Property Learning-Based Fault Detection for Liquid Propellant Rocket Engine Control Systems

Andrea Urgolo[1]
Silicon Austria Labs GmbH (SAL), Graz, Austria

Ingo Pill
Institute of Software Technology, Graz University of Technology, Austria

Günther Waxenegger-Wilfing
University of Würzburg, Germany
German Aerospace Center (DLR), Lampoldshausen, Germany

Manuel Freiberger
Silicon Austria Labs GmbH (SAL), Graz, Austria

Abstract

Accommodating the dynamic and uncertain operational environments that are typical for aerospace applications, our work focuses on robust fault detection and accurate diagnosis in the context of Liquid Propellant Rocket Engines. To this end, we employ techniques based on learning temporal properties which are then dynamically adapted and refined based on observed behavior. Leveraging the capabilities of genetic programming, our methodology evolves and optimizes temporal properties that are validated through formal methods in order to ensure precise, interpretable real-time fault monitoring and diagnosis. Our integrated strategy enables us to enhance resilience, safety and reliability when operating rocket engines – due to the proactive detection and systematic analysis of operational deviations before they would escalate into critical failures. We demonstrate the effectiveness of our method via a rigorous evaluation across varied simulated fault conditions, in order to showcase its potential to significantly mitigate the fault-related risks in aerospace systems.

2012 ACM Subject Classification Computing methodologies → Machine learning; Computer systems organization → Embedded and cyber-physical systems

Keywords and phrases Machine learning, Runtime verification, Property learning, Monitoring, Fault detection, Diagnosis, Genetic programming, Explainable AI

Digital Object Identifier 10.4230/OASIcs.DX.2024.15

Funding This project has been financially supported by the Austrian Research Promotion Agency (FFG) under grant no. 897989.

1 Introduction

With the visionary goal to become the first climate-neutral continent by 2050 [17], the European Union claims a global leadership role in the world-wide sustainability efforts. In the European aerospace sector, we still see a strong need for new technology that allows us to contribute to implementing this ambitious agenda. This need naturally fosters the advent of reusable rocket technology, such as to minimize the ecological footprint and maximize resource utilization. SpaceX's Falcon 9 [32] exemplifies the feasibility of reusable launch vehicles even today and illustrates a tangible path towards a more sustainable space sector.

The deployment of reusable rocket engines requires precise and dynamic control algorithms for Liquid Propellant Rocket Engines (LPREs) that need to be capable of compensating large and systematic deviations from expected behavior. In particular, intelligent control

[1] Corresponding author


© Andrea Urgolo, Ingo Pill, Günther Waxenegger-Wilfing, and Manuel Freiberger;
licensed under Creative Commons License CC-BY 4.0

approaches must (a) choose the operating point considering component wear (life extension), (b) robustly address internal and external disturbances, and (c) enable a seamless transition between alternate operation modes – during different flight phases, but also to mitigate issues.

Such intelligent control stands in contrast to classic rocket control designs though. Those rely mostly on open-loop control or, at best, steady-state controllers (see [31] for an overview) – requiring us to evolve our current concepts. Achieving intelligent control in a resource-limited LPRE scenario naturally entails the integration of resource-optimal, quick, and resilient monitoring and diagnosis systems. These systems must provide critical data by tracking component wear, detecting anomalies, and identifying system faults that are paramount for a comprehensive understanding of the current state of the entire system. This awareness is essential for enabling effective closed-loop control with fault management capabilities.

In this paper, we focus on monitoring LPREs and investigate the applicability of learning monitorable properties for fault detection. To that end, we develop and evaluate a property learning-based approach for fault detection and isolation, leveraging temporal logic and genetic programming techniques. Our contributions are as follows:

- a multi-objective genetic programming approach for automatically learning fault detection/isolation properties in Signal Temporal Logic (STL),
- its application to monitoring and diagnosing LPREs, enabling the detection and isolation of various fault types, including sensor drifts, offsets, and mechanical failures,
- a method for exploiting virtual sensing in the fault detection/isolation process,
- a comprehensive evaluation of our technology.

We structured our presentation as follows: We start with an introduction of preliminaries in Sec. 2. After discussing our approach and contributions in Sec. 3, we introduce the LPRE control application (domain) in Sec. 4, in which we furthermore discuss our experiments and evaluate the effectiveness of our methodology. A detailed analysis of the strengths and weaknesses is provided in Sec. 5, followed by a discussion of related work in Sec. 6. Finally, we summarize our main findings as well as outline future research directions in Sec. 7.

2 Preliminaries

In this section, let us introduce essential concepts and our notation. Covered topics include the Signal Temporal Logic and the basics of monitoring as well as evolutionary algorithms.

2.1 The Signal Temporal Logic STL

Similar to the well-known Linear Temporal Logic (LTL) [30], Signal Temporal Logic (STL) [23] enhances propositional logic with temporal modalities that allow us to reason about the future. Unlike LTL, which requires the discretization of signals and time, STL allows direct reference to continuous-valued signals and continuous real-time in formulae, making it particularly interesting for real-time monitoring and analyzing signals from cyber-physical systems (CPS). By leveraging STL, we can define and monitor nuanced temporal properties that are critical for assessing a system's behavior, ensuring that any deviation from the expected performance is detected promptly. This is essential for maintaining the safety and reliability of rocket engines in dynamic conditions, where even minor anomalies can have significant consequences.

Before defining a trace as aggregation of observed signals, we briefly introduce parts of our notation. Let $\top$ and $\bot$ denote the constants *True* and *False*, and let $\neg$ denote negation. A signal τ defines for each point in time $0 \leq t \leq T(\tau)$ a real value, where the temporal

horizon $T(\tau)$ of signal τ may be finite (so that the execution ends at $T(\tau)$), or infinite. For continuous time, some signal τ can thus be described by a function $f_\tau : \mathbb{R}_0^+ \to \mathbb{R}$ that maps a (non-negative) point in time to some (real) signal value. While STL can operate in a continuous domain, practical implementations often discretize time or restrict the signal values to rational numbers for computational reasons.

In our work, we will consider (clocked) time series data with a finite temporal horizon – which we will also refer to as a *finite trace*. Now, let us furthermore assume that we observe for each point in time all the signal values in a single vector π, such that we refer with (a) Π to the entire trace (so to say the sequence of vectors for the individual time steps), (b) $\pi = \Pi(t)$ to the signal vector at time t, (c) Π_i to the trace of signal i, (d) $\Pi_i(t)$ to the value of signal i at time step t, and (e) $\Pi[j, k]$ to a partial trace that is restricted to the time interval $[j, k]$. For a finite trace, we can thus define signals as finite sequences of signal values (one per $T(\Pi) + 1$ time steps), or as a function mapping each element in the sequence $[0, \ldots, T(\tau)]$ to a value in $\mathbb{R}$. Hereafter, for convenience, the length of a signal τ will be denoted by $len(\tau)$, equal to $T(\tau) + 1$. For an infinite temporal horizon and clocked time, we can define signals and traces either as functions mapping from $\mathbb{N}$ to $\mathbb{R}$ or $\mathbb{R}^{|\tau|}$ respectively, or as finite sequences with a finite stem and an infinite loop (k,l loops) like in model-checking [12]. With this notation, let us now define the syntax and semantics of STL:

▶ **Definition 1.** *Let $\mu = f(\Pi(t)) > c$ be an atomic predicate that compares a function over the signal values for some time step t to a real-valued constant c. With φ and ψ being* STL *formulae, and I being one of the intervals (a, b), $(a, b]$, $[a, b)$, or $[a, b]$ for two points in time such that $a \leq b$, we can define the syntax of* STL *inductively as*

$$\varphi ::= \top \mid \mu \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \,\mathsf{U}_I\, \psi$$

▶ **Definition 2.** *Let I_t be the interval of an until operator applied at time step t. The satisfaction of an* STL *formula φ as of Def. 1 by a trace Π at time t is then defined as follows*

$$\begin{aligned}
&\Pi(t) \models \mu && \textit{iff } f(\Pi(t)) \geq c \\
&\Pi(t) \models \neg\varphi && \textit{iff } \neg(\Pi(t) \models \varphi) \\
&\Pi(t) \models \varphi \wedge \psi && \textit{iff } \Pi(t) \models \varphi \wedge \Pi(t) \models \psi \\
&\Pi(t) \models \varphi \,\mathsf{U}_I\, \psi && \textit{iff } \exists t' \in I_t \textit{ s.t. } \Pi(t') \models \psi \wedge \forall t'' \in [t, t'] : \Pi(t'') \models \varphi
\end{aligned}$$

As with LTL, we can introduce syntactic sugar that does not increase STL's expressiveness but improves its usability and intuitiveness. Prominent examples include the *or* $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, *exclusive or* $\varphi \oplus \psi = \varphi \wedge \neg\psi \vee \neg\varphi \wedge \psi$, and *implication* $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$ on the logic level, and temporal operators like *finally* a.k.a. *eventually* for denoting $F_I\, \varphi = \top \,\mathsf{U}_I\, \varphi$, or *globally* a.k.a. *always* for denoting $G_I\, \varphi = \neg(F_I\, \neg\varphi)$.

Let us stress that for our definitions we adopt a simplified view of having only a scalar value per signal. It is easy to see that we can easily accommodate also multiple values for one signal (this does not change τ), or modify the definitions such that signals can take values other than reals numbers (affecting only μ); e.g., a 64-bit data signal interpreted at the logic or voltage level, can be split into individual signals for each data line. Please note that Def. 2 offers very simple STL semantics, with alternative ones discussed in the following subsection.

2.1.1 Monitoring STL: Bounded STL Formulae

Traditional offline verification methods, such as model checking or testing, enable comprehensive analysis of a system's behavior. Model checking, for instance, allows us to prove or falsify properties expressed in some temporal logic for a system's entire behavior. Testing involves

stimulating a system with specific inputs (defined in test cases) and observing the resulting behavior. In either case, we investigate complete executions or runs, i.e., the observable part coined as *trace*. However, when monitoring a system at run-time, we only have a finite prefix of an execution available. In contrast to implementations of a test oracle [29], we thus have to implicitly or explicitly consider the entire set of executions that extend this prefix.

Consequently, monitoring not only entails different semantics of expressed properties [5, 29], but also presents the challenge that not all properties are indeed monitorable. A property is *positively monitorable* if any satisfying trace can *conclusively* demonstrate this through a finite prefix, while it is *negatively monitorable* if any violating trace provides such evidence within a finite prefix. Safety properties, for instance (see [3] for a discussion of safety and liveness), are negatively monitorable because any trace showing a bad event violates the property, meaning that such events should never occur. Conversely, co-safety properties are positively monitorable since a finite trace can affirm a positive event within its scope.

As for LTL [5], there are alternative semantics that are of special interest for monitoring purposes, like the following quantitative one taken from [14]. The idea is that atomic predicates do not evaluate to $\top$ or $\bot$, but to a real value, i.e., $f(\Pi(t)) - c$ from μ in Def. 1, propagated along the entire formula. These quantitative semantics provide a robustness measure ρ, which indicates the degree to which a signal satisfies a formula at a given time.

▶ **Definition 3.** *The robustness ρ of an* STL *formula φ for a trace Π at time t is defined as:*

$$
\begin{aligned}
\rho(\top, \Pi, t) &= +\infty \\
\rho(\Pi_i \geq c, \Pi, t) &= \Pi_i(t) - c \\
\rho(\neg\varphi, \Pi, t) &= -\rho(\varphi, \Pi, t) \\
\rho(\varphi_1 \wedge \varphi_2, \Pi, t) &= \min\{\rho(\varphi_1, \Pi, t), \rho(\varphi_2, \Pi, t)\} \\
\rho(\varphi_1 \mathsf{U}_I \varphi_2, \Pi, t) &= \max_{t_1 \in t+I} \min\{\rho(\varphi_2, \Pi, t_1), \min_{t_2 \in [t, t_1)} \rho(\varphi_1, \Pi, t_2)\}
\end{aligned}
$$

With the latter concept outlined, we can now formalize the definition of STL monitoring.

▶ **Definition 4.** *Let $\mathcal{X}$ (resp., $\bar{\mathcal{X}}$) be the set of infinite signals (resp., finite signals), and $\mathcal{C}(\Pi)$, the set of completions of a given finite signal $\Pi \in \bar{\mathcal{X}}$ over $\{0, \ldots, t\} \subset \mathbb{N}$, defined as $\mathcal{C}(\Pi) = \{\hat{\Pi} \in \mathcal{X} \mid \hat{\Pi}(t') = \Pi(t')$ for all $t' \in \mathbb{N}, 0 \leq t' \leq t\}$.* STL *monitoring is defined by the function $mon : \bar{\mathcal{X}} \times \mathsf{STL} \to \top, \bot, ?$ s.t. $mon(\Pi, \varphi)$ returns $\top$ iff $\hat{\Pi}, 0 \models \varphi$ for all $\hat{\Pi} \in \mathcal{C}(\Pi)$, $\bot$ iff $\hat{\Pi}, 0 \not\models \varphi$ for all $\hat{\Pi} \in \mathcal{C}(\Pi)$, equivalent to $\rho(\varphi, \hat{\Pi}, 0) < 0$ for all $\hat{\Pi} \in \mathcal{C}(\Pi)$, or ? otherwise.*

Currently, there are no tools that support the monitoring of unbounded STL formulas as defined in our work. However, bounded-time STL (bSTL), a specific fragment of STL, supports monitoring in practical applications [26]. This is achieved by restricting the temporal modality time interval I in bSTL to finite boundaries, specifically to $I = [a, b]$, where $a, b \in \mathbb{N}$ are both finite. bSTL formulas are characterized by a temporal *horizon* $H(\phi)$, which quantifies the maximum future time span necessary to evaluate the truth of the formula. For instance, the formula $\phi = x \geq 3\ \mathsf{U}_{[0,3]}\ x \geq 5$ has a horizon of 3, indicating that its validation requires data spanning at least three time units from the initial point of evaluation.

▶ **Definition 5.** *Formally, the* bSTL *formula's temporal horizon $H : \mathsf{bSTL} \to \mathbb{N}$ is defined as: (i) $H(\top) = 0$; (ii) $H(\Pi_i \geq c) = 0$; (iii) $H(\neg\varphi) = H(\phi)$; (iv) $H(\varphi_1 \wedge \varphi_2) = \max\{H(\varphi_1), H(\varphi_2)\}$; (v) $H(\varphi_1\ \mathsf{U}_{[a,b]}\ \varphi_2) = b + \max\{H(\varphi_1) - 1, H(\varphi_2)\}$.*

Furthermore, to address real-world monitoring scenarios, which require responsive system behavior analysis where conditions may need to be evaluated at any given point across a signal, we extend monitoring capabilities to apply to any suffix beyond the horizon.

▶ **Definition 6.** bSTL *monitoring operates through the function eb-mon* : $\bar{\mathcal{X}} \times \mathsf{bSTL} \to \{\top, \bot, ?\}$*, which relies on the function mon from Def. 4, and is defined as:*

$$eb\text{-}mon(\Pi, \varphi) = \begin{cases} ? & \text{if } len(\Pi) < H(\varphi) + 1 \\ \bigvee_{0 \le i \le len(\Pi)-1-H(\varphi)} mon(\Pi[i, len(\Pi) - 1], \varphi) & \text{otherwise} \end{cases}$$

This formulation ensures that a monitor can definitively issue truth values $\top$ or $\bot$, or an undefined verdict ? if the horizon has not yet been reached, thus safeguarding against premature evaluations. The effectiveness of this extended monitoring is captured by the tool `rtamt` [26], which supports practical implementation of bSTL monitoring for both discrete and dense-time properties, facilitating robust and reliable system evaluations.

Fault detection in system monitoring typically involves identifying anomalies or operational deviations that could lead to system failures. In the following, the concept of fault detection grounded on the definitions provided in the existing literature on system verifiability and monitorability [1, 21], is formalized as it is applied within the scope of our study.

▶ **Definition 7.** *Given a set $\mathcal{P}$ of properties formulated in* bSTL*, a fault is detected on an execution trace $\Pi \in \bar{\mathcal{X}}$ at a time step $i \in \mathbb{N}$ if there exists at least one fault property $\psi \in \mathcal{P}$ such that the monitoring function eb-mon$(\Pi[0, i], \psi)$ returns $\top$. Here, $\bar{\mathcal{X}}$ represents the set of all considered (finite) engine run traces, and $\Pi[0, i]$ denotes the prefix of the trace Π from the start up to the i-th time step.*

This formalization captures the essence of real-time fault detection by leveraging the temporal properties specified in bSTL, allowing for the proactive identification and subsequent handling of system anomalies before they escalate into more severe problems.

2.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are population-based metaheuristics inspired by natural evolution, effectively addressing combinatorial optimization challenges by using historical data to guide search efforts towards promising areas [16]. These algorithms involve a population of individuals, each representing a solution to an optimization problem, evaluated through a fitness function based on adaptation criteria. This fitness may be assessed via single- or multi-objective approaches, depending on the problem's complexity. The evolutionary cycle advances through generations, with a selection mechanism favoring higher fitness individuals for reproduction. Genetic operations such as crossover and mutation facilitate diversity and prevent premature convergence by introducing variations. The process iterates until meeting predefined criteria such as optimal fitness or maximum generation count.

This research specifically utilizes genetic programming (GP), a branch of EAs where individuals are represented by means of computational trees evolving towards optimal solutions over successive generations through tree-specific genetic operations. Additionally, this study leverages multi-objective EAs to optimize multiple objectives simultaneously, generating diverse Pareto-optimal solutions. This approach enhances solution exploration, allows customization of the evolutionary process, supporting a detailed investigation into optimal configurations for the extraction of complex properties in diverse scenarios. The algorithm used in this study, as previously defined in [7], is implemented through DEAP (Distributed Evolutionary Algorithms in Python), a versatile framework for rapid prototyping of EAs [18]. Operationally, two sets containing nominal and anomalous finite traces are processed, and a bSTL formula is subsequently derived to distinguish between these two conditions. Below, we present the main implementation characteristics of this algorithm.

Population and its Initialization. The population within our evolutionary algorithm framework consists of individuals, each represented as a computation tree φ encoding a syntactically correct bSTL formula. The population is generated using the *genHalfAndHalf* method from DEAP, which adheres to guidelines established by Koza [20]. This method ensures diversity in the tree structures by alternating between generating trees with uniform leaf depths and trees where leaves may vary in depth.

Nodes of the Computation Tree. In the evolutionary framework, each node within a computation tree can represent various structural and functional components of a bSTL formula. Nodes primarily encapsulate constraints such as $\Pi_i \geq c$, where Π_i is a signal identifier and c is a constant. They can also represent complex bSTL formulas where the outermost operator is temporal, for example, $\varphi \mathsf{U}_{[a,b]} \psi$, or Boolean expressions like $\varphi \vee \psi$. Here, φ and ψ themselves are bSTL sub-formulas, recursively structured as trees within the main computation tree. Moreover, nodes may embody terminal values crucial for the construction and evaluation of formulas such as: (*i*) interval bounds of temporal operators, expressed as $I = [a, b]$, where $a, b \in \mathbb{N}$ and $a \leq b$; (*ii*) signal identifiers Π_i, with $1 \leq i \leq |\Pi|$, linking the formula directly to the specific components of a multi-dimensional signal; (*iii*) constants c, used within constraints, where each c is chosen from the domain $Dom(\Pi_i)$ of the signal component Π_i. To facilitate the dynamic and ephemeral representation of values such as constants and interval bounds, DEAP employs *EphemeralConstants*. These are particularly useful for encoding values that are not fixed but need to be generated anew for each instantiation of the tree, thus supporting a rich diversity in the evolutionary process.

Crossover. The crossover operation is a key genetic mechanism in evolutionary algorithms, used to generate new individuals from existing parent solutions. In our implementation, this process involves a one-point crossover technique facilitated by DEAP's *cxOnePoint* function. This function selects a random node within the computation tree of each parent and exchanges the subtrees rooted at these nodes, thereby producing two offspring with combined genetic traits from both parents. To manage the complexity of the resulting computation trees and prevent bloat – a growth in program size without fitness improvement – a static height limit of 17 is imposed on the offspring's computation trees (DEAP's *staticLimit*), once again following Koza's guideline [20]. If this limit is exceeded, the child is discarded and replaced by a randomly selected parent. Similarly, the crossover process might occasionally produce invalid individuals, especially in terms of computation tree's horizon. Specifically, if an offspring's formula φ results in a horizon $H(\varphi)$ that exceeds the permissible range (i.e., $H(\varphi) \geq len(\Pi)$), this offspring is deemed non-viable and replaced by a randomly chosen parent. This strategy ensures that only viable solutions with valid and effective genetic configurations persist in the population for further evolutionary processing.

Mutation. Mutation in evolutionary algorithms introduces variability and helps explore new genetic landscapes. In our implementation, two mutation operators are employed, each selected with equal probability to modify individuals within the population. The first operator, *mutNodeReplacement*, targets a randomly chosen node in the computation tree of an individual and replaces it with another node that maintains the tree's syntactical correctness. The second operator, *mutEphemeral*, is designed to alter the value of a single constant in the individual's tree. Consistent with our approach to manage program complexity and avoid bloat, the same *staticLimit* is applied to the height of the trees after mutation, mirroring the constraints imposed during the crossover process. Post-mutation, it is crucial to verify

the validity of the resulting individual. If the individual's horizon or height does not align properly with the problem's constraints, the mutation is considered unsuccessful, and the original pre-mutation individual is retained in the population, hence preserving the overall quality and relevance of the evolving population.

Selection. To foster diversity within the population, our approach utilizes the elitist selection strategy as outlined in NSGA-III [13]. This method incorporates *reference points* and *niche preservation* principles, ensuring a diverse set of solutions by adequately covering all areas of the Pareto front. For a detailed explanation of these concepts, readers are directed to the foundational work by Deb et al. [13]. Finally, before mutation and crossover, elite individuals from the previous generation are selected to form the elite set. This set is then added to the next population, ensuring the persistence of high-performing individuals and their contribution across the evolutionary process.

Termination Criteria and Extraction of Final Solutions. The termination criteria for our evolutionary algorithm are twofold. First, an upper limit is set on the number of generations, a standard practice in evolutionary computations to ensure the process concludes in a timely manner. Second, we employ an *early stopping* strategy based on the *hypervolume* indicator. This measure quantifies the volume covered by the Pareto front relative to a reference point and gauges the diversity and quality of the solutions [8]. Termination occurs if there is no improvement in the hypervolume over a predetermined number of generations. Once the evolutionary process is completed, the final solutions are extracted by analyzing the Pareto front. Initially, we filter this front to retain only those individuals whose formula φ achieves an accuracy over 0.5, thereby outperforming a random classifier. Among these viable solutions, the individual contributing most significantly to the hypervolume – indicative of its excellence in diversity and fitness – is selected for output.

Algorithm Hyperparameters. The configuration of hyperparameters for the evolutionary algorithm (EA) utilized in this study was methodically determined through a grid search approach, conducted on a synthetic dataset of binary labeled bSTL traces. These traces were distinctively marked by a diverse assortment of formulas, representing two separate classes. The optimal hyperparameters were selected as follows: *population size* was set at 100, after evaluating a range from 50 to 1000; *crossover probability* was established at 0.7, with tested values ranging from 0.5 to 0.8; *mutation probability* was dynamically set to $0.5/\sqrt[2]{num\_gen}$, encouraging robust initial exploration of the search space, which tapers off to enhance exploitation of promising solutions as generations progress; for this hyperparameter values between 0.3 and 0.6 were tested; *maximum generations* was conservatively capped at 500 to ensure computational feasibility while allowing ample evolution within the population; *hypervolume early stopping* was activated after 25 generations without improvement, with the threshold tested at intervals from 10 to 50 generations; *elite rate* was set at 0.05, determining the size of the elite set as 0.05 times the *population size*. This parameter governs the proportion of the population considered elite, influencing the selection process by favoring individuals associated to a higher hypervolume for reproduction. Additionally, a critical hyperparameter specific to this EA implementation, which leverages bSTL and `rtamt`, is the *max horizon*. This parameter intuitively sets an upper boundary h on the horizon of the formulas explored by the EA, leading to the following implications: formulas are restricted to capturing phenomena that extend no more than $h+1$ time points; experimental observations have shown that the computational demand of `rtamt` increases more than linearly with

the horizon length of a formula. For the purposes of expediting experimentation, which includes multiple runs to gather statistically significant results, the *max horizon* was set to 15. Although this may appear restrictive, it remains adequate for extracting significant and effective properties from the data, while also adhering to latency requirements typically considered for control systems in LPREs. In more general applications, the appropriate *max horizon* should be determined by domain experts, considering the aforementioned factors.

3 Material and Methods

Let us now describe our genetic programming-based methodology for learning STL properties that are capable of distinguishing between nominal and anomalous conditions, as well as the salient characteristics of the simulated dataset employed in our evaluation.

3.1 Our STL Property Learner

As outlined in the introduction, we use a genetic programming-based property learner, building on previous work discussed in Section 2.2. We extend this approach by employing a multi-objective fitness function that we define as follows.

To evaluate an individual in the population, the set of STL traces $\mathcal{X}$ is first "partitioned" into *nominal* ($\mathcal{X}_{neg}$) and *anomalous* ($\mathcal{X}_{pos}$) traces. This partitioning is necessary to distinguish between normal and faulty behavior, enabling the algorithm to effectively learn and optimize properties that are sensitive to anomalies while minimizing false positives from nominal traces. A multi-objective fitness function is then defined by using the `rtamt` monitoring algorithm for bSTL. Formally, our first objective measures how good a formula φ is in discriminating between nominal and faulty behavior traces.

▶ **Definition 8.** *For each trace* $\Pi \in \mathcal{X}$ *and each formula* φ*, the numerical counterpart of eb-mon*(Π, φ) *is the function* $NUM : \{\top, \bot, ?\} \to \{0, 1\}$ *s.t.* $NUM(\textit{eb-mon}(\Pi, \varphi)) = 1$ *if eb-mon*$(\Pi, \varphi) = \top$*, or* 0 *otherwise.*

▶ **Definition 9.** *The first objective measure, referred to as accuracy, is defined as follows:*

$$Acc(\mathcal{X}_{neg}, \mathcal{X}_{pos}, \varphi) = \frac{\sum_{\Pi \in \mathcal{X}_{neg}} 1 - NUM(\textit{eb-mon}(\Pi, \varphi)) + \sum_{\Pi \in \mathcal{X}_{pos}} NUM(\textit{eb-mon}(\Pi, \varphi))}{|\mathcal{X}_{neg}| + |\mathcal{X}_{pos}|}$$

It is worth noting that, in order to maximize $Acc(\mathcal{X}_{neg}, \mathcal{X}_{pos}, \varphi)$, a formula φ should evaluate to $\bot$ on the nominal behavior traces and to $\top$ on the anomalous ones.

The second objective measures the robustness of the formula (normalized in the $[0, 1]$ interval) by means of bSTL quantitative semantics. As a preliminary step, at the beginning of the execution of the genetic algorithm, every signal in $\mathcal{X}$ is normalized in the $[0, 1]$ interval so that ρ ranges between -1 and 1. This step is handled implicitly and it does not alter the constant value c of constraints $\Pi_i \geq c$ in the generated output formula, which are still represented with their raw, non-normalized value.

▶ **Definition 10.** *This second objective, the robustness, is defined as follows:*

$$Rob(\mathcal{X}_{neg}, \mathcal{X}_{pos}, \varphi) = \frac{\begin{array}{r} |\mathcal{X}_{neg}| + |\mathcal{X}_{pos}| - \sum_{\Pi \in \mathcal{X}_{neg}} \max_{0 \leq i \leq len(\Pi) - 1 - H(\varphi)} \{\rho(\varphi, \Pi, i)\} \\ + \sum_{\Pi \in \mathcal{X}_{pos}} \min_{0 \leq i \leq len(\Pi) - 1 - H(\varphi)} \{\rho(\varphi, \Pi, i)\} \end{array}}{2 \cdot (|\mathcal{X}_{neg}| + |\mathcal{X}_{pos}|)}$$

The third objective measures the recall of the formula, denoted as *Rec*, which quantifies the ability to correctly identify all anomalous traces.

▶ **Definition 11.** *The recall is calculated as follows:*

$$Rec(\mathcal{X}_{pos}, \varphi) = \frac{\sum_{\Pi \in \mathcal{X}_{pos}} NUM(eb\text{-}mon(\Pi, \varphi))}{|\mathcal{X}_{pos}|}$$

In order to optimize $Rec(\mathcal{X}_{pos}, \varphi)$, the formula φ should evaluate to $\top$ for as many traces in $\mathcal{X}_{pos}$ as possible, effectively maximizing the detection of anomalous behavior.

The fourth objective measure, the specificity, evaluates the ability of a formula φ to accurately reject anomalous behavior in nominal traces.

▶ **Definition 12.** *The specificity is defined as follows:*

$$Spec(\mathcal{X}_{neg}, \varphi) = \frac{\sum_{\Pi \in \mathcal{X}_{neg}} 1 - NUM(eb\text{-}mon(\Pi, \varphi))}{|\mathcal{X}_{neg}|}$$

This measure is crucial for ensuring that φ maintains a high rejection rate of anomalous conditions in traces classified as nominal, thereby minimizing false positives. The ideal outcome for $Spec(\mathcal{X}_{neg}, \varphi)$ is a value close to 1, indicating that most nominal traces are correctly identified as such by φ.

The fifth objective measure, the parsimony, emphasizes the simplicity of the derived formulas, aiming to minimize their complexity.

▶ **Definition 13.** *The parsimony measure is defined as follows:*

$$Pars(\varphi) = 1 - \frac{height(\varphi)}{max\ height}$$

In this context, the function $height(\varphi)$ takes a bSTL formula φ and returns a natural number in $\mathbb{N}_+$ representing the height of the formula's computational tree. The term *max height* refers to a predefined static limit on the height of the computational trees within our genetic programming algorithm. This objective aims to minimize the complexity of the formulas by favoring those with shorter computational trees, thereby enhancing the understandability and efficiency of the derived models.

The sixth objective is the responsiveness which evaluates the promptness of a formula φ in reacting to changes in behaviors. Responsiveness is crucial for ensuring that the formula not only discriminates and measures robustly but also reacts in a timely manner to deviations from nominal behavior.

▶ **Definition 14.** *The responsiveness is formally defined as follows:*

$$Resp(\varphi) = 1 - \frac{H(\varphi)}{max\ horizon}$$

Here, $H(\varphi)$ represents the horizon of the formula φ, and *max horizon* the maximum horizon considered as defined in Def. 5. This metric aims to maximize the timeliness of the response by minimizing the horizon length relative to the maximal allowed horizon, thereby ensuring that φ is both effective and efficient in signaling anomalies.

To conclude, the seventh objective, the detection timeliness measure, quantifies the relative delay in detecting the farthest anomaly within the anomalous traces in $\mathcal{X}_{pos}$. This measure is critical in assessing the timeliness of φ in identifying anomalies, particularly in systems where detection delays may compromise functionality or safety.

▶ **Definition 15.** *Formally, the detection timeliness is defined as follows:*

$$Time(\mathcal{X}_{pos}, \varphi) = 1 - \frac{\max_{\Pi \in \mathcal{X}_{pos}} \{dect\_point(\Pi, \varphi)\}}{\max_{\Pi \in \mathcal{X}_{pos}} \{len(\Pi)\}}$$

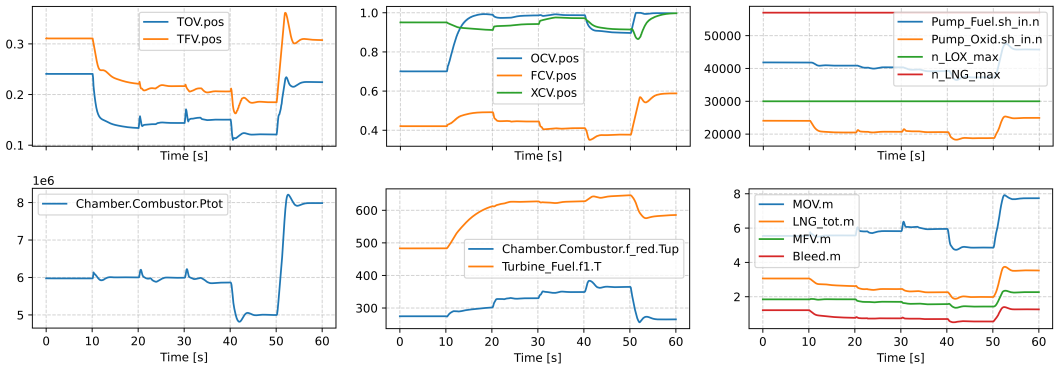

Figure 1 Simulated nominal run values for the engine under standard starting conditions.

In this definition, the detection point function, $dect\_point(\Pi, \varphi)$, determines the earliest time point within a trace Π at which the formula φ is satisfied.

▶ **Definition 16.** *The detection point is calculated using the robustness measure ρ as follows:*

$$dect\_point(\Pi, \varphi) = \min_{0 \leq i < len(\Pi)} (\{i \geq 0 \mid \rho(\varphi, \Pi, i) \geq 0\} \cup \{len(\Pi)\})$$

To maximize $Time(\mathcal{X}_{pos}, \varphi)$, φ should ideally detect anomalies as close to the start of each trace in $\mathcal{X}_{pos}$ as possible, thus ensuring minimal detection latency.

Since seven objectives are taken into consideration, no single best-performing solution can be directly selected from a given population by means of the fitness function. Rather, a *Pareto front* of optimal solutions can be identified, containing all *non-dominated* solutions.[2] Hence, once the evolutionary process is completed, the final solution, maximizing the hypervolume, is extracted from the Pareto front as previously described in Section 2.2.

3.2 Dataset

The dataset used in this work consists of transient state simulation of a representative pump-fed rocket engine of expander-bleed type similar to the LUMEN engine developed and tested by DLR Lampoldshausen [15, 33]. Each simulation was run for a total length of 60 s at a sampling frequency of 10 Hz. For a subset of runs, after a time of 15 s a fault had been injected. The faults considered were: a sensor drift, a sensor offset, a frozen sensor and a leakage in the methane sub-system. To formally define the dataset utilized in our study, let $\mathcal{X}$ represent the complete set of traces, where $\mathcal{X} = \mathcal{N} \cup \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4$. Here, $\mathcal{N}$ denotes the set of traces simulated under nominal conditions, and $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{A}_4$ represent the traces corresponding to the sensor drifting, sensor offset, sensor frozen, and leakage faults, respectively. Each subset $\mathcal{A}_i$ captures the distinct characteristics of the respective fault type, introduced to assess its impact on the system's performance and detectability.

The dataset captures various signals from the engine, categorized into several groups based on the type of measurement: *pressure sensors* measure the pressure inside the combustion chamber, at the inlet of the fuel and oxidizer turbines, and at the outlets from both the fuel

[2] A set $\mathcal{S}$ of solutions for an n-objective problem with fitness function $f = \langle f_i, \ldots, f_n \rangle$ is said to be *non-dominated* if and only if for each $x \in \mathcal{S}$, there exists no $y \in \mathcal{S}$ such that (i) $f_i(y)$ improves $f_i(x)$ for some i, with $1 \leq i \leq n$, and (ii) for all j, with $1 \leq j \leq n$ and $j \neq i$, $f_j(x)$ does not improve $f_j(y)$.

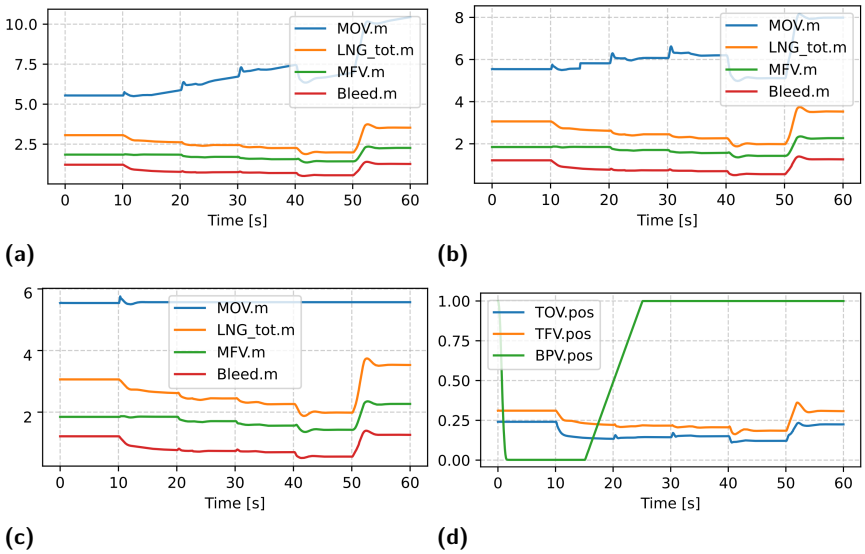

Figure 2 Simulated run values for the engine under standard starting conditions with the injection of faults: **(a)** sensor drift on `MOV.m`, **(b)** sensor offset on `MOV.m`, **(c)** sensor freezing on `MOV.m`, and **(d)** leakage on methane side (`BPV.pos`). For each case, the fault is injected after 15 seconds.

and oxidizer pumps, including inlet and outlet pressures of the cooling system; *temperature readings* are taken from the injected oxidizer and fuel and at the inlet and outlet of the cooling system, as well as at the input of both turbines; *valve positions* are recorded for the oxidizer control valve, fuel control valve, and the valves upstream of both turbines; finally, the *rotational speeds* of both the fuel and oxidizer pumps are also monitored.

Specifically, the features include `Chamber.Combustor.Ptot` representing the total pressure in the combustion chamber, `Chamber.Combustor.f_red.Tup` for the fuel injection temperature, `FCV.pos` indicating the fuel control valve position, `Chamber_Inj_red_dP_loss` and `Chamber_Inj_oxy_dP_loss` for pressure losses at fuel and oxidizer injection respectively. `Pump_Fuel.sh_in.n` and `Pump_Oxid.sh_in.n` measure the shaft velocity of the fuel and oxidizer turbopumps. `TFV.pos` and `TOV.pos` reflect the positions of the turbine fuel and oxidizer valves. Pressures at the inlets of fuel and oxidizer turbines are captured by `Turbine_Fuel.f1.P` and `Turbine_Oxid.f1.P`. `XCV.pos` logs the crossfeed control valve position, while `LOX_Inj.Pressure` and `Fuel_Inj.Pressure` represent the pressures at the liquid oxygen and fuel injection. `BPV.pos` denotes the position of the bypass valve, `MOV.m` measures the mass flow of the main oxidizer valve, `OCV.pos` indicates the oxidizer control valve position, `LNG_tot.m` represents the total mass flow of liquefied natural gas (LNG), `MFV.m` measures the flow rate through the main fuel valve, and `Bleed.m` captures captures the LNG mass flow that is not used for combustion.

These features were selected and analyzed to model the behavior of the propulsion system under nominal conditions and to identify deviations indicative of potential faults. An example of simulated values of these features for a nominal run of the engine under standard starting conditions is depicted in Figure 1, while Figure 2 illustrates four instances of fault injections performed on the `MOV.m` and `BPV.pos` features.

To rigorously evaluate the proposed methodologies for fault detection, the dataset $\mathcal{X}$ was randomly partitioned into training and testing splits to ensure unbiased assessment and validation of the learned properties. This split was done at the level of execution runs, allocating 80% of the data for training and 20% for testing. Formally, the dataset $\mathcal{X}$ was

divided into two subsets: $|\mathcal{X}_{\text{train}}| = 128$, with $|\mathcal{N}| = 64$, $|\mathcal{A}_1| = |\mathcal{A}_2| = |\mathcal{A}_3| = |\mathcal{A}_4| = 16$, and $|\mathcal{X}_{\text{test}}| = 32$, with $|\mathcal{N}| = 16$, $|\mathcal{A}_1| = |\mathcal{A}_2| = |\mathcal{A}_3| = |\mathcal{A}_4| = 4$. This structured approach to dataset division ensures a balanced representation of both nominal and anomalous conditions across the training and testing phases, providing a robust foundation for the evaluation of the fault detection algorithms developed in this study.

4 Experimentation

Our evaluation of the genetic algorithm's capabilities in distinguishing between nominal and faulty operational states under various predefined scenarios showcases our approach's applicability in real-world applications, where the timely and accurate detection of faults can enhance safety and operational efficiency significantly.

4.1 Experimental Setup

In the experimental setup, the genetic algorithm defined in Section 2.2 and 3.1 is employed to learn monitorable temporal properties that are critical for the fault detection task. The learning process is structured around three experimental cases, each designed to test various scenarios of property learning, pertaining to different splits of the trace set $\mathcal{X}_{train}$:

- *Anomaly vs. All:* in this scenario, the genetic algorithm is run separately for each type of anomaly ($\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$) related to the `MOV.m` sensor, where each run aims to learn a distinctive monitor encoded as a temporal property in bSTL that can detect the specific fault. Formally, for each anomaly type $\mathcal{A}_i$, with $i \in \{1, 2, 3\}$, $\mathcal{X}_{pos} = \mathcal{A}_i$, and $\mathcal{X}_{neg} = \mathcal{N} \cup \bigcup_{j \neq i, j \in \{1,2,3\}} \mathcal{A}_j$. This setup facilitates targeted learning for each fault type.
- *All vs. Nominal:* the algorithm runs with $\mathcal{X}_{pos} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3$, encompassing all types of anomalies, and $\mathcal{X}_{neg} = \mathcal{N}$, consisting of only nominal traces. The objective is to extract a general property that differentiates any faulty condition from normal operations.
- *Anomaly vs. Nominal:* similar to the first case, this setup runs the algorithm for each anomaly type $\mathcal{A}_i$, but although, $\mathcal{X}_{pos}$ contains traces from $\mathcal{A}_i$ for all $i \in \{1, 2, 3\}$, and $\mathcal{X}_{neg}$ is exclusively composed of nominal traces in $\mathcal{N}$. This configuration is intended to refine the detection properties to distinguish specific faults directly from normal behavior.

These experimental designs are critical in validating the robustness and specificity of the learned properties, ensuring that the fault detection system is both effective in identifying various faults and efficient in differentiating these from normal operational states.

In each experimental scenario, the learning phase produces a pool of properties denoted as $\mathcal{P}$. These properties are temporal formulas derived through the genetic algorithm, designed to effectively monitor and detect faults within the system. By integrating the `rtamt` monitoring algorithm within the genetic algorithm's fitness function, the properties in $\mathcal{P}$ are designed to be runtime-verifiable. This design allows them to be evaluated on a finite prefix of each execution trace, making them suitable for online monitoring. The efficacy of these properties is subsequently evaluated during the testing phase, where their performance is critically assessed. In this stage, the algorithm as of Listing 1 applies the learned properties to the test dataset $\mathcal{X}_{\text{test}}$ to validate their accuracy and reliability, a crucial step for verifying whether the properties can correctly identify faults without excessive false positives or negatives.

Operationally, the algorithm initializes counters for nominal traces (nn), anomalous traces (np), true positives (tp), and false positives (fp) (line 2). It then iterates through each trace in the test dataset, applying the learned temporal properties to detect faults as in Def. 7 (lines 3–21), determining whether each trace is correctly identified as faulty or nominal.

Listing 1 Fault monitoring (testing phase)

```
input: non-empty pool 𝒫 of formulas, test dataset 𝒳_test
nn, np, tp, fp ← 0, 0, 0, 0
for Π ∈ 𝒳
    if LABEL(Π, len(Π) − 1) % faulty trace
        np ← np + 1
    else      % nominal trace
        nn ← nn + 1
    end if
    for i ← 0 to len(Π)-1
        Σ ← Π[0, i]
        ℱ ← { ψ ∈ 𝒫 | eb-mon(Σ, ψ) returns true }
        if ℱ ≠ empty % fault detected
            if LABEL(Π, i) % true positive
                tp ← tp + 1
            else      % false positive
                fp ← fp + 1
            end if
            break
        end if
    end for
end for
P ← tp / (tp + fp) % Calculate performance metrics, first Precision
R ← tp / np        % Recall
FAR ← fp / nn      % False Alarm Rate
F1 ← 2 * (P * R) / (P + R) % F1-score
return P, R, FAR, F1
```

The function LABEL : $\mathcal{X} \times \mathbb{N} \to \{\top, \bot\}$, is defined such that LABEL$(\Pi, i) = \bot$ if $\Pi \in \mathcal{N} \vee i < 150$, and LABEL$(\Pi, i) = \top$ otherwise. Intuitively, this function categorizes each trace as either nominal ($\bot$) or anomalous ($\top$), facilitating the classification and evaluation of the test results. Here, 150 corresponds to the point in the signal where faults were injected, which, in a general case, may vary across different runs of the system.

The calculated metrics – precision (P), recall (R), false alarm rate (FAR), and F1-score ($F1$) – provide a comprehensive quantitative assessment of the properties' performance. Precision (P) measures the accuracy of fault detection, recall (R) assesses the algorithm's ability to identify all relevant instances, the false alarm rate (FAR) indicates the frequency of incorrect fault predictions among nominal traces, and the F1-score ($F1$) balances precision and recall in a single metric.

This detailed evaluation ensures the robustness and specificity of the learned properties, affirming the fault detection system's effectiveness and efficiency in distinguishing between normal and faulty operational states.

4.2 Experimental Results

Following the detailed description of the experimental setup, we now present and analyze the outcomes of our experiments to showcase the performance of the proposed property learning methodology. For each experimental scenario, a set of 10 trials was performed using 10 different random seeds to ensure the statistical robustness of the results. This led to 10 different pools of properties for each evaluated scenario.

The final experimental results are obtained by running the procedure from Listing 1 on the test set $\mathcal{X}_{test}$. As summarized in Table 1, the results show that the *Anomaly vs. Nominal* scenario achieved the best overall performance across all metrics, indicating a strong ability

to detect distinct types of faults compared to normal behaviors with minimal false alarms. This effectiveness is due to the targeted training approach, where properties were tuned to distinguish specific anomalies from nominal traces.

On the other hand, the *Anomaly vs. All* scenario showed lower precision but very high recall and FAR. This suggests that while the model is effective in identifying nearly all fault conditions, it also misclassifies many normal operations as faults, leading to a high rate of false alarms. This could be due to the inclusion of multiple fault types in the negative class, which can complicate and reduce the effectiveness of the learning phase, as it contains a diverse range of behaviors difficult to characterize within a single property.

The *All vs. Nominal* scenario yields intermediate performance compared to the others, with moderate values across all metrics. This outcome results from the learned properties needing to generalize across various types of anomalies, which leads to higher false alarm rates than those observed in the *Anomaly vs. Nominal* scenario.

These insights are instrumental for refining the fault detection strategies, particularly emphasizing the need for scenario-specific tuning of the learning process to enhance precision and reduce false alarms, thereby making the fault detection system more reliable and applicable in operational settings.

4.2.1 Extended Scenario: Fault Isolation

In this extended scenario, we focus on a simple fault isolation case. Specifically, this scenario advances the *Anomaly vs. Nominal* setup by considering an additional fault type, $\mathcal{A}_4$ (methane leakage), as $\mathcal{X}_{pos}$, while maintaining nominal traces in $\mathcal{X}_{neg} = \mathcal{N}$. Through this extension, the genetic algorithm was tasked with learning properties able to effectively distinguish methane leakage anomalies from normal operational states, culminating in a tailored pool of properties optimized for this specific detection challenge.

Following this learning phase, we obtained two distinct pools of properties: $\mathcal{P}_{\text{sens}}$ for sensor-related anomalies (previously learned for traces in $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ associated with `MOV.m` sensor's anomalies) and $\mathcal{P}_{\text{leak}}$ for methane leakage anomalies in $\mathcal{A}_4$. Subsequently, we embarked on an evaluative phase to assess how these properties, contained within $\mathcal{P}_{\text{sens}}$ and $\mathcal{P}_{\text{leak}}$, perform in classifying these two distinct sources of faults. For this purpose, we applied a modified version of Listing 1 to the combined property pool $\mathcal{P}' = \mathcal{P}_{\text{sens}} \cup \mathcal{P}_{\text{leak}}$ on the subset of all test traces with anomalies, $\mathcal{X}'_{\text{test}} = \{\Pi \mid \Pi \in \mathcal{X}_{\text{test}} \wedge \Pi \notin \mathcal{N}\}$.

We computed several key metrics to evaluate our performance. First, we consider *accuracy*, defined as the number of correct detections divided by the total number of traces in $\mathcal{X}'_{\text{test}}$. We also measured the rate of *missed detections* which quantifies instances where no property in $\mathcal{P}'$ is satisfied. Additionally, we assessed the rate of *overlapping detections*, which measures cases where a trace simultaneously satisfies properties from both $\mathcal{P}_{\text{sens}}$ and $\mathcal{P}_{\text{leak}}$. The experimental results from this extended scenario demonstrate a high degree of *accuracy*, with an average of 0.96 and a standard deviation of 0.03. The rate of *missed detections*, where neither type of fault was identified, averaged at 0.01 with a standard deviation of 0.01. Furthermore, the rate of *overlapping detections*, which indicates instances where both types of faults were erroneously detected simultaneously, averaged at 0.04 with a standard deviation of 0.03.

Table 1 Summary of experimental results for each testing scenario.

Scenario	Precision	Recall	FAR	F1-score
Anomaly vs. All	0.42 ± 0.00	$\mathbf{0.99 \pm 0.01}$	0.99 ± 0.01	0.60 ± 0.01
All vs. Nominal	0.71 ± 0.05	0.88 ± 0.06	0.34 ± 0.08	0.78 ± 0.05
Anomaly vs. Nominal	$\mathbf{0.92 \pm 0.05}$	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.10 \pm 0.07}$	$\mathbf{0.95 \pm 0.03}$

These metrics underscore the robustness of the proposed solution in adapting to a new fault isolation task and illustrate the model's flexibility and effectiveness in differentiating between types of faults that could occur within the system. Nevertheless, the potential of this methodology to be applied in more complex, real-world settings where multiple faults may occur simultaneously requires further validation through tailored experimentation.

5 Discussion

In this section, we delve into a comprehensive analysis of the experimental outcomes derived from our research, addressing both the strengths and limitations of our approach. This exploration aids in understanding the nuanced implications of our methodology and situates our contributions within the broader landscape of fault detection technologies.

The results presented in this study underscore the efficacy of our property learning-based approach in fault detection for LPREs. Particularly in the *Anomaly vs. Nominal* scenario, our method exhibited exceptional precision and recall, highlighting its capability to effectively pinpoint specific fault conditions. Such precise detection is crucial for minimizing operational disruptions and enhancing system safety.

5.1 Strengths and Limitations of the Property Learning Approach

In the following, we present a detailed analysis of the strengths and limitations of our property learning-based approach to fault detection. This evaluation aims to comprehensively understand the potential impact and practical implications of our methodology within the field, providing a balanced view of its practical applicability in real-world settings.

The primary strength of our method lies in its high specificity and sensitivity, particularly notable in scenarios such as *Anomaly vs. Nominal*, where the method demonstrated precise fault detection capabilities. The genetic algorithm's adaptability allows it to tailor properties specifically for different fault types and operational contexts, which is evident from the high performance metrics achieved in targeted scenarios. Furthermore, the integration of the `rtamt` monitoring algorithm facilitates real-time application, a critical feature for operational settings where timely fault detection is paramount. This capability ensures that the learned properties can be applied in real-time, providing timely detections crucial for operational safety. Additionally, the interpretability of the learned properties enhances their usability, making the approach more accessible to domain experts who can understand and trust the detection logic. It is important to point out that the genetic algorithm at the core of our proposed methodology can effectively be integrated in a preemptive failure detection framework, as demonstrated in [7]. This further showcases the versatility of the approach across various operational settings.

However, the complexity of the training process forms a notable challenge. The need to encompass all potential fault scenarios in the training dataset makes the process resource-intensive and potentially cumbersome. Furthermore, our approach faces generalization challenges, as demonstrated by the elevated false alarm rates in scenarios requiring property generalization across diverse fault types. This indicates a trade-off between specificity and generalizability, highlighting the dependence on comprehensive, high-quality training data for effective property learning. Due to the extensive resources required for the property learning phase, this phase should ideally be executed offline. Once learned, monitoring of the properties can occur in real-time and can be executed even on hardware with limited resources, such as the control systems present in launch vehicles, ensuring operational feasibility in critical applications. Additionally, in all anomalous traces considered for this study, a single

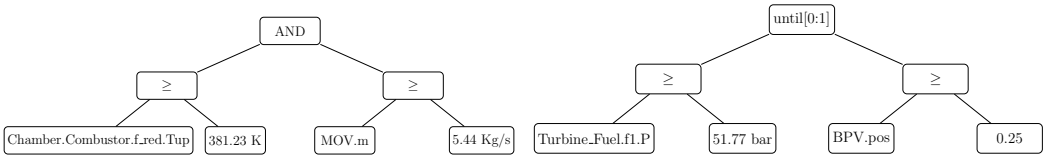

Figure 3 Syntactic trees of two properties extracted from the property learning phase representing **(a)** a `MOV.m` sensor fault and **(b)** a methane leakage.

fault is simulated while in a real-world scenario multiple faults may happen in parallel. This aspect has not been investigated, and it may affect the capability of our solution to function effectively under such conditions.

In summary, the adaptability and real-time application capabilities of our method provide significant advantages for fault detection in complex systems. However, the training complexity and challenges in generalization underscore the need for careful implementation and dataset preparation to fully realize the potential of the proposed approach.

5.2 Interpretability

Interpretability stands as a critical dimension in the assessment of fault detection methods, particularly when applying complex algorithms like genetic algorithms for property learning. The properties extracted in this study underscore not only the method's effectiveness but also its transparency, essential for practical implementation and trust by domain experts.

Figure 3 presents two representative properties derived from the pool obtained during the *Anomaly vs. Nominal* scenario, which exhibited the best performance metrics. These properties are interpreted below, demonstrating the method's ability to link observable engine behaviors to potential underlying faults, making the solution both effective and understandable.

The first property (Figure 3a) relates to the `MOV.m` sensor, crucial in measuring the mass flow of the main oxidizer valve. The condition expressed in this property asserts that if the temperature at the `Chamber.Combustor.f_red.Tup` remains above 381.23 K, while `MOV.m` is simultaneously above 5.44 Kg/s, a fault is likely occurring. This scenario suggests an anomaly where high fuel mass flow does not lead to expected increases in temperature, possibly indicating a sensor reading failure or calibration issue.

The second property (Figure 3b) concerns methane leakage, a critical safety hazard. This property's logic uses the temporal until operator, highlighting conditions that persist over a specific period. It specifies that the pressure at the `Turbine_Fuel.f1.P` should remain above 51.77 bar until the `BPV.pos` exceeds 0.25, indicating abnormal conditions likely due to a leakage affecting a pipeline or manifold close to the bypass valve.

These examples illustrate how genetic algorithms help derive actionable insights from complex data streams in real-time applications. The derived properties are not only high performing but also provide clear, interpretable logic for domain experts. This interpretability bridges the gap between automated fault detection systems and practical engineering applications, ensuring that the outputs are understandable and, hence, adoptable for practical implementation in real-world applications.

Statistics from the *Anomaly vs. Nominal* scenario further highlight the effectiveness and sensible application of the learned properties. Defined by properties with an average horizon of 5.37 (corresponding to 0.537 s in our setup) and an average height of 4.60, the monitors cover sufficient complexity to capture essential dynamics without overfitting, ensuring robustness across different operational settings, while preserving their understandability.

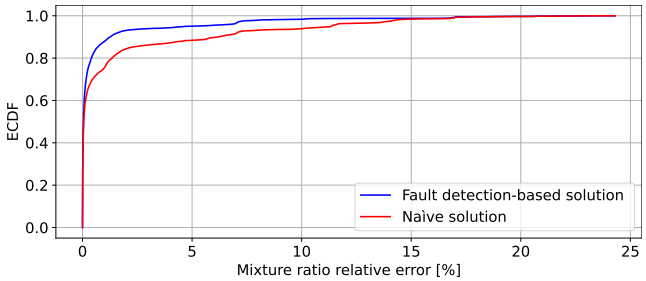

Figure 4 Empirical Cumulative Distribution Function (ECDF) of the mixture ratio relative prediction error for the naïve and fault detection-based virtual sensing solutions.

In order to enhance the intuitiveness of the generated properties, our genetic algorithm's objectives such as *parsimony*, *responsiveness*, and *timeliness* (as defined in Section 3.1) aim to limit the size and temporal horizon of the generated properties. From a structural perspective, let us point out that we can easily deduce from the STL syntax that there are at most two sub-trees for any STL operator, so that there is a direct link between an STL formula's parse tree depth and the size of the formula – both associated with a property's intuitiveness. With the objectives, we thus aim at enhancing the properties' interpretability indirectly (via their structure). Currently, we do not employ an optimization stage that would consider a user's preferences though, which could be implemented, e.g., via rewriting the properties during the generation or a posteriori. A future evaluation of the generated properties by STL engineers in terms of interpretability, and deriving corresponding optimization options (automatically or using a human in the loop) as well as evaluating their effectiveness and efficiency will help us to further improve our concept's usability aspects.

5.3 Virtual Sensing Optimization

Once a fault is detected and identified, leveraging this detection allows for the implementation of various mitigation strategies to reduce the impact caused by the fault. One such strategy involves the use of a virtual sensing model to replace faulty measurements.

We demonstrate this capability on an estimator for the mixture ratio – the ratio of oxidizer to fuel within the combustion chamber – a critical factor in maintaining optimal combustion efficiency and engine performance. This quantity is hard to measure directly due the dynamic conditions during rocket operation. The XGBoost regressor [10] used for the estimation is trained on nominal training data, $\mathcal{X}_{\text{train}} \cap \mathcal{N}$, ensuring that it learns from fault-free operational conditions, as discussed in [34].

In our experiment, two approaches were evaluated: (*i*) *Naïve*, where the estimator utilizes test data in $\mathcal{X}_{test}$ that includes both faulty sensor readings and nominal runs without any adjustments, providing a baseline performance measure; (*ii*) *Fault Detection-Based*, which involves feeding the estimator with test data in $\mathcal{X}_{test}$ that has been adjusted based on real-time monitoring of the engine's operational traces, following the steps outlined in Listing 1. Specifically, when a fault is detected by the monitoring algorithm, mitigation strategies are applied within the `if` branch (lines 12–19) for the remaining timesteps of the run. In this scenario, `MOV.m` sensor values are estimated using a virtual sensing model (another XGBRegressor) that has been previously trained exclusively on the partition of nominal training data $\mathcal{X}_{\text{train}} \cap \mathcal{N}$. This approach aims to correct sensor data anomalies before they are used in mixture ratio estimations, thereby enhancing the accuracy of predictions.

The effectiveness of solution (*ii*) is supported by Figure 4, which compares the empirical cumulative distribution functions (ECDF) of the mixture ratio prediction relative errors for both (*i*) *Naïve* and (*ii*) *Fault Detection-Based* approaches. The *Fault Detection-Based* solution exhibited a clear improvement in prediction accuracy over the *Naïve* one, highlighting the benefits of integrating real-time fault detection with virtual sensing.

6 Related Work

Detecting anomalous behavior in CPS is a vibrant research domain. Machine learning and deep learning are featured prominently in this field, due to their efficacy, even though the resultant black-box models lack interpretability. Recent work highlights neural networks which learn temporal relationships in the data in order to detect faults and failures in cloud data centers [19], aero-propulsion systems [2], disk drives [2, 22] and electrocardiogram (ECG) data [28]. To foster explainability and ensure that solutions are applicable to different domains and contexts, recent approaches combining machine learning and formal techniques have emerged. Specifically, [25, 6, 9] present STL property mining techniques, using genetic algorithms, decision trees, and reinforcement learning to distinguish between different time series data. Bartocci et al [4] provide a survey of 15 procedures covering template-based and template-free, model-based and model-free, active and passive, and supervised and unsupervised methods. According to their categorization, the method presented in here is template-free, model-free, passive, and supervised, i.e. learning from labeled data.

Diagnosis in CPS is an established field that focuses on identifying the root causes of system malfunctions, which is crucial for maintaining system reliability and safety. Notably, [27] has contributed significantly to ML-based diagnosis, highlighting techniques that combine machine learning with model-based approaches for effective monitoring and diagnosability of CPS. Additionally, [11] has explored diagnostic methods specifically for assessing the diagnosability of systems, ensuring that potential faults can be accurately detected and isolated during system operation. Furthermore, [24] demonstrates the application of classifier-based approaches in real-time fault detection scenarios. Unlike these cited works, which primarily address diagnosability and model-based diagnosis, our approach integrates property learning directly into the monitoring process, enabling both real-time fault detection and subsequent diagnosis in a unified process.

7 Conclusions

In this work, we tackled the challenge of developing a multi-objective genetic programming methodology that automatically learns STL properties for fault detection in LPREs, effectively distinguishing between nominal and faulty behaviors. This methodology was validated with a comprehensive dataset simulating various fault conditions, ensuring the robustness and accuracy of the learned properties.

The experimental results demonstrated the effectiveness of the proposed approach in learning STL properties essential for run-time fault detection in LPRE control systems. The high precision and recall in distinguishing normal from faulty behaviors ensure timely and accurate fault detection, which is crucial for maintaining LPRE operational integrity. Moreover, incorporating virtual sensing, guided by detected faults, significantly improved the accuracy of critical parameter estimations, such as the combustion chamber's mixture ratio, underscoring our solution's practical applicability. The interpretability of the learned STL properties further enhances our approach's value, making it both effective and accessible to domain experts, thereby facilitating adoption in real-world aerospace applications.

Future research avenues include developing models for nominal engine behavior to enable (unsupervised) anomaly detection, evaluating cases with simultaneous faults in multiple sensors to assess and improve the robustness in complex scenarios, and devising strategies to minimize redundant physical sensors ensuring the operational functionality of the engine.

References

1 A. Abid, M. T. Khan, and J. 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 K. Aggarwal, O. Atan, A. K. Farahat, C. Zhang, K. Ristovski, and C. Gupta. Two birds with one network: Unifying failure event prediction and time-to-failure modeling. In *Proc. 6th Big Data*, pages 1308–1317. IEEE, 2018. doi:10.1109/BIGDATA.2018.8622431.

3 B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987. doi:10.1007/BF01782772.

4 E. Bartocci, C. Mateis, E. Nesterini, and D. Nickovic. Survey on mining signal temporal logic specifications. *Information and Computation*, 289:104957, 2022. doi:10.1016/J.IC.2022.104957.

5 A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010. doi:10.1093/LOGCOM/EXN075.

6 G. Bombara and C. Belta. Offline and online learning of signal temporal logic formulae using decision trees. *ACM Transactions on Cyber-Physical Systems*, 5(3):1–23, 2021. doi:10.1145/3433994.

7 A. Brunello, D. Della Monica, A. Montanari, N. Saccomanno, and A. Urgolo. Monitors that learn from failures: Pairing STL and genetic programming. *IEEE Access*, 11:57349–57364, 2023. doi:10.1109/ACCESS.2023.3277620.

8 Y. Cao, B. J. Smucker, and T. J. Robinson. On using the hypervolume indicator to compare Pareto fronts: Applications to multi-criteria optimal experimental design. *Journal of Statistical Planning and Inference*, 160:60–74, 2015.

9 G. Chen, M. Liu, and Z. Kong. Temporal-logic-based semantic fault diagnosis with time-series data from industrial internet of things. *IEEE Transactions on Industrial Electronics*, 68(5):4393–4403, 2020. doi:10.1109/TIE.2020.2984976.

10 T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proc. 22nd SIGKDD*, pages 785–794. ACM, 2016.

11 A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. In *Proc. 18th IJCAI*, pages 363–369, 2003.

12 E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, and H. Veith. *Model checking*. MIT Press, 2nd edition, 2018.

13 K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2013. doi:10.1109/TEVC.2013.2281535.

14 A. Donzé, T. Ferrère, and O. Maler. Efficient robust monitoring for STL. In *Computer Aided Verification*, pages 264–279. Springer, 2013. doi:10.1007/978-3-642-39799-8_19.

15 K. Dresia, M. Boerner, W. Armbruster, S. Klein, T. Traudt, D. Suslov, J. Hardi, G. Waxenegger-Wilfing, and J. Deeken. Design and Control Challenges for the LUMEN LOX/LNG Expander-Bleed Rocket Engine. In *Proc. 34th ISTS*, 2023.

16 A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer, 2003.

17 European Commission. Communication from the commission to the european parliament, the european council, the council, the european economic and social committee and the committee of the regions- the european green deal. Technical report,

European Commission, 2019. URL: https://eur-lex.europa.eu/resource.html?uri=cellar:b828d165-1c22-11ea-8c1f-01aa75ed71a1.0002.02/DOC_1&format=PDF.

18 F. A. Fortin, F. M. De Rainville, M. A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(70):2171–2175, 2012. doi:10.5555/2503308.2503311.

19 J. Gao, H. Wang, and H. Shen. Task failure prediction in cloud data centers using deep learning. *IEEE Transactions on Services Computing*, 15(3):1411–1422, May 2022. doi:10.1109/TSC.2020.2993728.

20 J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994.

21 M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009. doi:10.1016/J.JLAP.2008.08.004.

22 S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi. Making disk failure predictions smarter! In *Proc. 18th FAST*, pages 151–167. USENIX, 2020.

23 O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004. doi:10.1007/978-3-540-30206-3_12.

24 I. Matei, J. de Kleer, A. Feldman, M. Zhenirovskyy, and R. Rai. Classification based diagnosis: Integrating partial knowledge of the physical system. In *Proc. 11th PHM*, 2019.

25 L. Nenzi, S. Silvetti, E. Bartocci, and L. Bortolussi. A robust genetic algorithm for learning temporal specifications from data. In *Proc. 15th QEST*, pages 323–338. Springer, 2018. doi:10.1007/978-3-319-99154-2_20.

26 D. Ničković and T. Yamaguchi. RTAMT: Online robustness monitors from STL. In *Proc. 18th ATVA*, volume 12302, pages 564–571. Springer, 2020. doi:10.1007/978-3-030-59152-6_34.

27 O. Niggemann, G. Biswas, J. S. Kinnebrew, H. Khorasgani, S. Volgmann, and A. Bunte. Data-driven monitoring of cyber-physical systems leveraging on big data and the internet-of-things for diagnosis and control. In *Proc. 26th DX*, pages 185–192, 2015.

28 G. Petmezas, K. Haris, L. Stefanopoulos, V. Kilintzis, A. Tzavelis, J. A. Rogers, A. K. Katsaggelos, and N. Maglaveras. Automated atrial fibrillation detection using a hybrid CNN-LSTM network on imbalanced ECG datasets. *Biomedical Signal Processing and Control*, 63:102194, 2021. doi:10.1016/J.BSPC.2020.102194.

29 I. Pill and F. Wotawa. Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software*, 139:124–141, 2018. doi:10.1016/J.JSS.2018.02.002.

30 A. Pnueli. The temporal logic of programs. In *Proc. 18th SFCS*, pages 46–57. IEEE, 1977.

31 S. Pérez-Roca, J. Marzat, H. Piet-Lahanier, N. Langlois, F. Farago, M. Galeotta, and S. Le Gonidec. A survey of automatic control methods for liquid-propellant rocket engines. *Progress in Aerospace Sciences*, 107:63–84, 2019.

32 V. S. Reddy. The SpaceX effect. *New Space*, 6(2):125–134, 2018.

33 T. Traudt, W. Armbruster, C. Groll, R. Hahn, K. Dresia, M. Börner, S. Klein, D. I. Suslov, J. Haemisch, M. A. Müller, J. 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 *Proc. 9th 3AF Space Propulsion*, 2024.

34 A. Urgolo, C. Gei, K. Dresia, M. Freiberger, E. Kurudzija, H. Neumann, I. Pill, F. Pittino, G. Radchenko, and G. Waxenegger-Wilfing. Feature selection and virtual sensing based mixture ratio estimation for liquid propellant rocket engine control systems. In *Proc. 9th 3AF Space Propulsion*, 2024.