, Masao Someki, Nelson Enrique Yalta Soplin, Ryuichi Yamamoto, Xiaofei Wang, et al. A comparative study on transformer vs rnn in speech applications. In *2019 IEEE automatic speech recognition and understanding workshop (ASRU)*, pages 449–456. IEEE, 2019.

16 Johan De Kleer. Hitting set algorithms for model-based diagnosis, 2011.

17 Andreas Lundgren and Daniel Jung. Data-driven fault diagnosis analysis and open-set classification of time-series data. *Control Engineering Practice*, 121:105006, 2022.

18 Arman Mohammadi, Mattias Krysander, and Daniel Jung. Consistency-based diagnosis using data-driven residuals and limited training data. *Control Engineering Practice*, 159:106283, 2025. doi:10.1016/j.conengprac.2025.106283.

19 Benjamin Nachman and David Shih. Anomaly detection with density estimation. *Physical Review D*, 101(7):075042, 2020.

20 Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*, 54(2):1–38, 2021. doi:10.1145/3439950.

21 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

22 Jinxin Wang, Xiuquan Sun, Chi Zhang, and Xiuzhen Ma. An integrated methodology for system-level early fault detection and isolation. *Expert Systems with Applications*, 201:117080, 2022. doi:10.1016/J.ESWA.2022.117080.

23 Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

24 Haiyue Wu, Matthew J Triebe, and John W Sutherland. A transformer-based approach for novel fault detection and fault classification/diagnosis in manufacturing: A rotary system application. *Journal of Manufacturing Systems*, 67:439–452, 2023.

25 Qiao-Ning Xu, Kok-Meng Lee, Hua Zhou, and Hua-Yong Yang. Model-based fault detection and isolation scheme for a rudder servo system. *IEEE Transactions on industrial electronics*, 62(4):2384–2396, 2014. doi:10.1109/TIE.2014.2361795.

26 Xuyang Yan, Mrinmoy Sarkar, Benjamin Lartey, Biniam Gebru, Abdollah Homaifar, Ali Karimoddini, and Edward Tunstel. An online learning framework for sensor fault diagnosis analysis in autonomous cars. *IEEE Transactions on Intelligent Transportation Systems*, 24(12):14467–14479, 2023. doi:10.1109/TITS.2023.3305620.

27 Christos T Yiakopoulos, Konstantinos C Gryllias, and Ioannis A Antoniadis. Rolling element bearing fault detection in industrial environments based on a k-means clustering approach. *Expert Systems with Applications*, 38(3):2888–2911, 2011. doi:10.1016/J.ESWA.2010.08.083.

Unsupervised Multimodal Learning for Fault Diagnosis and Prognosis – Application to Radiotherapy Systems

Kélian Poujade[1]
Université de Toulouse, Oncopole Claudius Regaud, Institut Universitaire du Cancer de Toulouse (IUCT), France
Université de Toulouse, CNRS, INSERM, Centre de Recherches en Cancérologie de Toulouse (CRCT), France

Louise Travé-Massuyès
Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS), Université de Toulouse, CNRS, France

Jérémy Pirard
Airbus, Toulouse, France

Laure Vieillevigne
Université de Toulouse, Oncopole Claudius Regaud, Institut Universitaire du Cancer de Toulouse (IUCT), France
Université de Toulouse, CNRS, INSERM, Centre de Recherches en Cancérologie de Toulouse (CRCT), France

Abstract

Modern complex systems, such as radiotherapy machines, require robust strategies for fault detection, diagnosis, and prognosis to ensure operational continuity and patient safety. While data-driven methods have gained traction, few studies address diagnostic and prognostic tasks using multimodal operational data under unsupervised or semi-supervised learning settings. This gap is particularly critical given the scarcity of labeled failure data in real-world environments. This work aims to design a unified approach for fault detection, diagnosis, and prognosis using multimodal data in the absence of complete labeling. To this end, autoencoders (AEs) are employed due to their suitability for unsupervised and self-supervised learning, flexibility in handling heterogeneous data, and ability to construct latent representations optimized for various downstream tasks. A specific implementation based on a Long Short-Term Memory β-Variational Autoencoder (LSTM-β-VAE) was developed to detect anomalies in machine logs. This framework is applied to TomoTherapy® systems – a highly complex and under-explored use case within the radiotherapy domain. Initial results demonstrate strong anomaly detection performance on both a public benchmark dataset (HDFS) and a proprietary dataset derived from real-world TomoTherapy® machine faults. Beyond methodology, the paper includes a concise literature review of multimodal learning and data-driven diagnosis and prognosis with a focus on AEs. Based on this review, key research directions are identified for the continuation of the thesis, especially the integration of explainable AI as a means to enhance diagnosis capabilities in the absence of labeled faults.

2012 ACM Subject Classification Computing methodologies → Unsupervised learning

Keywords and phrases Artificial Intelligence, Diagnosis, Prognosis, Radiotherapy machines

Digital Object Identifier 10.4230/OASIcs.DX.2025.16

Category PhD Panel

[1] Corresponding author

36th International Conference on Principles of Diagnosis and Resilient Systems (DX 2025).
Editors: Marcos Quinones-Grueiro, Gautam Biswas, and Ingo Pill; Article No. 16; pp. 16:1–16:17

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This research was supported by Accuray Inc as part of the doctoral funding. It has also benefited from the AI Interdisciplinary Institute ANITI funded by the France 2030 program under the Grant agreement n°ANR-23-IACL-0002.

Acknowledgements This work is the result of a collaborative tripartite agreement involving IUCT-Oncopole, Airbus, and Accuray Inc.

1 Introduction

In complex modern systems, fault detection, diagnosis, and prognosis are critical components for maintaining system integrity and performance. As these systems grow in complexity, traditional rule-based maintenance approaches are increasingly being supplemented – or even replaced – by data-driven methods. These techniques hold promise not only for timely fault detection, but also for enabling robust diagnostic and prognostic capabilities. Despite significant advances, few studies have investigated comprehensive data-driven diagnosis and prognosis frameworks that leverage unlabeled multimodal operational data such as machine logs and time-series sensor measurements. Addressing this limitation is essential, especially in real-world settings, where labeled failure data is scarce and system behavior is often stochastic.

A first step in this direction is fault detection through anomaly detection. We have already explored the use of machine logs for this purpose. As presented in Section 2, a deep learning approach based on β-Variational Autoencoders (β-VAE) was developed to identify abnormal sequences in machine-generated log data. This method demonstrated the feasibility of using logs as a data source to detect anomalies in the absence of explicit fault labels.

Building upon this foundation, the next phases involve exploring data-driven diagnosis and prognosis methodologies, particularly under semi-supervised or unsupervised learning paradigms. Such approaches are more appropriate for the nature of real-world data, where anomalies may be poorly labeled or entirely unlabeled. This progression is illustrated through an original case study focusing on TomoTherapy® systems used in cancer radiotherapy.

In radiotherapy, equipment reliability is critical to ensuring effective and uninterrupted patient treatment. Unplanned faults in radiotherapy systems can lead to treatment delays, rescheduling, and workflow disruptions, all of which can compromise treatment efficacy and negatively affect patient outcomes. The continuity of treatment has a well-documented impact on clinical results, with studies such as [18] demonstrating that extended overall treatment times can negatively impact local tumor control and influence survival rates.
Among the various technologies employed in radiotherapy, TomoTherapy® machines (Accuray, Madison, WI) (Figure 1a) stand out for their high complexity and versatility. These systems integrate advanced image-guided radiotherapy (IGRT) with intensity-modulated radiation therapy (IMRT) in a helical delivery mode [35]. Treatment delivery involves a continuously rotating gantry and a synchronized, translating treatment couch, paired with a binary multi-leaf collimator (MLC) consisting of 64 pneumatically actuated leaves (Figure 1b). This configuration enables fine-tuned modulation, making the system particularly suitable for anatomically extensive or complex treatments such as total body irradiation, craniospinal irradiation, and re-irradiation [54]. Within this system, the MLC is a key subsystem due to its direct role in beam shaping and dose modulation. However, its mechanical and electronic complexity, relying on high-frequency actuation, makes it susceptible to faults. A particularly vulnerable component is the bumper pack – a mechanical absorber designed to cushion the rapid opening and closing movements of the leaves. Faults in the bumper pack can arise

from progressive wear or sudden rupture and may compromise both system performance and treatment accuracy. This work focuses on the MLC subsystem, with the aim of developing an approach that can be readily generalized to other subsystems.

Although preventive maintenance is standard practice in radiotherapy, including for MLC components [2], there is growing interest in transitioning toward predictive maintenance paradigms. Predictive maintenance aims to anticipate equipment failures using real-time or near-real-time operational data, enabling interventions before breakdowns occur. Such an approach can minimize unplanned interruptions, optimize spare parts logistics, and most importantly, safeguard the continuity of care for patients.

To date, research in radiotherapy has focused primarily on performance monitoring techniques. Studies have analyzed trajectory log files from systems such as standard photon linacs (TrueBeam® Varian Medical Systems, Palo Alto, CA) and proton therapy machines, employing methods like threshold-based monitoring [56] and Statistical Process Control (SPC) [1] . Some studies have investigated predictive methods for system fault anticipation, typically relying on data from routine quality control (QC) or assurance (QA) tests [34, 16]. This approach limits real-time detection of incipient faults and restricts the resolution of predictive models. To our knowledge, no studies have explored predictive strategies for monitoring radiotherapy systems using continuously generated operational data.
This study aims to address this gap by developing a predictive framework tailored to TomoTherapy® machines. It proposes to go beyond fault detection by leveraging operational logs and sensor data for fault diagnosis and prognosis in a data-driven manner. Ultimately, this approach aims to improve equipment reliability and support more consistent and effective radiotherapy delivery, thereby enhancing the quality of patient care.

Figure 1 (a) Schema of TomoTherapy® system (model TomoHD™). (b) Schema of main subsystems composing the TomoTherapy® machines.

2 Conducted research

We first explored anomaly detection on machine log data. These logs, which chronicle the states, events, and procedures of the system, represent a valuable but underutilized resource for modeling system behavior and identifying early signs of malfunction. To address the challenges posed by limited labeled data, the approach relies on Autoencoders (AEs) and semi-supervised learning strategy, enabling the model to learn normal patterns without requiring exhaustive labels. A key objective is to construct a latent space that not only

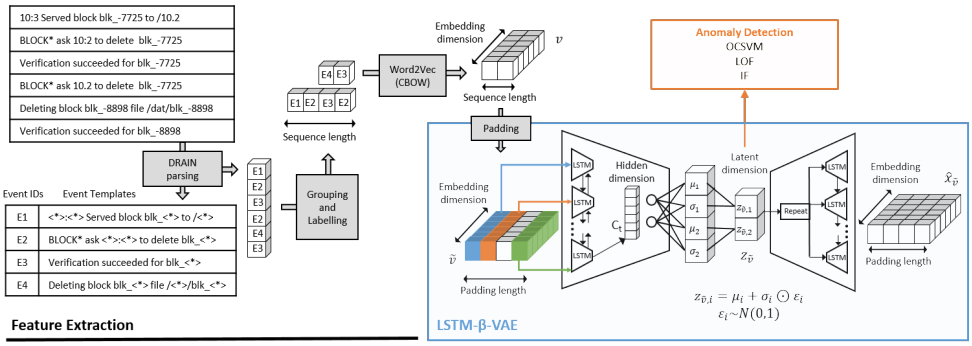


Figure 2 Overview of the proposed pipeline. The first stage includes DRAIN parsing, grouping, labeling and Word2Vec to extract features from log files. Obtained embedded sequences are padded to obtain the input $\tilde{v} \in \tilde{V}$ that feed an LSTM-β-VAE. While reconstructing the input at its output $\hat{x}_{\tilde{v}}$, this later learns compressed latent representations $z_{\tilde{v}} \in Z_{\tilde{v}}$ useful for anomaly detection. Different algorithms are then applied on the learned latent space $Z_{\tilde{v}} = \{z_{\tilde{v}}\}$.

captures the essential structure of normal operations, but also improves interpretability and facilitates the distinction between normal and anomalous behaviors. This framework aims to support more transparent and generalizable anomaly detection in complex systems. A preprint version of this work has been deposited on HAL [42] and the associated code is available online [2].

2.1 Proposed pipeline

We developed a semi-supervised pipeline for anomaly detection in log sequences. It assumes the availability of a subset of logs representing normal system behavior, which is used to train representation learning models. The pipeline consists of three main stages: log preprocessing and feature extraction, latent space learning via a Long Short-Term Memory-based β-Variational Autoencoder (LSTM-β-VAE), and anomaly detection on learned latent space using traditional machine learning algorithms enhanced with conformal prediction. The proposed pipeline is illustrated in Fig. 2. The full pipeline is designed to generalize well across different datasets, relying on minimal tuning and emphasizing the interpretability and robustness of the learned representations.

2.1.1 Log Preprocessing and Feature Extraction

Raw log files are first parsed using the DRAIN algorithm [22], which groups similar log lines into structured templates, allowing them to be represented as sequences of discrete event identifiers. These log lines are then grouped into sequences – either based on procedure windows or sliding time windows – and each sequence is labeled as either normal or faulty. The event identifiers from DRAIN parsing are then embedded into dense vector representations using the Word2Vec model (CBOW variant) [37]. Each log sequence is thus transformed into a sequence of embedding vectors $v \in V$, which are then padded or truncated to a fixed length to obtain $\tilde{v} \in \tilde{V}$. This process ensures compatibility with the subsequent neural architecture.

[2] `https://gitlab.laas.fr/addram/anomaly-detection-in-log-data.git`

2.1.2 Learning Latent Representations

To capture both semantic and sequential characteristics of log sequences, a latent space is learned using a LSTM-β-VAE trained using normal labeled data. This model encodes each embedded sequence $\tilde{v} \in \tilde{V}$ into a low-dimensional latent vector $z_{\tilde{v}} \in Z_{\tilde{v}}$ that captures the key patterns and structure of normal behavior, while reconstructing the input sequence from this compressed representation. The β term in the objective function controls the balance between reconstruction accuracy and latent space regularization [23], allowing better separation of anomalies from normal samples.

2.1.3 Anomaly Detection in the Latent Space

Then, traditional anomaly detection algorithms $\mathcal{A}$ such as One-Class SVM (OCSVM)[48], Isolation Forest (IF)[27], and Local Outlier Factor (LOF)[8] are applied on the learned latent representations $z_{\tilde{v}} \in Z_{\tilde{v}}$. These algorithms are learned in a semi-supervised manner using the latent representations of the normal labeled data used for the LSTM-β-VAE training. To provide statistical guarantees on model predictions, Conformal Anomaly Detection (CAD) – a technique derived from the Conformal Prediction framework [5] – is used. This method allows to define a threshold on anomaly scores returned by the algorithms $\mathcal{A}$ to correct the predictions and guarantee a statistical confidence in the results. Each confidence level is associated to a threshold value.

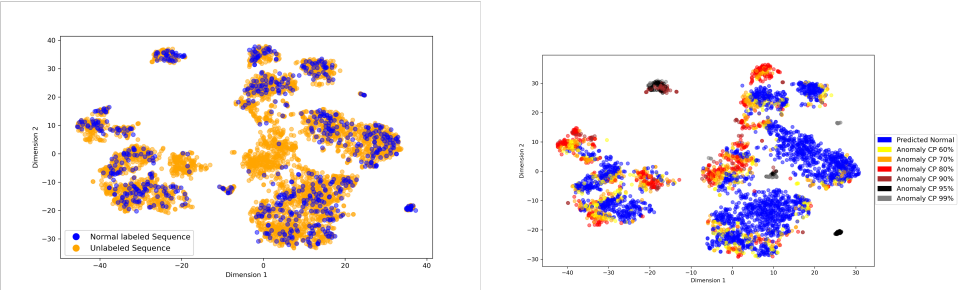
2.2 Evaluation

The evaluation of the proposed pipeline was conducted using two datasets: a publicly available benchmark (HDFS) [66] and a proprietary dataset derived from the logs of TomoTherapy® machines (TOMO). The TOMO dataset focuses on MLC faults linked to the bumper pack, gathering data from 20 fault cases across 15 different machines. Each fault case is referenced as FC(MM/DD/YYYY), identified by the date of the bumper pack repair.

One of the objectives of this study was to investigate the use of Mass-Volume (MV) and Excess-Mass (EM) scores [12, 20] – metrics specifically designed for unlabeled datasets – as a means to optimize the model's parameters and evaluate its performance in the absence of ground truth labels. These metrics evaluate the distance between the level sets of an anomaly scoring function and those of the underlying data distribution. They offer a principled way to assess a model's ability to capture the structure of the data and produce a meaningful anomaly score with statistical rigor.

For the HDFS dataset, both EM and MV scores were computed and compared with conventional classification metrics, including the area under the receiver operating characteristic curve (ROC), the area under the precision-recall curve (PR), and the F1 score. We first demonstrated that using the EM and MV scores to optimize the model and evaluate anomaly detection algorithms in the learned latent space yielded results comparable to those obtained with conventional metrics such as ROC and PR.

Then, for the TOMO dataset – where only a limited number of log sequences were labeled as normal (specifically, those recorded after system repairs) – only the EM and MV scores were employed. These metrics were used to assess the model's performance on a subset of normal labeled sequences unseen during model training.



(a) (b)

Figure 3 T-SNE visualization of log sequences from FC(06/15/2023) using their learned LSTM-β-VAE latent representations. (a) Colored by normal label (blue) and unlabeled (orange). (b) Colored by conformal predictions with different confidence levels based on learned OCSVM scoring function.

2.3 Results

Building on these findings, the EM and MV scores were then used to guide model optimization on the TOMO dataset and to assess the performance of various machine learning algorithms applied in the latent space. To complement this quantitative evaluation, several visualizations of the test data latent representations related to the fault case FC(06/15/2023) were produced using t-distributed stochastic neighbor embedding (t-SNE) algorithm [53], which projects high-dimensional representations into a two-dimensional space for interpretability.

Fig. 3a shows the latent space learned by the LSTM-β-VAE, where a clear separation appears between labeled normal sequences (blue) and a central group of unlabeled sequences (orange), indicating behaviors differing from the normal patterns used during training.

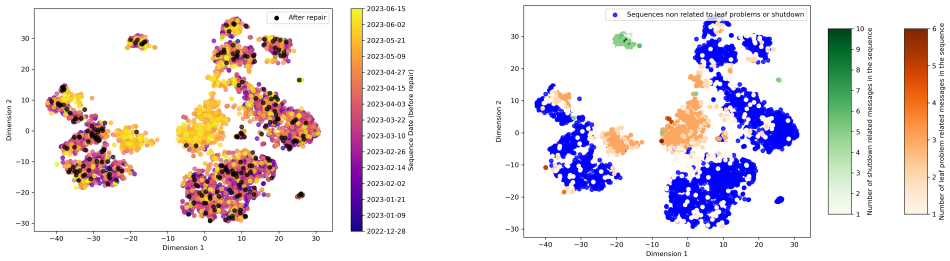
Fig. 3b presents the conformal prediction results of OCSVM, the best-performing algorithm based on EM and MV metrics. The color-coded conformal predictions can be seen as level sets of the OCSVM's scoring function. This figure highlights sequences in the center – isolated from normal ones 3a – as anomalies, mainly with 80% statistical confidence. Additional isolated groups are flagged as abnormal with 90–95% confidence, despite containing sequences labeled as normal.

To contextualize these detections, Fig. 4a maps the detected anomalies to their temporal position relative to subsystem maintenance. Central sequences flagged as abnormal predominantly occurred in the month before the identified fault, with none appearing post-repair, confirming the relevance of the detections.

Finally, Fig. 4b links the detected anomalous groups to specific log message content. Messages related to MLC issues (e.g., overtravel, position, bounce) (orange in Fig. 4b), known indicators of faults, are found in the central anomaly cluster. Another high-confidence anomaly group, detected in black in Fig. 3b and overlapping normal data in Fig. 3a, corresponds to machine shutdowns (green in Fig. 4b), reinforcing the consistency of the detection results with expert knowledge.

3 Thesis roadmap

So far, an approach has been developed to detect anomalous log sequences from the perspective of the system under study. This method leverages AEs to learn, in a semi-supervised manner, a latent space optimized for anomaly detection while preserving interpretability. The next phase of the project aims to enrich this framework by integrating additional data sources,



(a) (b)

Figure 4 T-SNE visualization of log sequences from FC(06/15/2023) using their learned LSTM-β-VAE latent representations. (a) Colored by temporality. (b) Colored by the number of known MLC-problem related messages.

such as time-series sensor measurements, to move toward fault diagnosis and prognosis using multimodal data and AI models. In this context, AEs remain central due to their key advantages: they naturally support multimodal learning, enable unsupervised or semi-supervised training, and allow for the construction of meaningful latent representations tailored to the different mentioned tasks.

3.1 Exploration

3.1.1 Multimodal learning

Multimodal learning refers to the development of AI models capable of processing and integrating heterogeneous data sources – such as images, time series, text, or tabular data – within a unified framework. This paradigm is particularly relevant in radiotherapy systems monitoring, where combining event logs, sensor streams, and contextual metadata yields richer representations of machine behavior. The goal is to extract complementary information from each modality to improve tasks such as anomaly detection, diagnosis, and prognosis. A key challenge lies in aligning and fusing modalities with differing structures, semantics, and temporal characteristics. Central to this is representation learning, which encodes raw multimodal inputs into robust, task-relevant vector representations.

Early approaches, like Multimodal Deep Boltzmann Machines [50], illustrated the potential of probabilistic graphical models to jointly model diverse modalities via a shared latent layer. However, these models are computationally intensive and require complex variational inference [50]. More scalable alternatives have emerged with AEs, now central to unsupervised and self-supervised multimodal learning. In this setting, AEs use modality-specific encoders (e.g., CNNs, RNNs or transformers) fused into a shared latent space.

AEs are well-suited for self-supervised learning, where reconstruction or auxiliary tasks drive representation learning without labels. Robinet et al. (2024) [46] introduced DRIM, which uses dual encoders per modality. The DRIM-U variant minimizes reconstruction of modality-unique components, using a tailored loss inspired by supervised contrastive learning and an adversarial objective. Contrastive learning has emerged as a powerful unsupervised approach, with methods like contrastive predictive coding [39], which use InfoNCE loss to bring similar representations closer and push others apart. Geng et al. (2022) [19] proposed Multimodal Masked AEs (M3AE), which learn joint vision-language representations via masked token prediction, avoiding modality-specific encoders and contrastive learning. Feng et al. (2024) [17] added a modality-consistency detection task, where the network learns to identify tampered modalities, enhancing cross-modal coherence.

AEs also optimize latent spaces for downstream goals. DRIM separates shared representations – rich in patient-specific information – from unique ones by minimizing mutual information, improving interpretability and predictive utility [46]. Correlational Neural Networks (CorrNet) [10] maximize cross-modal correlation in the latent space. Yang et al. extended this idea to temporal data with CorrRNN [61].

Fusion strategies in AE-based architectures vary. As surveyed by Zhao et al. (2024) [64], fusion can occur at the raw-data, feature, or decision levels. DRIM combines shared and unique representations through attention-based fusion, handling missing modalities effectively [46]. Other techniques like Tensor Fusion Networks model high-order modality interactions, though with higher computational cost [63].

Overall, AE-based multimodal learning offers a flexible framework for heterogeneous data, especially when labels are limited. Key directions include incorporating temporal dynamics, improving robustness to missing modalities, and optimizing latent spaces for real-world tasks such as anomaly detection, diagnosis, and prognosis.

3.1.2 Diagnosis

Fault diagnosis is a core task in system monitoring and maintenance, ensuring safety, reliability, and efficiency. It refers to the reasoning process used to identify the nature and root cause of a failure based on observed symptoms from measurements, checks, or tests. Formally, it can be seen as an inference problem: determining a system's internal (possibly faulty) state from its external outputs. Diagnosis typically involves three stages: fault detection (whether a fault occurred), fault isolation (identifying the faulty component), and fault identification (characterizing the fault's type and severity). Methods vary by system complexity and data availability, and can be categorized into model-based, data-driven, or hybrid approaches combining physical knowledge with statistical or machine learning techniques.

3.1.2.1 Model-based diagnosis

Model-based approaches have long been the dominant paradigm in fault diagnosis [7, 40]. These approaches rely on constructing analytical or physical models that describe the system's behavior under nominal and faulty conditions. Such models are usually derived from first-principles knowledge, including conservation laws, differential equations, discrete event models, or logical constraints, and are validated through expert analysis and simulation. Model-based methods provide high interpretability, allow detection of specific fault patterns with precision, and generally perform well in systems where accurate models are available.

However, their effectiveness is often limited by the complexity of real-world systems. Modeling nonlinear dynamics, stochastic disturbances, time-varying behaviors, and interactions between subsystems remains a challenging task. See [41] for challenges referring to the DX approaches. Furthermore, developing and validating such models requires significant domain expertise and resources, which can be prohibitive in systems that evolve rapidly or lack comprehensive documentation.

3.1.2.2 Data-driven diagnosis

Data-driven fault diagnosis has gained momentum over the last decade due to increasing sensor data availability, advances in machine learning, and enhanced computational power. Unlike model-based methods, these approaches infer patterns or fault signatures directly from historical or real-time data. This enables fault detection in complex or poorly understood

systems and improves scalability. Reviews such as [11, 47] highlight the maturity and applicability of these methods across domains like HVAC systems and general industrial equipment, underlining their potential to automate diagnostics and reduce expert reliance.

Among these techniques, AEs are widely used for their ability to learn compact, informative representations from high-dimensional data. Often, they serve as feature extractors, with supervised classifiers trained on the latent space. For instance, Han Liu et al. (2018) [28] proposed a recurrent AE-based method using gated recurrent units (GRU-NP-DAEs), where each AE is trained on a specific fault class and the classification is determined by identifying the AE that minimizes the reconstruction error. Similarly, Lang Liu et al. (2024) [29] introduced a variable-wise stacked temporal AE (VW-STAE), in which a variable sensitivity analysis guides the classification, again relying on supervised training per fault type.

Other works improved AE architectures for better feature extraction. Shao et al. (2021) [49] used adaptive Morlet wavelets to capture nonlinearities. Yang et al. (2020) [60] combined sparse and denoising AEs with ensemble learning. Qiu et al. (2025) [43] proposed a multimodal fusion scheme using multiscale stacked denoising AEs for noise robustness. Zhao et al. (2024) [65] presented a semi-supervised Gaussian mixture VAE for few-shot learning, adapting to new fault classes via episodic training and a dynamic multimodal prior.

However, these methods often assume the availability of labeled data for all fault types – an unrealistic assumption in practice. Real-world systems frequently encounter rare or unknown faults, and collecting exhaustive labeled datasets is infeasible. Fully supervised AE-based methods may thus struggle to generalize.

To address this, semi- and unsupervised AE-based methods have emerged. These models learn representations of normal data patterns without relying on fault labels. For example, Amini and Zhu (2022) [4] introduced a source-aware AE that can operate with or without labels. Cacciarelli and Kulahci (2022) [9] proposed an orthogonal AE to decorrelate latent features, improving fault detection and interpretability. Ma et al. (2018) [33] developed a deep coupling AE for multimodal sensory data, learning a shared representation and applying late fusion for diagnosis.

These studies reflect growing interest in unsupervised learning frameworks for early fault detection in safety-critical systems. While supervised AEs remain popular, semi-supervised or unsupervised models better match real-world constraints, offering scalable, realistic solutions for modern diagnostic challenges.

3.1.2.3 Fault diagnosis and AI explainability

Another promising avenue for fault diagnosis using data-driven and unsupervised learning methods is to investigate the role of explainability in AI models. Explainability refers to the extent to which a model's internal mechanisms, decisions, and outputs can be interpreted and understood by humans. In fact, the explainability of the models can be viewed as a form of diagnosis. While fault detection methods typically indicate the presence of an abnormal state, explainability goes further by identifying the elements or patterns that contributed to this state – essentially answering why the system deviated from normal behavior. In this sense, providing an explanation can lead to diagnosing the cause of the fault. Therefore, designing an explainable fault detection model is inherently aligned with building a diagnosis system. Despite its importance, this connection has received limited attention in data-driven research. In such unsupervised contexts, explainability plays a crucial role, as it enables the interpretation – and thus the diagnosis – of detected anomalies in the absence of explicit ground truth. Exploring explainability as a diagnostic tool is therefore not only a promising direction, but also a necessary one to improve the understanding, trustworthiness, and practical applicability of data-driven fault detection systems under realistic constraints.

Explainability methods are typically categorized into two families: post-hoc methods, which seek to interpret already trained models, and intrinsic methods, which embed interpretability directly into the model architecture or training process. This distinction is particularly relevant for AEs, which are widely used in unsupervised tasks such as anomaly detection but often operate as black boxes.

Post-hoc explainability techniques are applied after the model is trained, often without modifying the model's structure. Among the most widely used post-hoc tools are Shapley Additive Explanations (SHAP)[32], which attribute contributions of input features to a model's predictions. SHAP has been extensively employed in the context of AEs to understand anomalies and latent representations. For instance, Antwarg et al. (2021) [6] applied Kernel SHAP to explain reconstruction-based anomalies by linking reconstruction errors to influential input features. Similarly, Xu et al. (2021) [59] used SHAP in a dynamic multimodal VAE (DMVAE) to provide both local and global feature attribution in a clinical prediction task. In genomics, Li et al. (2023) [26] introduced XA4C, an AE-based pipeline where SHAP values derived from XGBoost identify critical genes contributing to latent representations, supporting downstream biological interpretation.

Another line of work focuses on counterfactual explanations, which generate hypothetical examples to highlight what changes would be required to alter a model's output, which shares high similarity with the concept of *conflict* known in the logical diagnosis theory [44, 13]. Using contrastive supervision, Todo et al. (2023) [52] trained a VAE to disentangle class-relevant and class-irrelevant components in multivariate time series, enabling the generation of plausible counterfactuals by manipulating only the class-relevant latent subspace. Extending this concept, Haselhoff et al. (2024) [21] proposed the Gaussian discriminant VAE (GdVAE), a self-explainable generative model that integrates a class-conditional latent space with closed-form counterfactual generation, balancing interpretability and quality of explanations in vision tasks.

Gradient-based attention mechanisms have also been applied to AEs to enhance interpretability. Liu et al. (2020) [30] derived visual attention maps from VAE latent variables to localize anomalies in images, while Nguyen et al. (2019) [38] used gradient-based fingerprinting in an unsupervised VAE for network anomaly detection.

Another line of work involves surrogate models like LIME [45], which approximate AE behavior locally using interpretable models (e.g., decision trees). Wu and Wang (2021) [57] proposed a LIME-based framework with explainers for reconstruction, classification, and global behavior in fraud detection.

In contrast to post-hoc approaches, intrinsic explainability is built into the model architecture or training objective. One strategy is to enforce interpretable representations through structured constraints. For example, Di Clemente et al. (2025) [15] developed a physics-informed AE where latent codes directly correspond to astrophysical quantities such as mass and radius of neutron stars. By embedding domain knowledge and explicit constraints in the loss function, the model achieves physical interpretability of latent variables. Other studies integrate interpretability by combining AEs with inherently transparent models. Aguilar et al. (2022) [3] proposed a decision tree-based AE capable of handling categorical data without encoding, offering interpretable internal representations through the branching structure of the tree itself. In probabilistic frameworks, Bayesian autoencoders (BAEs) can enhance interpretability via uncertainty estimation. Yong and Brintrup (2022) [62] introduced coalitional Bayesian AEs, where explanations are derived from the mean and epistemic uncertainty of log-likelihood estimates, providing insight into model behavior under covariate shift without relying on additional explainer models. Other works focused on providing explanations based on reconstruction errors from AEs. Kieu et al. (2022) [24] used Robust Principal Component

Analysis (RPCA) combined with AEs to improve the explainability of outlier detection in time series, separating outliers from clean data. Martinez-Garcia et al. (2019) [36] proposed the entropy of the AE's reconstructed outputs as a form of explanation.

3.1.3 Prognosis

In prognosis, as in diagnosis, methodologies can broadly be divided into model-based and data-driven approaches, each offering distinct strategies for predicting the Remaining Useful Life (RUL) and anticipating system failures.

Model-based methods rely on physical laws and tools like Physics of Failure (PoF), Kalman filters, and finite element analysis to estimate RUL without historical data.

Data-driven methods leverage sensor data and failure history to predict degradation. They include stochastic models (e.g., Weibull distribution, Bayesian networks, Hidden Markov Models), statistical techniques (e.g., ARMA, ARIMA), and AI-based approaches.

Among AI-based methods, similarity-based learning has gained prominence. Widodo et al. (2025) [55] demonstrated an approach for boiler prognosis using Support Vector Machines (SVMs), Random Forest Algorithms (RFAs), and Dynamic Time Warping (DTW) for RUL estimation, showing potential for real-world deployment in power plants.

Deep learning models, particularly LSTM networks, have been widely used to model temporal dependencies in degradation data. Liu et al. (2021) [31] introduced an elastic-net-regularized LSTM (E-LSTM) to mitigate overfitting and improve RUL prediction stability for rolling bearings. Wu et al. (2018) [58] utilized vanilla LSTM networks and dynamic differential technology to enhance RUL prediction under varying operational conditions and noise levels.

AE-based architectures have also been explored for prognostic tasks. Robinet et al. (2024) [46] proposed a method for survival prediction using disentangled representations from incomplete multimodal healthcare data, applying a discretized time model supervised by a specialized loss function for censored survival data as described by Kvamme and Borgan (2021) [25]. This approach models hazard probabilities over time intervals and learns individualized survival curves from multimodal inputs. De Pater and Mitici (2023) [14] designed an LSTM-AE with attention to develop health indicators for aircraft system in an unsupervised manner. These health indicators are further used to predict the RUL of the aircraft system using a similarity-based matching approach. In a more recent contribution, Tefera et al. (2025) [51] introduced a constraint-guided deep learning framework to generate physically consistent health indicators from bearing sensor data. The proposed AE model integrates domain constraints – monotonicity, bounded output, and energy-consistency – into the training process via a custom optimization scheme. Compared to baseline models, this approach enhances trendability, robustness, and consistency, yielding interpretable degradation profiles aligned with physical expectations.

Overall, the landscape of prognosis methodologies continues to evolve, with hybrid approaches combining physics-based insight and data-driven learning offering powerful solutions for anticipating failures and optimizing maintenance in complex systems.

4 Schedule

The proposed research roadmap is structured into three interconnected phases. Each phase explores a fundamental capability – fault detection, diagnosis, and prognosis – through the lens of multimodal and unsupervised (or semi-supervised) learning. A central prerequisite for each stage is the curation of a clean and well-structured dataset focused on the MLC subsystem, enabling a controlled yet realistic environment for experimentation.

4.1 Phase 1 – Multimodal Representation Learning for Anomaly Detection

Phase 1 aims to construct joint representations of operational logs and time-series sensor data through unsupervised deep learning. Inspired by recent advances such as DRIM [46] and Correlational Neural Networks [10], this step will evaluate various fusion strategies – including dual encoder architectures, disentangled shared/unique representations, and correlation-maximizing latent spaces.

The goal is to design an embedding space where normal and abnormal behaviors can be effectively separated, even in the absence of fault labels. Special attention will be given to robustness in the presence of missing modalities and asynchronous data.

Milestones:

- Construction of a labeled and time-aligned multimodal dataset focused on MLC.
- Leveraging and extending existing multimodal AE approaches (e.g., DRIM, CorrNet) to construct a latent space that better suit the characteristics of radiotherapy system data.
- Testing different self-supervised learning strategies: reconstruction [46], masked token prediction [19] and modality-consistency detection task [17].

4.2 Phase 2 – Explainable Fault Diagnosis via Latent Representations

Phase 2 focuses on leveraging the learned multimodal latent space to perform fault diagnosis in an unsupervised or weakly supervised setting. A key hypothesis is that explainability can serve as a proxy for diagnosis, especially when ground truth labels are scarce. Building on the work of Todo et al. [52] and the logical theory of conflict-based diagnosis [44, 13], counterfactual explanation techniques will be explored as a means of identifying latent dimensions or input factors contributing to anomalies.

In parallel, post-hoc tools such as SHAP will be employed to generate interpretable attributions on both the model outputs and the latent encoding. The interplay between these explanations and traditional diagnostic tasks (fault isolation, severity ranking) will be investigated.

Milestones:

- Adaptation of counterfactual explanations to the latent space of multimodal AEs.
- SHAP-based analysis of log and sensor contributions to fault signatures.
- Evaluation of the diagnostic capability of selected explainable methods.

4.3 Phase 3 – Prognosis and Remaining Useful Life Estimation

Phase 3, and the final stage, addresses long-term prediction of subsystem degradation. Two complementary directions will be explored: (1) survival analysis with multimodal latent embeddings, following Robinet et al. [46]; (2) unsupervised health indicator construction with physical constraints, following Tefera et al. [51].

In the first direction, discrete-time hazard models will be used to estimate individualized survival curves from latent variables, integrating sensor and log-derived features. In the second, attention will be paid to embedding physical priors (e.g., monotonicity, boundedness) directly into AE training, to produce interpretable and consistent degradation profiles.

Milestones:

- Derivation of latent health indicators from log and sensor embeddings.
- Modeling of hazard probabilities from multimodal data (DRIM-like survival modeling).
- Training of constraint-aware AEs to enforce physically consistent degradation behavior.

Comparison across Phases: Throughout all phases, systematic comparisons will be conducted between semi-supervised learning (enabled by partially labeled TOMO data) and fully unsupervised alternatives, to assess scalability and realism in operational settings.

5 Conclusion

This work presents a semi-supervised learning framework for fault detection in complex systems using log data. The method, based on a LSTM-β-VAE, demonstrated effective anomaly detection on both benchmark and real-world datasets, leveraging an optimized latent space combined with conformal prediction. The case study on TomoTherapy® machines highlights the practical relevance of this approach in a safety-critical healthcare setting.

Looking forward, the research will expand to incorporate time-series sensor data alongside log sequences, moving toward a multimodal diagnostic and prognostic framework. Upcoming phases will explore multimodal fusion strategies, counterfactual and SHAP-based explainability techniques for unsupervised diagnosis, and survival modeling for prognosis. These directions aim to produce interpretable, generalizable models that support fault isolation and remaining useful life estimation under realistic operational constraints.

References

1 Charles M Able, Alan H Baydush, Callistus Nguyen, Jacob Gersh, Alois Ndlovu, Igor Rebo, Jeremy Booth, Mario Perez, Benjamin Sintay, and Michael T Munley. A model for preemptive maintenance of medical linear accelerators—predictive maintenance. *Radiation Oncology*, 11:1–9, 2016.

2 Christina Elizabeth Agnew, Sergio Esteve, Glenn Whitten, and William Little. Reducing treatment machine downtime with a preventative mlc maintenance procedure. *Physica Medica*, 85:1–7, 2021.

3 Diana Laura Aguilar, Miguel Angel Medina-Pérez, Octavio Loyola-Gonzalez, Kim-Kwang Raymond Choo, and Edoardo Bucheli-Susarrey. Towards an interpretable autoencoder: A decision-tree-based autoencoder and its application in anomaly detection. *IEEE transactions on dependable and secure computing*, 20(2):1048–1059, 2022. doi:10.1109/TDSC.2022.3148331.

4 Nima Amini and Qinqin Zhu. Fault detection and diagnosis with a novel source-aware autoencoder and deep residual neural network. *Neurocomputing*, 488:618–633, 2022. doi:10.1016/J.NEUCOM.2021.11.067.

5 Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021. arXiv:2107.07511.

6 Liat Antwarg, Ronnie Mindlin Miller, Bracha Shapira, and Lior Rokach. Explaining anomalies detected by autoencoders using shapley additive explanations. *Expert systems with applications*, 186:115736, 2021. doi:10.1016/J.ESWA.2021.115736.

7 Pietro Baroni, Gianfranco Lamperti, Paolo Pogliano, and Marina Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110(1):135–183, 1999. doi:10.1016/S0004-3702(99)00019-3.

8 Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000. doi:10.1145/342009.335388.

9 Davide Cacciarelli and Murat Kulahci. A novel fault detection and diagnosis approach based on orthogonal autoencoders. *Computers & Chemical Engineering*, 163:107853, 2022. doi:10.1016/J.COMPCHEMENG.2022.107853.

10 Sarath Chandar, Mitesh M Khapra, Hugo Larochelle, and Balaraman Ravindran. Correlational neural networks. *Neural computation*, 28(2):257–285, 2016. doi:10.1162/NECO_A_00801.

11 Zhelun Chen, Zheng O'Neill, Jin Wen, Ojas Pradhan, Tao Yang, Xing Lu, Guanjing Lin, Shohei Miyata, Seungjae Lee, Chou Shen, et al. A review of data-driven fault detection and diagnostics for building hvac systems. *Applied Energy*, 339:121030, 2023.

12 Stéphan Clémençon and Jérémie Jakubowicz. Scoring anomalies: a m-estimation formulation. In *Artificial Intelligence and Statistics*, pages 659–667. PMLR, 2013. URL: http://proceedings.mlr.press/v31/clemencon13a.html.

13 Johan De Kleer and Brian C Williams. Diagnosing multiple faults. *Artificial intelligence*, 32(1):97–130, 1987. doi:10.1016/0004-3702(87)90063-4.

14 Ingeborg de Pater and Mihaela Mitici. Developing health indicators and rul prognostics for systems with few failure instances and varying operating conditions using a lstm autoencoder. *Engineering Applications of Artificial Intelligence*, 117:105582, 2023. doi:10.1016/J.ENGAPPAI.2022.105582.

15 Francesco Di Clemente, Matteo Scialpi, and Michał Bejger. Explainable autoencoder for neutron star dense matter parameter estimation. *Machine Learning: Science and Technology*, 2025.

16 Tai Dou, Benjamin Clasie, Nicolas Depauw, Tim Shen, Robert Brett, Hsiao-Ming Lu, Jacob B Flanz, and Kyung-Wook Jee. A deep lstm autoencoder-based framework for predictive maintenance of a proton radiotherapy delivery system. *Artificial Intelligence in Medicine*, 132:102387, 2022. doi:10.1016/J.ARTMED.2022.102387.

17 Wenjun Feng, Xin Wang, Donglin Cao, and Dazhen Lin. An autoencoder-based self-supervised learning for multimodal sentiment analysis. *Information Sciences*, 675:120682, 2024. doi:10.1016/J.INS.2024.120682.

18 José A González Ferreira, Javier Jaén Olasolo, Ignacio Azinovic, and Branislav Jeremic. Effect of radiotherapy delay in overall treatment time on local control and survival in head and neck cancer: review of the literature. *Reports of Practical Oncology and Radiotherapy*, 20(5):328–339, 2015.

19 Xinyang Geng, Hao Liu, Lisa Lee, Dale Schuurmans, Sergey Levine, and Pieter Abbeel. Multimodal masked autoencoders learn transferable representations. *arXiv preprint arXiv:2205.14204*, 2022. doi:10.48550/arXiv.2205.14204.

20 Nicolas Goix. How to evaluate the quality of unsupervised anomaly detection algorithms? *arXiv preprint arXiv:1607.01152*, 2016. arXiv:1607.01152.

21 Anselm Haselhoff, Kevin Trelenberg, Fabian Küppers, and Jonas Schneider. The gaussian discriminant variational autoencoder (gdvae): A self-explainable model with counterfactual explanations. In *European Conference on Computer Vision*, pages 305–322. Springer, 2024. doi:10.1007/978-3-031-73668-1_18.

22 Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE, 2017. doi:10.1109/ICWS.2017.13.

23 Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2017.

24 Tung Kieu, Bin Yang, Chenjuan Guo, Christian S Jensen, Yan Zhao, Feiteng Huang, and Kai Zheng. Robust and explainable autoencoders for unsupervised time series outlier detection. In *2022 IEEE 38th International conference on data engineering (ICDE)*, pages 3038–3050. IEEE, 2022. doi:10.1109/ICDE53745.2022.00273.

25 Håvard Kvamme and Ørnulf Borgan. Continuous and discrete-time survival prediction with neural networks. *Lifetime data analysis*, 27(4):710–736, 2021.

26 Qing Li, Yang Yu, Pathum Kossinna, Theodore Lun, Wenyuan Liao, and Qingrun Zhang. Xa4c: explainable representation learning via autoencoders revealing critical genes. *PLOS Computational Biology*, 19(10):e1011476, 2023. doi:10.1371/JOURNAL.PCBI.1011476.

27 Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008. doi:10.1109/ICDM.2008.17.

28 Han Liu, Jianzhong Zhou, Yang Zheng, Wei Jiang, and Yuncheng Zhang. Fault diagnosis of rolling bearings with recurrent neural network-based autoencoders. *ISA transactions*, 77:167–178, 2018.

29 Lang Liu, Ying Zheng, and Shaojun Liang. Variable-wise stacked temporal autoencoder for intelligent fault diagnosis of industrial systems. *IEEE Transactions on Industrial Informatics*, 2024.

30 Wenqian Liu, Runze Li, Meng Zheng, Srikrishna Karanam, Ziyan Wu, Bir Bhanu, Richard J Radke, and Octavia Camps. Towards visually explaining variational autoencoders. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8642–8651, 2020.

31 Zhao-Hua Liu, Xu-Dong Meng, Hua-Liang Wei, Liang Chen, Bi-Liang Lu, Zhen-Heng Wang, and Lei Chen. A regularized lstm method for predicting remaining useful life of rolling bearings. *International Journal of Automation and Computing*, 18:581–593, 2021. doi:10.1007/S11633-020-1276-6.

32 Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

33 Meng Ma, Chuang Sun, and Xuefeng Chen. Deep coupling autoencoder for fault diagnosis with multimodal sensory data. *IEEE Transactions on Industrial Informatics*, 14(3):1137–1145, 2018. doi:10.1109/TII.2018.2793246.

34 Min Ma, Chenbin Liu, Ran Wei, Bin Liang, and Jianrong Dai. Predicting machine's performance record using the stacked long short-term memory (lstm) neural networks. *Journal of Applied Clinical Medical Physics*, 23(3):e13558, 2022.

35 T Rockwell Mackie, John Balog, Ken Ruchala, Dave Shepard, Stacy Aldridge, Ed Fitchard, Paul Reckwerdt, Gustavo Olivera, Todd McNutt, and Minesh Mehta. Tomotherapy. In *Seminars in Radiation Oncology*, volume 9, pages 108–117. Elsevier, 1999.

36 Miguel Martinez-Garcia, Yu Zhang, Jiafu Wan, and Jason Mcginty. Visually interpretable profile extraction with an autoencoder for health monitoring of industrial systems. In *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 649–654. IEEE, 2019.

37 Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

38 Quoc Phong Nguyen, Kar Wai Lim, Dinil Mon Divakaran, Kian Hsiang Low, and Mun Choon Chan. Gee: A gradient-based explainable variational autoencoder for network anomaly detection. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 91–99. IEEE, 2019. doi:10.1109/CNS.2019.8802833.

39 Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

40 Bernhard Peischl and Franz Wotawa. Model-based diagnosis or reasoning from first principles. *IEEE intelligent systems*, 18(3):32–37, 2005.

41 Ingo Pill and Johan De Kleer. Challenges for model-based diagnosis. In *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*, pages 6–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/OASIcs.DX.2024.6.

42 Kélian Poujade, Louise Travé-Massuyès, Jérémy Pirard, and Laure Vieillevigne. B-variational autoencoder based anomaly detection in log data – application to radiotherapy systems. Preprint, HAL archive, 2025. URL: `https://hal.science/hal-05209127`.

43 Zhi Qiu, Shanfei Fan, Haibo Liang, and Jincai Liu. Multimodal fusion fault diagnosis method under noise interference. *Applied Acoustics*, 228:110301, 2025.

44 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. `doi:10.1016/0004-3702(87)90062-2`.

45 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

46 Lucas Robinet, Ahmad Berjaoui, Ziad Kheil, and Elizabeth Cohen-Jonathan Moyal. Drim: Learning disentangled representations from incomplete multimodal healthcare data. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 163–173. Springer, 2024. `doi:10.1007/978-3-031-72384-1_16`.

47 Atma Ram Sahu, Sanjay Kumar Palei, and Aishwarya Mishra. Data-driven fault diagnosis approaches for industrial equipment: A review. *Expert Systems*, 41(2):e13360, 2024. `doi:10.1111/EXSY.13360`.

48 Bernhard Schölkopf, Robert C Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. *Advances in neural information processing systems*, 12, 1999.

49 Haidong Shao, Min Xia, Jiafu Wan, and Clarence W de Silva. Modified stacked autoencoder using adaptive morlet wavelet for intelligent fault diagnosis of rotating machinery. *IEEE/ASME Transactions on Mechatronics*, 27(1):24–33, 2021.

50 Nitish Srivastava and Russ R Salakhutdinov. Multimodal learning with deep boltzmann machines. *Advances in neural information processing systems*, 25, 2012.

51 Yonas Tefera, Quinten Van Baelen, Maarten Meire, Stijn Luca, and Peter Karsmakers. Constraint-guided learning of data-driven health indicator models: An application on the pronostia bearing dataset. *arXiv preprint arXiv:2503.09113*, 2025. `doi:10.48550/arXiv.2503.09113`.

52 William Todo, Merwann Selmani, Béatrice Laurent, and Jean-Michel Loubes. Counterfactual explanation for multivariate times series using a contrastive variational autoencoder. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023. `doi:10.1109/ICASSP49357.2023.10095789`.

53 Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

54 David C Westerly, Emilie Soisson, Quan Chen, Katherine Woch, Leah Schubert, Gustavo Olivera, and Thomas R Mackie. Treatment planning to improve delivery accuracy and patient throughput in helical tomotherapy. *International Journal of Radiation Oncology* Biology* Physics*, 74(4):1290–1297, 2009.

55 Achmad Widodo, Toni Prahasto, Mochamad Soleh, and Herry Nugraha. Diagnostics and prognostics of boilers in power plant based on data-driven and machine learning. *International Journal of Prognostics and Health Management*, 16(1), 2025.

56 Binbin Wu, Pengpeng Zhang, Bill Tsirakis, David Kanchaveli, and Thomas LoSasso. Utilizing historical mlc performance data from trajectory logs and service reports to establish a proactive maintenance model for minimizing treatment disruptions. *Medical physics*, 46(2):475–483, 2019.

57 Tung-Yu Wu and You-Ting Wang. Locally interpretable one-class anomaly detection for credit card fraud detection. In *2021 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 25–30. IEEE, 2021.

58 Yuting Wu, Mei Yuan, Shaopeng Dong, Li Lin, and Yingqi Liu. Remaining useful life estimation of engineered systems using vanilla lstm neural networks. *Neurocomputing*, 275:167–179, 2018. `doi:10.1016/J.NEUCOM.2017.05.063`.

59 Yiming Xu, Xiaohong Liu, Liyan Pan, Xiaojian Mao, Huiying Liang, Guangyu Wang, and Ting Chen. Explainable dynamic multimodal variational autoencoder for the prediction of patients with suspected central precocious puberty. *IEEE Journal of Biomedical and Health Informatics*, 26(3):1362–1373, 2021. doi:10.1109/JBHI.2021.3103271.

60 Jing Yang, Guo Xie, and Yanxi Yang. An improved ensemble fusion autoencoder model for fault diagnosis from imbalanced and incomplete data. *Control Engineering Practice*, 98:104358, 2020.

61 Xitong Yang, Palghat Ramesh, Radha Chitta, Sriganesh Madhvanath, Edgar A Bernal, and Jiebo Luo. Deep multimodal representation learning from temporal data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5447–5455, 2017.

62 Bang Xiang Yong and Alexandra Brintrup. Coalitional bayesian autoencoders: Towards explainable unsupervised deep learning with applications to condition monitoring under covariate shift. *Applied Soft Computing*, 123:108912, 2022. doi:10.1016/J.ASOC.2022.108912.

63 Amir Zadeh, Minghai Chen, Soujanya Poria, Erik Cambria, and Louis-Philippe Morency. Tensor fusion network for multimodal sentiment analysis. *arXiv preprint arXiv:1707.07250*, 2017. arXiv:1707.07250.

64 Fei Zhao, Chengcui Zhang, and Baocheng Geng. Deep multimodal data fusion. *ACM computing surveys*, 56(9):1–36, 2024. doi:10.1145/3649447.

65 Zhiqian Zhao, Yeyin Xu, Jiabin Zhang, Runchao Zhao, Zhaobo Chen, and Yinghou Jiao. A semi-supervised gaussian mixture variational autoencoder method for few-shot fine-grained fault diagnosis. *Neural Networks*, 178:106482, 2024. doi:10.1016/J.NEUNET.2024.106482.

66 Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R Lyu. Loghub: A large collection of system log datasets for ai-driven log analytics. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 355–366. IEEE, 2023. doi:10.1109/ISSRE59848.2023.00071.