Using Multi-Modal LLMs to Create Models for Fault Diagnosis

Silke Merkelbach ✉ ⓘ
Fraunhofer IEM, Paderborn, Germany

Alexander Diedrich ✉ ⓘ
Helmut-Schmidt-University, Hamburg, Germany

Anna Sztyber-Betley ✉ ⓘ
Warsaw University of Technology, Poland

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

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

Oliver Niggemann ✉ ⓘ
Helmut-Schmidt-University, Hamburg, Germany

Roman Dumitrescu ✉ ⓘ
Advanced Systems Engineering, Paderborn University, Germany

Abstract

Creating models that are usable for fault diagnosis is hard. This is especially true for cyber-physical systems that are subject to architectural changes and may need to be adapted to different product variants intermittently. We therefore can no longer rely on expert-defined and static models for many systems. Instead, models need to be created more cheaply and need to adapt to different circumstances. In this article we present a novel approach to create physical models for process industry systems using multi-modal large language models (i.e ChatGPT). We present a five-step prompting approach that uses a piping and instrumentation diagram (P&ID) and natural language prompts as its input. We show that we are able to generate physical models of three systems of a well-known benchmark. We further show that we are able to diagnose faults for all of these systems by using the Fault Diagnosis Toolbox. We found that while multi-modal large language models (MLLMs) are a promising method for automated model creation, they have significant drawbacks.

1 Introduction

Model-based fault diagnosis requires the availability of logical or physical models to reason from observations that deviate from normal behaviour to possible root causes (i.e. faults) [26]. The major drawback common to all model-based fault diagnosis approaches is the limited availability of complete models [5] that exactly describe the behaviour of the underlying system. Obtaining a model is usually an expensive process, requires the close cooperation of experts, and results in most cases in static formulations with respect to the system's architecture. However, changes in architecture and changes in system parameters are common in today's cyber-physical systems (CPS), which places strong requirements on models and thus makes the application of fault diagnosis methods quite rare.

The challenge in model creation is to reliably detect the physical dependencies between components along with each component's behaviour. Formally, this is known as the system identification problem. According to Aguirre [1], system identification is done using a set of measured data to find a mathematical model $\mathcal{M}$ that represents a system in some meaningful way. To build $\mathcal{M}$ exclusively from data is called a black-box identification problem. In grey-box problems, besides data, there is some other source of external information about the system. In this article, we use a white-box approach to system identification, using detailed knowledge about the system's physical laws and underlying mechanisms.

Several authors have attempted to solve the system identification problem for fault diagnosis through black-box and grey-box approaches [7, 18]. Frisk et al. [10] have come up with a semi-automatic white-box approach to create fault diagnosis models from physical equations. But they rely on exact specifications of the system's components, connections, and their governing equations, which we are trying to avoid in this article. Plambeck et al. [23] propose to learn models of hybrid systems using symbolic regression. Their method leverages the power of genetic programming to automatically discover interpretable mathematical models in the form of hybrid systems from observed data. But they require a historical dataset, which is something we want to avoid. We also do not want to rely on complex and hard-to maintain ontologies that some authors have used for this task [11]. So far, large-language models (LLMs) have not been broadly adopted by the fault diagnosis community yet. Some [13, 28] have presented the usage of LLMs for software fault localization. Others have shown how to generate equations with LLMs [8, 24]. Balhorn et al. [2] have used LLMs to correct P&ID diagrams, but have not attempted to generate any kinds of models from the diagrams. Hirtreiter et al. [12] attempted automatic generation of control structures using LLMs. Ogundare et al. [19] have analysed the resilience of LLMs to create system models. But their models are limited to single equations that have no automatic dependencies between each other. Kato et al. [14] extract equations from a large number of scientific documents and create physical models out of them by judging their equivalence using a pre-trained LLM and defining requirements the physical model has to meet. But they neither use P&IDs nor do they create models for fault diagnosis. Peifeng et al. [21] presented the application of fault diagnosis with LLMs in aviation assembly, but this approach requires additional knowledge graphs. The problem of automatic knowledge extraction from P&IDs has been considered in recent works using deep learning and graph search [16]. Sinha et al. [25] propose a solution to detect tables and extract descriptions therein from P&ID diagrams automatically.

In this article we present a novel approach to use common design documentation for cyber-physical systems in the process industry in the form of piping and instrumentation diagrams (P&IDs) to generate a physical model using a multi-modal large language model (MLLM). P&IDs picture the components, sensors, the structure of the system, and the names

of the elements, providing a lot of helpful information for fault diagnosis. MLLMs are, among other things, able to handle images and text as input and create text as output. As MLLM we use OpenAI's GPT4 (ChatGPT) [20] with prompt engineering. As a direct continuation of some of our earlier work [17], we now present a more extended approach, do not provide the equations to the model, and, given the physical model created by ChatGPT, use the Fault Diagnosis Toolbox (FDT) [10] to generate analytical redundancy relations (ARRs). The ARRs are used to: i) compute the fault signature matrix [9]. This is a validation technique to establish how well we can diagnose potential faults. ii) compute residuals to determine faulty signals and thus discriminate faults.

With our contribution we want to present a novel diagnosis methodology for the fault diagnosis (DX) and fault detection and isolation (FDI) communities which uses LLMs, in particular MLLMs, to automatically generate physical models from existing design documentation. Our methodology consists of five steps, takes existing P&IDs as input, and creates models, that can be directly used as input for the fault diagnosis toolbox which can then be applied to create and evaluate ARRs for fault diagnosis. As expected, MLLMs still have severe limitations and our approach lacks generalisation to systems that are outside of well-known water tank benchmarks and similar process industrial examples. However, using our method in a generate-and-test approach we believe that practitioners can generate a statistically significant number of models and then select those that i) compile and ii) have a sufficiently high isolability. We believe that our approach has strong benefits in practical use-cases where models of systems with known components need to be created cheaply. Our method enables practitioners to generate new models whenever an updated system documentation exists. It also directly enables the generation of residuals from the identified model and thus decreases the costs to create self-diagnosing and resilient systems.

We have evaluated our approach on a one-tank, one three-tank, and one four-tank system of the benchmark provided by Balzereit et al. [3]. We first evaluated each system qualitatively and then used it together with the FDT to perform a quantitative evaluation.

2 Background

In this section, we explain the basis of our physical models, in particular how they integrate faulty behaviour, how we process observations from the system, and how we create diagnosis tests.

▶ **Definition 1** (Measurement, Control Value). *A measurement is a single measured value from a CPS sensor $z_{s_i}(t) \in \mathbb{R}$. A control value is a single value from a CPS actuator $z_{c_i}(t) \in \mathbb{R}$.*

Measurements and control values form the sets of known variables $\mathcal{Z} = \mathcal{Z}_s \cup \mathcal{Z}_c$. For simplicity, time notation is often omitted, assuming values are taken concurrently. We thus write z_i instead of $z_i(t)$. There are two types of faults: additive and multiplicative.

▶ **Definition 2** (Additive Fault). *An additive fault is a fault that changes the value of some measurement or control value $z_i \in \mathbb{R}$ by introducing an offset $f_i \in \mathbb{R}$ in an additive way, so that the sensor/actuator reads $z_i' = z_i + f_i$ instead of z_i.*

▶ **Definition 3** (Multiplicative Fault). *A multiplicative fault is a fault that changes the value of some parameter $p_i \in \mathbb{R}$ by introducing a factor $f_i \in \mathbb{R}$ that impacts on the system's dynamics.*

Additive faults occur when some bias is introduced into the system, such that the measurement is changed by some fixed amount. For example, the wrong configuration of some analog-to-digital converter, or the introduction of the wrong control voltage may lead

to additive faults. Conversely, multiplicative faults, also called parameter faults, affect the system's dynamics and hence, the stability of the system. They may lead to components limited in their functionality. For example a valve that does not open anymore, may limit its throughput to only 30%.

A physical model describes the information flow within a CPS. Through the use of physical relations, measurement values and intermediate values can be predicted.

A system model $\mathcal{M}(z, x, f)$ generally involves non observable or unknown variables, also named state variables, denoted x, and known variables z as defined previously. State variables x decompose in differential state variables x_1 and algebraic state variables x_2. The faults may be represented explicitly through a specific parameter vector f. The sets of known variables, unknown variables, and faults are denoted by $\mathcal{Z}$, $\mathcal{X}$, and $\mathcal{F}$ respectively. Note that in this paper, we limit ourselves to continuous dynamics physical systems. A typical model, known as a *state-space model*, may be formulated in the temporal domain as follows :

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

where $x_1(t) \in \mathbb{R}^{n_1}$ and $x_2(t) \in \mathbb{R}^{n_2}$ are the vectors of state variables (unknown), $z_s(t) \in \mathbb{R}^m$ and $z_c(t) \in \mathbb{R}^l$ denote the output and input vectors (known variables). $z_c(t)$ may be equal to 0 in case of an uncontrolled system. The functions h, l, and g are linear or nonlinear functions that involve a set of parameters denoted $\mathcal{Z}_p$ (as tank diameter, nominal flow, ...).

For any vector ν, let us define $\bar{\nu}$ to stand for ν and its time derivatives up to some (unspecified) order.

▶ **Definition 4** (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 5** (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 simpler terms, a residual generator produces a signal (residual) that should be zero when the system is working correctly, and any deviation from zero indicates a potential fault.

From the system model, ARRs can be obtained based on the analytical redundancy embedded in the model. For this, variable elimination can be applied, thereby obtaining relations that involve only known variables. ARRs allow us to assess whether the measurements z are consistent with the model $\mathcal{M}(z, x, f)$, hence defining *diagnosis tests*. Once a fault exists within the CPS it will hopefully have some influence on the measurements and therefore also on the residuals. If it does not have any detectable influence then the fault is said non detectable and it cannot be detected [15].

Following the ideas from Cassar and Staroswiecki [4] and Travé-Massuyès et al. [27], structural analysis can be advantageously used to obtain ARRs. It consists in 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, even under uncertainty.

When used for fault diagnosis purposes, structural analysis allows one to find subsets of equations endowed with redundancy. 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.

Actually, minimal subsets of equations endowed with structural redundancy have been proved to provide sets of equations supporting diagnostic tests [15]. These have been defined as Fault-Driven Minimal Structurally Overdetermined (FMSO) sets [22]. Assume $\mathcal{F}_\varphi$ as the set of faults that are involved in a set of equations $\varphi \subseteq \mathcal{M}(z, x, \mathtt{f})$.

▶ **Definition 6** (FMSO set). *A subset of equations $\varphi \subseteq \mathcal{M}(z, x, \mathtt{f})$ is an FMSO set of $\mathcal{M}(z, x, \mathtt{f})$ if (1) $\mathcal{F}_\varphi \neq \emptyset$ and $\rho_\varphi = 1$, (2) no subset of φ is overdetermined, i.e. with more equations than unknown variables. The set of FMSO sets of $\mathcal{M}$ is denoted Φ.*

FMSO sets can be converted into ARRs. By their nature, all the undetermined variables involved in an FMSO set φ can be resolved using $|\varphi| - 1$ equations. These variables can subsequently be substituted into the $|\varphi|^{th}$ equation to formulate an ARR off-line, which is then utilized on-line as a diagnostic test. Furthermore, the concept of an FMSO set is crucial for defining detectable faults and isolable faults. Here, we revisit these definitions [15].

▶ **Definition 7** (Detectable fault). *A fault $f \in \mathcal{F}$ is detectable in the system $\mathcal{M}(z, x, \mathtt{f})$ if there exists an FMSO set $\varphi \in \Phi$ such that $f \in \mathcal{F}_\varphi$.*

▶ **Definition 8** (Isolable faults). *Given two detectable faults f and f' of $\mathcal{F}$, $f \neq f'$, f is 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) [10] is designed for the analysis and creation of fault diagnosis systems for dynamic systems, which are mainly characterized by differential-algebraic equations. Utilizing a structural model, this toolbox facilitates the production of FMSO sets. From these sets, it can automatically produce ARRs that are employed as diagnostic tests.

To summarise, in this article we are using a MLLM to generate system models. Then we use the above theory and employ the FDT to create ARRs and residuals for fault diagnosis.

3 Creating Physical Models for Fault Diagnosis Using MLLMs

This section presents a novel approach to create physical system models for fault diagnosis using a five-step approach to query a MLLM. Our goal is to obtain a physical model $\mathcal{M}$ describing a cyber-physical system's normal behaviour put in a form $\tilde{\mathcal{M}}$ which can be used as input for the FDT [10]. Using our model with the FDT we want to show how to automatically diagnose faults using the theory described in the section above. In particular, we solve $\mathcal{P}, \mathcal{I} \to \tilde{\mathcal{M}}$, with a P&ID $\mathcal{P}$, some additional external information $\mathcal{I}$, and create the physical model in a format suitable for the FDT $\tilde{\mathcal{M}}$.

Our goal is to obtain a method that practitioners can use to quickly create new system models without much expert knowledge and to test the capabilities of modern MLLMs in the domain of fault diagnosis. To realise this, we are relying mainly on the design of a system prompt of MLLMs, which we attempted to keep as generic as possible. In the user prompt, information specific to the system being modeled is provided. We also make use of the FDT, which is an established technique to generate ARRs for fault diagnosis from physical models. We will therefore show that our method can automatically create physical fault diagnosis models for process industrial use cases.

In every step, a prompt with some external information is sent to the MLLM that can be summarised as the tuple $(\mathcal{P}, \mathcal{A}, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{X}, \mathcal{F}_{des})$, with P&ID $\mathcal{P}$, assumptions for the physical model $\mathcal{A}$, parameter names $\mathcal{Z}_p$, control variable names $\mathcal{Z}_c$, unknown variable names $\mathcal{X}$, and faults that should be detected, with a short description, $\mathcal{F}_{des}$. We refer to $\mathcal{A}, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{X}, \mathcal{F}_{des} \in \mathcal{I}$ with external information $\mathcal{I}$.

The five steps of our prompting approach are the following: i) Let the MLLM read the diagram image data and represent it in some partially specified intermediate format, ii) Identify the sensors from the diagram, iii) Create the physical equations, iv) Match sensors and variables, and v) Format the model for the FDT. The steps are formalised in Algorithm 1 and will be described in more detail below. 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. The prompts can be found in GitHub[1]. So far, our approach aims to work for water tank systems with standard components, such as tanks, pumps, valves, flow indicators, and level indicators.

■ **Algorithm 1** `Create Physical Models with MLLMs.`

Data: $\mathcal{P}, \mathcal{A}, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{X}, \mathcal{F}_{des}$
Result: $\tilde{\mathcal{M}}$
1 $CompsConnections \leftarrow ReadDiagram(\mathcal{P})$;
2 $Sensors \leftarrow IdentifySensors(\mathcal{P}, CompsConnections)$;
3 $Equations \leftarrow CreateEquations(\mathcal{P}, CompsConnections, \mathcal{X}, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{A})$;
4 $\mathcal{M} \leftarrow Matching(\mathcal{P}, Sensors, Equations, CompsConnections, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{F}_{des})$;
5 $\tilde{\mathcal{M}} \leftarrow Formatting(\mathcal{P}, \mathcal{M})$
6 $return\ \tilde{\mathcal{M}}$;

Step 1: Read Diagram. In the first step, the components of the system and their connections are extracted from the diagram $\mathcal{P}$ with the vision capabilities of the MLLM. The output, which is formatted as a table with additional explanatory information is captured in *CompsConnections*. The model receives context about the diagram in the system message. It is requested to output the components and their connections in a simple table that can easily be interpreted by the MLLM in later steps. To avoid that the sensors are listed as elements, they should be ignored in this step. The model is also instructed to write 'unclear' for connections it cannot identify. In previous work we found that including these 'unclear' connections significantly reduces hallucinations [17]. At the end, the MLLM is told to make sure that the valves only occur once in the input and output columns. Otherwise, inconsistent models are generated that cannot be used by subsequent steps. In this step, no user prompt is needed, the input is just the diagram.

Step 2: Identify Sensors. The second step creates another table *Sensors* containing the existing sensors $\mathcal{Z}_s$, their type, and their placement from diagram $\mathcal{P}$ and from the output from step 1 in the form of *CompsConnections*. Context about P&IDs, possibly occurring

---

[1] `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 physical models, and all intermediate outputs.

sensors, and the placement of the sensors in the diagram are provided in the system message. To reduce hallucinations, the MLLM is advised to check if the combination of number and sensor type is actually in the image and that each number can only occur once. To make sure there is an output, the MLLM is advised to try the task, even if it tends to say the diagram is too complex. The user prompt contains only the input from the previous step *CompsConnections*.

Step 3: Create Equations. For the creation of the physical equations *Equations*, the MLLM gets the diagram $\mathcal{P}$, the names of the unknown variables $\mathcal{X}$, *CompsConnections*, the parameter names $\mathcal{Z}_p$, the control variable names $\mathcal{Z}_c$, and the assumptions $\mathcal{A}$ as input. The model is instructed to use as many of the given symbols as possible to take care that the equations are detailed enough for the later steps. It is also told to calculate the values that could be measured with the sensors (volume flow through valves and pumps, and levels of the tanks) to make the model suitable for fault diagnosis later. The 'unclear' connections from *CompsConnections* are excluded to obtain an executable model. In our approach, we prefer an incomplete model over a wrong model. At the end of the system message, there are some formatting instructions to prepare the equations for the usage in the FDT. Unlike in our previous work [17], we do not provide any equations and rely completely on the equations known to the MLLM to increase usability. In the user prompt, we include the external information $\mathcal{I}'$. To provide the model with additional context it is exemplary mentioned that the variables and parameters have the same names as in OpenModelica. When the method is applied to other systems that are not modelled with OpenModelica, the variables and parameters should have self-explaining names such that the MLLM is able to use them correctly. In this case, the hint to OpenModelica in the user prompt would need to be adjusted accordingly.

Step 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 physical model $\mathcal{M}$ is created. The step takes as input the diagram $\mathcal{P}$, the identified *Sensors*, the *Equations*, *CompsConnections*, the parameter names $\mathcal{Z}_p$, the control variable names $\mathcal{Z}_c$, and the faults with a short description $\mathcal{F}_{des}$. The system message contains an explanation of how to handle the input, how to match the sensors $\mathcal{Z}_s$ with the variables from the equations, to identify parameters $\mathcal{Z}_p$, and how to handle control variables $\mathcal{Z}_c$. In addition, it is instructed to add the faults $\mathcal{F}_{des}$ as multiplicative or additive faults to the model. The variables should be stored in a dictionary, assigning them to the suitable key depending on if they are known ($\mathcal{Z}$), unknown ($\mathcal{X}$), faults ($\mathcal{F}$), or parameters. The user prompt contains only the mentioned input with a short introductive description.

Step 5: Format Model for Fault Diagnosis Toolbox. In the final step, the physical model $\mathcal{M}$ is transformed to be suitable for the Fault Diagnosis Toolbox into $\tilde{\mathcal{M}}$. In the system message, the structure of the model is provided which just needs to be filled in by the MLLM. It contains many specific instructions to how the format should look like and to force the MLLM to output only the desired format, since otherwise, $\tilde{\mathcal{M}}$ is not executable. The user prompt only contains the physical model $\mathcal{M}$ that already includes all the other external information at this point.

4   Evaluation

To evaluate our approach we used three different multiple tank systems (i.e. S1-S3) from the benchmark from Balzereit et al. [3]. The P&ID for system S1 is shown exemplarily in Figure 1. We manually re-created the P&IDs to ensure that the MLLM has not seen them before as part of its training data. The P&IDs follow the standard DIN EN ISO 10628 [6]. For each system we generated physical models to evaluate our MLLM approach. For the qualitative evaluation, we compared the models created by the MLLM to manually created models. We looked at the following aspects: visual analysis of the diagram, usability of the model, and correctness of the model. In addition, we performed a quantitative evaluation to check how our approach is able to diagnose real injected faults with simulated data.
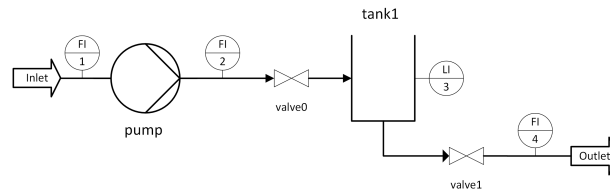
Figure 1 P&ID diagram of the one-tank system S1.

The implementation was done in Python, using the OpenAI library to call OpenAI's GPT4 via the API. The version of the model is 'gpt-4-vision-preview' and the following parameters were used to make the model less creative and more reliable: seed=42, temperature=0, top_p=0.1, frequency_penalty=0, and presence_penalty=0. We repeated the experiment for each of the three systems 100 times. For each step's input we used exactly the output of the previous step.

The data we used was created using OpenModelica 1.13 with the benchmark of Balzereit et al. [3]. We extracted the variable names and excluded internal variables (those starting with '$cse'). We listed the names of control variables $\mathcal{Z}_c$ separately and created another list with the names of the parameters $\mathcal{Z}_p$. A list with faults to be detected $\mathcal{F}_{des}$ was created manually for the systems. The assumptions $\mathcal{A}$ are used to simplify the model and are the same for all systems. An excerpt of external information $\mathcal{I}$ for system S1 is shown in Table 1. An overview of the number of elements and names is listed in Table 2.

Table 1 Inputs for S1.

Input	Count	Content
Parameter names $\mathcal{Z}_p$	16	['pipe_Diameter', 'pipe4_Diameter', 'tank1_Diameter', 'pipe1_Diameter', 'pipe2_Diameter', 'pipe_Length', ...]
Control variable names $\mathcal{Z}_c$	3	['pump_N', 'valve0_opening', 'valve1_opening']
Unknown variable names $\mathcal{X}_n$	348	['time', 'pump_medium_T', 'pump_medium_p', 'tank1_level', 'tank1_medium_T', 'der_pump_medium_T', 'der_pump_medium_p', ...]
Faults $\mathcal{F}_{des}$	4	Leakage of tank1: 'f_tank1leak'. Valve0 blocked: 'f_valve0'. Valve1 blocked: 'f_valve1'. Degraded rotational speed of pump: 'f_pumpSlow'.
Assumptions $\mathcal{A}$	5	\nThe fluid in the system is water\nThe fluid is incompressible\nThere are no energy losses\nThe process is adiabatic\nThe tanks are open.

Table 2 Overview of the number of elements and names in S1-S3.

System	Components	Sensors	$\mathcal{Z}_p$	$\mathcal{Z}_c$	$\mathcal{X}_n$	$\mathbf{F}_{des}$	$\mathcal{A}$
S1	4	4	16	3	348	4	5
S2	8	7	32	5	691	5	5
S3	11	9	36	7	1007	4	5

4.1 Qualitative Evaluation

Our motivation for the qualitative evaluation was to generate insights on how well current MLLMs can be used for fault diagnosis. We therefore manually analyzed the models created for each system. The models were assessed according to the following criteria: i) Visual analysis of the diagram. We investigated the recognition of elements, of connections, and sensors (step 1 and step 2). ii) Usability of the model. We identified errors that lead to non-executable models and analysed how this affects usability. iii) Correctness of the model. We checked the assignment of the variables to the four categories (unknown variables $\mathcal{X}$, faults $\mathcal{F}$, known variables including control variables $\mathcal{Z}_c$ and sensor variables $\mathcal{Z}_s$, and parameters $\mathcal{Z}_p$), the number and design of equations, and if the assumptions were interpreted correctly. We will now present each of the analysis results in detail.

i) Visual Analysis of the Diagram. In step 1 (Read Diagram) almost all components were recognized correctly, except for some cases in S3, where *valveLinear7* was missing. The sensors were successfully ignored in this step, such that they are not misclassified as components. The complete structure of S1 and S2 was identified correctly in all cases. For S3 with its complex structure and more elements, some connections were hallucinated or confused. The upper part of the diagram was always recognized correctly, while the connections to valveLinear7 were never identified and the output of valveLinear5 was interpreted diversely. This leads to wrong inputs for tank4 and, in some cases, valveLinear6. Some repetitions were completely correct, except for valveLinear7, which was marked as unclear. A model resulting from that would be incomplete and not wrong. In step 2 (Identify Sensors) the level indicators were nearly always correct in all systems. The flow indicators were always correct in S1. In S2, FI3 was always wrong, FI7 was sometimes right. The sensors were interpreted left of valveLinear1, and valveLinear3, respectively, instead of the actual placement at the right side of the valves. The wrong placement does not necessarily make the model wrong in the end, if the sensor is assigned to the valve and there is no leak in the pipe. The same issue occurred in S3, where in addition two sensors (FI15 and FI16) were identified at wrong locations, such as between valveLinear3 and tank3 for sensor FI15. Also in the mostly correct repetitions, these two sensors were at the wrong position, leading to wrong measurement assignments. In total, there was not much variation in the outputs of steps 1 and 2, the same tables occurred quite often. For a more detailed evaluation on the frequency of unique results, please check our previous work [17].

ii) Usability of the Model. We validated the output of step 3-5 by checking the final model for its usability from an expert's standpoint. In all repetitions for all systems, the models had the correct basic structure, consisting of the import of the required libraries and the dictionary in which the model was stored. The correct variable names were used in most cases but sometimes variables were not defined or contained special characters which cannot be handled by the Diagnosis Toolbox, leading to non-executable models. ChatGPT was

advised to check if variables are in the equations but not in the dictionary with the variables. Instead of adding the missing variables to the dictionary directly, they were added at the end of the model in some cases. The missing symbol definition resulted in a non-executable model, if the symbol was needed for solving the equations. It did the same for equations, but since they do not need to be defined as symbols, the models were still executable. Some model definitions being not exactly consistent with the modeling conventions of the Diagnosis Toolbox, for example that only variables in $\mathcal{X}$ can be used in differential constraints, often result in index out of bound errors during matching computation while creating the residuals. For residuals, the values for some parameters, such as π or the gravity acceleration, were not present since they were not defined in the model provided by ChatGPT. Some of the successfully computed residuals did not work due to numerical errors, such as square root of a negative number, division by zero, or numerical overflow. An examination of how many residuals were calculated correctly, can be found in the quantitative evaluation.

In addition to the errors leading to non-executable models, we made some observations that did not disturb the usability of the model. The format of the models varies. Sometimes the equations were defined directly in the model structure, sometimes they were defined before and converted afterwards. Sometimes they were stored in variables that are summarized in a list later, sometimes they were directly listed. And sometimes single equations were added to the dictionary at a later point. In the majority of the models, the equations were suitably structured and the code was well commented, making it easy to gain an overview of the model.

iii) Correctness of the Model.   The assignment of the given variables ($\mathcal{X}$, $\mathcal{F}$, $\mathcal{Z}_c$, $\mathcal{Z}_s$, and $\mathcal{Z}_p$) to the four categories unknown variables, faults, known variables, and parameters, was in most cases completely correct, and depended on the sensors that were identified in step 2. Faults and known variables were always correct, parameters sometimes contained values that were not in $\mathcal{Z}_p$. Sometimes, the unknown variables also contained some parameters, which is probably due to the instructions of step 4 (Sensor Matching) to store all not-assigned variables in the unknown variables. The consequence are more unknown variables and thus potentially an under-determined model for the system.

The number of equations varies strongly. Many models have one equation per component as intended, others have more redundant equations, such as equations for the water volume and mass in the tank in addition to the level. In general, there are many models with correct equations. In some models, single equations were wrong. In other models, it seems like more assumptions were applied than provided, resulting in simpler equations with fewer variables, such that, for example, the model works only with nominal flow. In general, the assumptions were included as intended, for example by ignoring the density of water. Sometimes the wrong variables were used, such as the pump's volume instead of the pump's volume flow. For each sensor there should be an extra equation but in some cases, the sensors were added to other equations as well. Sometimes the sensors were matched with the wrong variables, for example a level sensor was matched with the derivative tank level instead of the tank level. Under these conditions, residuals can be calculated without errors but do not work according to theoretical fault sensitivity, which will be checked in the quantitative evaluation. Another aspect are missing equations. ChatGPT does not add flow balance equations to valves or pumps, only to tanks. More faults might be detected with these equations. The system message of step 3 (Create Equations in Python) might be the reason why the balance equations are not there.

4.2 Quantitative Evaluation

The Quantitative evaluation was done with the Fault Diagnosis Toolbox [10]. Our five-step approach generated models in the form of a Python module that is expected to be consistent with FDT requirements. Only modules that load without errors are considered for further evaluation. We created a FDT model object for each model, computed all FMSO sets [15], and selected only the sets that can give residuals in integral or algebraic causality. Integral causality is preferred to derivative causality because numerical differentiation is noisy, and residual generators in integral causality correspond to the standard state-space formulation (eq. 1). For each FMSO, we tried to create a residual generator (corresponding to an ARR). We evaluated each correct residual generator on simulated benchmark data and verified fault detection and isolation performance. For each system S1-S3, we evaluated 100 generated models.

Table 3 presents the number of models imported into Python without errors and the mean number of FMSOs computed for each model. For each FMSO, we tried to compute residual generators. The following rows of Table 3 show the mean number of residual generators computed without errors (in the ideal case, we have one residual generator per FMSO) and the number of residuals that were correctly evaluated on the simulation files (in the ideal case each residual generator gives one residual that can be assessed with data). For S1, we consider tank leakage (tank1leak) and blockage of valve0. The last rows of Table 3 show the fraction of cases when these faults were successfully detected out of all correctly imported models. Fault detection results for S2 and S3 are presented in a similar way.

Additionally, Table 4 shows the frequencies of ambiguity groups. For S1, in 74% of cases, all system states are correctly isolated; in 12% of cases, only tank1 leak can be detected; in 10% of cases, only valve0 fault can be detected, and in 3% of cases, none of the faults is detected. For S2, in 73% of cases, none of the faults are detected; in 25% of cases, only tank2 leak can be detected, and in 2% of cases, all system states are correctly diagnosed. For S3, in 1% of cases, all faults can be detected; in 4% of cases, all faults except tank2 leak are correctly diagnosed, and in 34% of cases, none are detected. We assume that the errors in the image interpretation lead to the bad performance for S3. Table 4 also shows the statistically required number of executable models to expect at least one that supports the ambiguity group with a chance of 95%. The number of models n is calculated with

$$n \geq \frac{\ln(1 - P)}{\ln(1 - p)}, \tag{2}$$

where P is the desired chance (95%), and p is the frequency in which the ambiguity group was observed.

Figure 2 shows computed residuals for module S1_3, which gives complete fault isolability for two simulation files. In each file, the fault starts 250 seconds after the start of the simulation. We can observe that residual generator ResGen0 is sensitive to tank leakage, and residuals ResGen2 and ResGen4 are sensitive to valve blockage. ResGen0 computes tank level from the flow measurements and equation e_1; ResGen2 and ResGen4 use valve equations. It can be observed that these residuals are not precisely zero on the time intervals where they should, for instance ResGen2 and ResGen4 should be zero in [t=0,t=250], as the approximate valve flow is based on nominal flow and valve opening. As valve differential pressure measurement is not available, this is a reasonable attempt.

For comparison, we created the model for S1 by hand (see GitHub). It correctly detects and isolates two faults considered for S1 (residuals in Fig. 3). Therefore, we achieve optimal performance in 74% of automatically generated models and would need 3 models to have a probability of at least 95% that one of the models has optimal performance (Table 4).

■ **Table 3** Model results of models generated for S1-S3. Valve faults are cloggings and tank faults are leaks.

| System	S1		S2		S3	
# correctly imported models	93		85		83	
Mean # FMSOs	5.40		22.27		48.70	
Mean # corr. res. gen. per FMSO	0.9830		0.2247		0.5783	
Mean # correct res. per gen.	0.9825		1		0.9251	
Individual Fault Isolation Accuracy	valve0	0.8495	tank1	0.0235	pipe4	0.2169
	tank1	0.8710	tank2	0.2706	tank2	0.0120
					valve3	0.6627
					valve6	0.1928

For S2 and S3 we found a common pattern for worse performing cases. We observe that the generated models do not adequately handle the situations where the flow is split (like after pump in S2) and where the flows merge (like before tank4 in S3). The lack of direct measurements of tank inflows causes poor performance for tank1 leak in S2. The poor performance for tank2 leak in S3 is caused by the lack of measurement of tank4 inflow. These situations can be correctly detected with correct models but require writing balance equations for flows.

■ **Table 4** Ambiguity groups for S1 and S3 with the required sample size (indicating how many executable models need to be created to have at least one that works for the ambiguity group with a probability of 95%).

System	Ambiguity groups	Frequency	Required Sample Size
S1	{valve0}, {tank1leak}, {NF}	0.7419	3
	{valve0, NF}, {tank1leak}	0.1183	24
	{valve0}, {NF, tank1leak}	0.0968	30
	{valve0, tank1leak, NF}	0.0323	92
	{valve0, tank1leak}, {NF}	0.0108	276
S2	{tank2leak, NF, tank1leak}	0.7294	3
	{tank2leak}, {NF, tank1leak}	0.2471	11
	{tank2leak}, {NF}, {tank1leak}	0.0235	126
S3	{valve6, tank2leak, pipe4, NF}, {valve3}	0.4458	6
	{pipe4, valve6, valve3, NF, tank2leak}	0.3373	8
	{pipe4, valve6}, {NF, tank2leak}, {valve3}	0.1325	22
	{pipe4}, {NF, tank2leak}, {valve3}, {valve6}	0.0482	61
	{pipe4}, {valve3}, {valve6, tank2leak, NF}	0.0241	123
	{pipe4, valve6, tank2leak, valve3}, {NF}	0.0120	249

5   Discussion

The evaluation shows the potential of MLLMs like OpenAI's GPT4 (ChatGPT) for the fully automated creation of models for fault diagnosis. Within the five steps of our approach, there are many influencing factors and many decisions to be made by the MLLM that are potentially wrong. Especially because MLLMs are highly non-deterministic. Under these circumstances, we find the results to be surprisingly good. Even though the quantitative
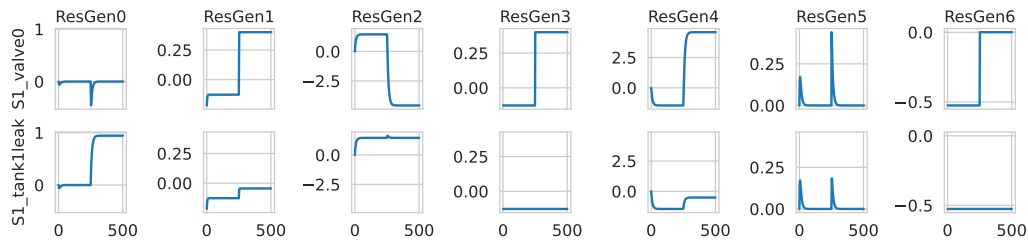
Figure 2 Residuals for model S1_3.
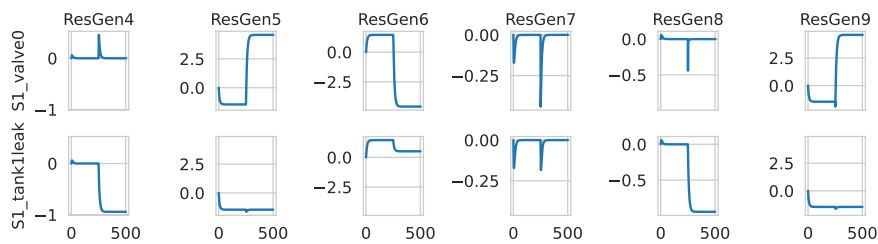
Figure 3 Residuals for the correct model.

evaluation shows severe limitations of overall fault diagnosis performance, it indicates that the creation of such models is possible with MLLMs, especially to support practitioners with some acceptable drafts they can choose from. By creating a number of models with our approach and evaluating them in terms of executability and correctness, a good model can be selected. However, in its current form, our approach is not reliable enough to be applied without an expert checking the models.

We evaluated our approach with simulated data from OpenModelica. Hence we do not know how it reacts to other sources of data, such as real-world data. In S3, some of the faults were easier to detect than tank1 leak in S2, which makes the results not fully comparable between the systems. In our opinion, the image analysis is the biggest issue, as those diagrams are the most common in the process industry. While the image analysis works perfectly for small systems, the missing ability to detect long lines and the correct placement of sensors for complex systems leads to errors in the first two steps which then subsequently propagate. With sensors assigned to the wrong variables, it is not possible to calculate correct residuals, even though the residual generators were created successfully. A more reliable method to extract the system structure needs to be developed. Alternatively, the extraction of the structure and sensor placements could be done manually and provided to the model as input for step 3. We assume, that the approach does not work well for systems in which the flow is split, since ChatGPT did not create flow balance equations correctly. This issue might be solved by explicitly mentioning it in the prompt of step 3. Another way to improve the results could be to include sample equations to reduce the arbitrariness of the models and to be less dependent on the data the MLLM has seen during training. This was done in our previous work [17] successfully but has the disadvantage that suitable equations need to be identified in advance. Our approach was only validated on ChatGPT so far which makes it unpredictable how it might work with other MLLMs.

Unfortunately we had to restrict our approach within the prompts with the following assumptions. Our approach is intended to only work for flow sensors that are between exactly two elements, valves that are connected to one element without a split or a merge, components and sensors with unique names, and P&IDs that follow the standard DIN EN

ISO 10628 [6]. The units of the variables and parameters were not provided to ChatGPT and could lead to wrong residuals if they are not consistent. Since MLLMs are inherently non-deterministic, the reliability of our quantitative evaluation based on 100 repetitions for each system is uncertain.

6   Conclusion

We presented a novel five step-approach to generate physical models for fault diagnosis with MLLMs. We showed that it is possible to generate suitable models with a completely automated chain for small systems. In 74% of the cases we were able to detect and isolate all faults for the small system S1. In 97% of the cases, we could detect and isolate at least one fault correctly. For the other two systems, the performance was weaker, leading to a detection rate of 27% for S2 and 66% for S3. For S3 in no case the faults were completely isolable. Our results show that MLLMs can be used for the generation of models for fault diagnosis, but so far MLLMs are not reliable enough to truly automate model creation. Instead, we still need an expert to check and evaluate the generated models. The main weakness we identified is that the bad performance mainly stems from the imprecise detection of connections between elements and sensor placements (i.e. the image interpretation by the MLLM). However, we think that the image recognition feature of MLLMs will strongly improve in the future and scalable divide-and-conquer approaches will emerge. Future work should focus on a more reliable method to extract the system structure of complex systems, evaluating the approach with other MLLMs, and trying more assumptions about the system behaviour,

References

1   Luis A Aguirre. An introduction to nonlinear system identification. In *Lectures on Nonlinear Dynamics*, pages 133–154. Springer, 2023.

2   Lukas Schulze Balhorn, Marc Caballero, and Artur M Schweidtmann. Toward autocorrection of chemical process flowsheets using large language models. *arXiv preprint arXiv:2312.02873*, 2023. `doi:10.48550/arXiv.2312.02873`.

3   Kaja Balzereit, Alexander Diedrich, Jonas Ginster, Stefan Windmann, and Oliver Niggemann. An ensemble of benchmarks for the evaluation of ai methods for fault handling in cpps. In *19th IEEE International Conference on Industrial Informatics*, November 2021.

4   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.

5   Cody James Christopher and Alban Grastien. Critical observations in model-based diagnosis. *Artificial Intelligence*, page 104116, 2024. `doi:10.1016/J.ARTINT.2024.104116`.

6   Deutsches Institut für Normung e.V. (DIN). DIN EN ISO 10628-2: Flow diagrams for process plants - part 2: Graphical symbols. DIN standard, DIN, 2012.

7   Alexander Diedrich, Lukas Moddemann, and Oliver Niggemann. Learning system descriptions for cyber-physical systems. In *Proceedings of 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, 2024.

8   Marius-Constantin Dinu, Claudiu Leoveanu-Condrei, Markus Holzleitner, Werner Zellinger, and Sepp Hochreiter. Symbolicai: A framework for logic-based approaches combining generative models and solvers. *arXiv preprint arXiv:2402.00854*, 2024. `doi:10.48550/arXiv.2402.00854`.

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

10   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.

11   Constantin Hildebrandt, Sebastian Törsleff, Birte Caesar, and Alexander Fay. Ontology building for cyber-physical systems: A domain expert-centric approach. In *2018 IEEE 14th international conference on automation science and engineering (CASE)*, pages 1079–1086. IEEE, 2018. `doi:10.1109/COASE.2018.8560465`.

12   Edwin Hirtreiter, Lukas Schulze Balhorn, and Artur M Schweidtmann. Toward automatic generation of control structures for process flow diagrams with large language models. *AIChE Journal*, 70(1):e18259, 2024.

13   Sungmin Kang, Gabin An, and Shin Yoo. A preliminary evaluation of llm-based fault localization. *arXiv preprint arXiv:2308.05487*, 2023. `doi:10.48550/arXiv.2308.05487`.

14   Shota Kato, Chunpu Zhang, and Manabu Kano. Simple algorithm for judging equivalence of differential-algebraic equation systems. *Scientific reports*, 13(1):11534, 2023.

15   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`.

16   Shouvik Mani, Michael A. Haddad, Dan Constantini, Willy Douhard, Qiwei Li, and Louis Poirier. Automatic Digitization of Engineering Diagrams using Deep Learning and Graph Search. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 673–679. IEEE, June 2020. `doi:10.1109/CVPRW50498.2020.00096`.

17   Silke Merkelbach, Alexander Diedrich, Sebastian Von Enzberg, Oliver Niggemann, and Roman Dumitrescu. Towards the generation of models for fault diagnosis of cps using vqa models. *ML4CPS 2024 – Machine Learning for Cyber Physical Systems Conference*, 2014.

18   Lukas Moddemann, Henrik Sebastian Steude, Alexander Diedrich, and Oliver Niggemann. Discret2di - deep learning based discretization for model-based diagnosis. In *Proceedings of 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, 2024.

19   Oluwatosin Ogundare, Gustavo Quiros Araya, Ioannis Akrotirianakis, and Ankit Shukla. Resiliency analysis of llm generated models for industrial automation. *arXiv preprint arXiv:2308.12129*, 2023. `doi:10.48550/arXiv.2308.12129`.

20   OpenAI. Chatgpt 4 vision preview. version gpt-4-1106-vision-preview. url: https://platform.openai.com/docs/api-reference, 2023.

21   LIU Peifeng, Lu Qian, Xingwei Zhao, and Bo Tao. Joint knowledge graph and large language model for fault diagnosis and its application in aviation assembly. *IEEE Transactions on Industrial Informatics*, 2024.

22   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.

23   Swantje Plambeck, Aaron Bracht, Nemanja Hranisavljevic, and Goerschwin Fey. Famos–fast model learning for hybrid cyber-physical systems using decision trees. In *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2024. `doi:10.1145/3641513.3650131`.

24   Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. Llm-sr: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024. `doi:10.48550/arXiv.2404.18400`.

25   Arka Sinha, Johannes Bayer, and Syed Saqib Bukhari. Table localization and field value extraction in piping and instrumentation diagram images. In *2019 international conference on document analysis and recognition workshops (ICDARW)*, volume 1, pages 26–31. IEEE, 2019. `doi:10.1109/ICDARW.2019.00010`.

26   Louise Travé-Massuyès. Bridging control and artificial intelligence theories for diagnosis: A survey. *Engineering Applications of Artificial Intelligence*, 27:1–16, 2014. `doi:10.1016/J.ENGAPPAI.2013.09.018`.

27   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`.

28   Yonghao Wu, Zheng Li, Jie M Zhang, Mike Papadakis, Mark Harman, and Yong Liu. Large language models in fault localisation. *arXiv preprint arXiv:2308.15276*, 2023. `doi:10.48550/arXiv.2308.15276`.