"Where is My Troubleshooting Procedure?": Studying the Potential of RAG in Assisting Failure Resolution of Large Cyber-Physical System

Maria Teresa Rossi
University of Milano - Bicocca
Milan, Italy
Gran Sasso Science Institute
L'Aquila, Italy
maria.rossi@unimib.it

Leonardo Mariani
University of Milano - Bicocca
Milan, Italy
leonardo.mariani@unimib.it

Oliviero Riganelli
University of Milano - Bicocca
Milan, Italy
oliviero.riganelli@unimib.it

Giuseppe Filomento
University of Milano - Bicocca
Milan, Italy
g.filomneo@campus.unimib.it

Danilo Giannone
University of Milano - Bicocca
Milan, Italy
d.giannone2@campus.unimib.it

Paolo Gavazzo
University of Milano - Bicocca
Milan, Italy
p.gavazzo@campus.unimib.it

Abstract

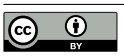
In today's complex industrial environments, operators must often navigate through extensive technical manuals to identify troubleshooting procedures that may help react to some observed failure symptoms. These manuals, written in natural language, describe many steps in detail. Unfortunately, the number, magnitude, and articulation of these descriptions can significantly slow down and complicate the retrieval of the correct procedure during critical incidents. Interestingly, Retrieval Augmented Generation (RAG) enables the development of tools based on conversational interfaces that can assist operators in their retrieval tasks, improving their capability to respond to incidents.

This paper presents the results of a set of experiments that derive from the analysis of the troubleshooting procedures available in Fincantieri, a large international company developing complex naval cyber-physical systems. Results show that RAG can assist operators in reacting promptly to failure symptoms, although specific measures have to be taken into consideration to cross-validate recommendations before actuating them.

Keywords

Troubleshooting, RAG, CPS, workarounds, manuals.

ACM Reference Format:
Maria Teresa Rossi, Leonardo Mariani, Oliviero Riganelli, Giuseppe Filomento, Danilo Giannone, and Paolo Gavazzo. 2026. "Where is My Troubleshooting Procedure?": Studying the Potential of RAG in Assisting Failure Resolution of Large Cyber-Physical System. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-SEIP '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3786583.3786890



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICSE-SEIP '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2426-8/26/04
https://doi.org/10.1145/3786583.3786890

1 Introduction

Large industrial Cyber-Physical Systems (CPS) consist of many heterogeneous components (e.g., software, hardware, and mechanical components) that have to interact properly to guarantee the intended functionality [3]. However, their inherent complexity often leads to unexpected behaviors or failures, and operators have to intervene promptly to solve or work around issues [5].

Operating in a timely and proper manner in response to misbehaviors can be extremely challenging. In fact, operators may have to carefully pair failure symptoms with many different procedures that can potentially be actuated. For instance, the industrial system we considered in this paper is operated by Fincantieri, our partner in the ATOS project (https://sites.google.com/unimib.it/atos/home), and involves over 100 manuals with hundreds of procedures described in each of them. Depending on the symptoms, there might be hundreds of troubleshooting procedures that operators may potentially activate to address an observed problem.

This problem is exacerbated by the fact that many failure symptoms are common across multiple troubleshooting procedures. For example, in the troubleshooting procedures we experimented with, a loose or uneven pipe connection may occur in both the installation and the resolution of pressure issues of the fire prevention system. Similarly, drips or leaks are present in procedures related to both valve malfunction and loose connections.

Additionally, these procedures are documented in natural language, lacking a formal machine-processable specification. For instance, the troubleshooting procedures used in our experiments are documented as XML files, where the XML elements are used to distinguish the trigger of a troubleshooting procedure (i.e., the set of failure symptoms that justify the execution of the procedure) and the description of the procedure itself. However, all descriptions are in plain natural language. Effectively browsing and searching within this corpus of information can be challenging, and even the most experienced operators may react slowly or incorrectly due to the difficulty of this task.

These challenges are not unique to Fincantieri. Troubleshooting manuals of large CPS are often documented with natural language documents that are hard to parse and process [8, 18, 21].

An intuitive and useful option for operators is using natural language interfaces to request the appropriate procedure with a natural language description of the observed failure symptoms. For instance, operators may want to simply ask questions such as "*What should I do if the pressure of subsystem X is higher than Y and the temperature alarm has been raised for subsystem Y?*".

So far, research has mostly focused on the design of repair techniques [12, 28] and automatic workarounds [6], with limited attention on the design of semi-automatic solutions that can yet improve the capability to respond to system failures. With the development of Large Language Models (LLMs), natural language interfaces are gaining popularity [29]. In particular, the *Retrieval-Augmented Generation (RAG)* architecture [20] can promisingly leverage *semantic distances* and *LLM*s to *search & retrieve* information from a *knowledge base* populated with documents (e.g., the documents with troubleshooting procedures) that must be used to answer the asked questions (e.g., questions about how to react to failures). In [24], we report preliminary results of a RAG-based system that automatically generates prompts from anomaly data and retrieves relevant troubleshooting procedures from system documentation.

This appealing option is not, however, free of challenges and perils. Indeed, responses are useful only as long as they are accurate. To understand the feasibility and effectiveness of this solution, *we empirically studied* to what extent RAG can be used by operators to efficiently *retrieve* troubleshooting procedures from natural language queries. To achieve this goal, we investigated five research questions that overall required the assessment of queries.

RQ1-Accuracy: *To what extent can a RAG recommend accurate troubleshooting instructions?* With this research question, we study both the capability of retrieving the correct troubleshooting procedures from the knowledge base and the capability of formulating an adequate natural language description of the operations that the operator must complete to respond to an incident.

RQ2-Sensitivity: *To what extent is the accuracy of the RAG dependent on how the questions are formulated?* RQ2 studies how the level of agreement between the content of the question and the content of the knowledge base impacts the accuracy of the results. In fact, questions might be formulated imprecisely, or not using the same terms present in the knowledge base. This RQ assesses its impact, both in the retrieval and in the formulation of the response.

RQ3-Derivation: *To what extent is the RAG able to derive undocumented troubleshooting procedures?* This RQ investigates whether the generative capabilities of the LLMs can be useful to guess procedures that are not explicitly documented in the KB, supporting operators towards the resolution of undocumented problems.

RQ4-Qualitative Assessment *To what extent do the results produced by the best-performing configurations reflect accurate and complete troubleshooting procedures?* This RQ complements RQ1, RQ2, and RQ3, which, because of their scale, rely on automatic and potentially imprecise evaluation methods, with a precise assessment based on the manual inspection of the responses generated by the best-performing configuration. The goal is to report and discuss the results qualitatively, identifying whether the returned procedures include all the troubleshooting steps or introduce inaccuracies.

RQ5-Performance: *How quickly can the RAG respond to the questions?* RQ5 studies how quickly RAG can respond to the users' questions, to investigate their suitability as tools available to operators to handle incidents, as soon as they occur.

Our results reveal that a RAG can *quickly suggest useful procedures*, even *accommodating for some imprecision* on the way questions are formulated by operators. On the other hand, the sources used by a RAG to generate a response *must always be returned* together with the answer, so that operators can cross-validate recommendations, if necessary. Sometimes, the RAG has also been able to *derive undocumented procedures* from the information stored in the knowledge base, helping operators with unexpected scenarios. From our experiments, a *trade-off between speed and accuracy* of answers in relation to model size emerges. Indeed, smaller language models offer significant advantages in terms of computational costs, while larger models foster accuracy of the answers by decreasing response times. The encouraging results pave the way to more research in the area, to understand how emergencies can be addressed more efficiently with the help of LLMs, especially in the frequent situation where the procedures are encoded in natural language. Our lesson learned identifies several possible areas of improvement for future research on this subject.

2 Natural Language Troubleshooting Procedures

Technical manuals are essential for diagnosing and resolving system failures. However, quickly identifying the correct troubleshooting procedure within a large corpus of documents can be difficult and time-consuming. Even the most experienced operators are not used to failures, which occur rarely, and thus have to rely on the troubleshooting procedures documented in manuals to react to them.

Currently, operators manually search through these documents, primarily using the *search* feature available in the editors. They must first locate the appropriate document and then browse or search within it to find a suitable troubleshooting procedure. This process is inefficient and requires familiarity with the manuals, increasing the risk of errors, such as missing the correct procedure or selecting the wrong one.

The manuals available in Fincantieri are similar to the manuals that can be found in many other contexts: they contain natural language descriptions of symptoms and related troubleshooting procedures. Table 1 shows an example of the key items reported in the manuals. Specifically, for each *failure symptom*, a list of *possible causes* and their corresponding *troubleshooting actions* are provided. For instance, we reported procedures *11-A* and *11-B* that are related to the same failure symptoms and that propose two different troubleshooting procedures depending on the possible cause of the failure. Instead, the procedure *8-A* comes from a different document, but it is related to a failure symptom dealing with a component called *E/pump* as for the other two procedures. This exemplifies how a trivial search for symptom names would lead to the selection of multiple alternative procedures that could be activated, with little clue for the actual choice.

3 Troubleshooting with RAG

A RAG allows users to ask LLMs natural language questions that need information stored in a Knowledge Base (KB) to be answered. The answers of the RAG shall consist of procedures that are then

ID	Failure Symptom	Possible Cause	Troubleshooting Action
11-A	During operation of E/pump and its desalinator on the automation consoles, the vibration values for one or more accelerometers are not present	The signal cable from the accelerometer panel to the ship's automation system is not properly connected or is damaged.	Check that the connection cable from the accelerometer panel to the automation system is correctly connected and not damaged. Restore if necessary
11-B	During operation of E/pump and its desalinator on the automation consoles, the vibration values for one or more accelerometers are not present	Accelerometer is faulty	Replace the accelerometer
8-A	Pressing the E/P start button on the desalinator control display does not start the E/P of the E/Pump Module.	Desalinator panel touch screen failure.	Replace faulty touch screen

Table 1: Example of the key items present in the troubleshooting manuals.

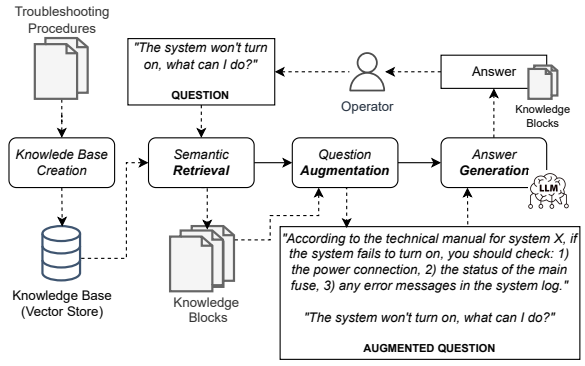


Figure 1: Architecture of the proposed RAG for the recommendation of troubleshooting procedures.

suggested to the operators. Note that operators are not supposed to apply procedures blindly. On the contrary, they use their skills to decide about their suitability. The RAG supports cross-validation of the responses, since each answer is associated with the items of the KB that motivate the response, offering the operators means to verify the answer by accessing the sources.

Figure 1 illustrates how we use the RAG to retrieve troubleshooting procedures of a naval CPS. The process starts with the step *knowledge base creation* that parses the *Troubleshooting Procedures* and embeds them into a *Knowledge Base*, accessible to the *LLM*.

The knowledge base is populated by first dividing the input documents into chunks (i.e., non-overlapping sections of text of fixed or variable length tagged with indices to facilitate information retrieval). These chunks serve as the basic units of knowledge accessible to the LLM and are then uploaded into the knowledge base [10].

Each chunk is then mapped into an embedding to allow for the semantic retrieval of the information; that is, the embedding maps chunks that are semantically related in nearby regions of the space, enabling effective similarity-based retrieval. Once the chunks have been uploaded, the knowledge base is ready to be used.

The user interested in retrieving the procedure to be applied in reaction to an issue asks a question that is submitted to the RAG. Before the question is routed to the LLM, the *knowledge retrieval* looks for chunks that are semantically close to the question according to the cosine similarity [26]. The intuition is that text (i.e., chunks) semantically close to the text in the question is likely useful to answer the question (e.g., it includes information about the problem described in the question and the possible reaction to that problem).

In principle, semantic retrieval might already be sufficient to respond to the operators' questions: that is, operators may simply process the list of retrieved chunks one after the other. However, our experiments show that retrieval is difficult and often imprecise, due to the similarity between troubleshooting procedures, with the correct text often not reported in the first position, or not retrieved at all. The LLM used to process the output of the retrieval can, interestingly, compensate for this weakness. The retrieved chunks are automatically embedded as a preamble in the prompt of the LLM. This helps the language model to ground its answer on the retrieved information, while exploiting generative capabilities to compensate for any inaccurate retrieval.

For instance, a user's question such as *The system won't turn on, what can I do?* can be augmented with the retrieved chunks, obtaining a final augmented question that includes in the context a description such as *According to the technical manual for system X, if the system fails to turn on, you should check: 1) the power connection, 2) the status of the main fuse, 3) any error messages in the system log file.* This step is particularly important to let the LLM have access to information about the system that is malfunctioning and the specific procedures that can be actuated. In fact, this information, stored in private documents of companies, would not be otherwise available to the model.

In the *answer to the question* step, the augmented question is passed to the LLM that formulates the final response.

We designed the RAG to work with a fully local deployment, not relying on any third-party service, to preserve the intellectual property and the privacy requirements required by companies, thus not sending any information to any online service. For this reason, we have not considered the usage of online LLMs, such as ChatGPT [7] or Gemini [30], and the project material is not provided in this paper.

To simplify the integration of multiple LLM models within the RAG, we used Ollama (https://ollama.com/) as the container of our RAG architecture. We used LangChain (https://www.langchain.com/) as RAG framework and Chroma (https://www.trychroma.com/) as vector store, since they are both popular frameworks offering production-quality capabilities.

4 Methodology

To evaluate the effectiveness of RAG in supporting troubleshooting activities in large industrial Cyber-Physical Systems, we designed a set of experiments guided by five main research questions. Each question targets a specific aspect of the troubleshooting identification process:

RQ1 – Accuracy: *To what extent can a RAG produce accurate troubleshooting instructions?* This question investigates both the precision of the retrieval task and the quality of the generated response. It is further divided into:

RQ1.1 – Retrieval Accuracy: *Can a RAG retrieve troubleshooting procedures from operators' queries?* This sub-RQ specifically assesses the precision of the retrieval task of the RAG, namely the output of the *Semantic Retrieval* task in Figure 1, by measuring if and in which position the block with the correct troubleshooting procedure occurs in the set of retrieved blocks.

RQ1.2 – Response Accuracy: *Can the LLM exploit the retrieved information to accurately describe the troubleshooting procedure?* This sub-RQ studies if the LLM can properly formulate the troubleshooting procedure that must be applied to work around a problem. In fact, the role of the LLM might be either beneficial, producing an accurate description of the troubleshooting procedures from the blocks returned by the retrieval task, or deleterious, producing an inaccurate response from the retrieved blocks. Ultimately, the accuracy of the response (the answer block in Figure 1) determines the accuracy of the whole RAG system. To scale the evaluation to a large set of responses obtained with different questions, models, configuration, and repetitions, we adopt an LLMs-as-jury strategy, presented in Section 4.3.

RQ2 – Sensitivity: *To what extent is the accuracy of the RAG dependent on how the questions are formulated?* This question studies how the level of agreement between the content of the question and the content of the knowledge base impacts the accuracy of the results. In fact, questions might be formulated imprecisely, or not using the same terms present in the knowledge base. This RQ assesses its impact, both in the retrieval and in the formulation of the response. It is split into:

RQ2.1 – Retrieval Sensitivity: *How sensitive is the retrieval to the way questions are formulated?* This sub-RQ studies the sensitivity of the retrieval task to the input questions. We assess the results using the same metrics used for RQ1.1.

RQ2.2 – Response Sensitivity: *How sensitive is the LLM to the way questions are formulated?* This sub-RQ studies the sensitivity of the final response to the input question. We assess the results using the same metrics used for RQ1.2.

RQ3 – Derivation: *To what extent is the RAG able to derive undocumented troubleshooting procedures?* LLMs have generative capabilities, that is, they can suggest procedures that are not necessarily in the knowledge base, but they could potentially derive new procedures, exploiting both the knowledge present in the knowledge base and the knowledge intrinsically stored in the LLMs themselves. This sub-RQ assesses the capability of the RAG to recommend undocumented troubleshooting procedures that have not been anticipated by the domain experts, and thus are not present in the manuals. To measure how diverse the answers are when the support of the KB is lacking, we compute the BLEU [13] metric between the expected and given response.

RQ4-Qualitative Assessment *To what extent do the results produced by the best-performing configurations reflect accurate and complete troubleshooting procedures?* To scale the evaluation to a large number of queries, RQ1-RQ3 use automatic metrics. However, these metrics only provide an approximation of the real accuracy of the answers. To gain a more accurate view of the results, we answer RQ4 by manually inspecting the outputs generated by the best-performing configurations in RQ1-3. The goal is to report and discuss the results qualitatively, identifying whether the returned procedures include all the troubleshooting steps or introduce inaccuracies.

RQ5 - Performance: *How quickly can the RAG respond to questions?* This research question investigates the efficiency of the RAG, to assess the possibility of establishing actual live conversations between the operator and the RAG system.

4.1 Questions Asked

To sample a range of different cases, we randomly selected 25 distinct sets of symptoms and corresponding troubleshooting procedures from the available documents. For each selected set of symptoms, we defined four questions, all expecting the same troubleshooting procedure as an answer. Each question is characterized by a different level of precision in the included statements and the possible inclusion of contextual information. In total, we thus considered 100 questions.

In particular, each question consists of two parts. The initial part is the question itself. It is a sentence that begins with "*What should I do if...*" and continues with a description of the observed symptoms. The description could be either using the same terminology used in the troubleshooting (accurate question) or using an imprecise terminology (inaccurate question). To obtain the inaccurate question, we substitute the technical terms used in the documentation with more generic synonyms. The second part of the question is a reference to the document where the searched procedure is present. The second part is included for questions with context, and skipped for questions without contexts. We thus ask questions like *"What should I do if the red indicator 'LOW PUMP PRESSURE' is on?"*.

The four types of questions capture four different use cases: the *accurate questions with contexts* correspond to an operator with good knowledge of the available documentation who knows which document describes the searched troubleshooting procedure; the *accurate questions without context* correspond to an operator who knows the domain terminology, but does not have the exact knowledge of the many documents available; the *inaccurate questions with context* correspond to an operator who does not know the domain, but knows the documentation (e.g., a beginner who is not yet confident in the domain but has worked on the documentation long enough to know where to search); and the *inaccurate question without context* corresponds to an operator who is not able to use the exact domain terminology used in the documentation and cannot provide a priori reference to the documentation that includes the answer to the question.

To specifically answer RQ3, we need to assess questions that have no direct answer in the KB. To this end, we initialized the

knowledge base with three different configurations, which correspond to three cases with gradually less information stored. Given a question Q about some failure symptoms S that must be handled with the troubleshooting procedure T, the *no response* configuration corresponds to the case in which S occurs in the KB, but not T; the *no entry* configuration corresponds to the case in which both S and T do not occur in the KB; and the *no KB* configuration corresponds to the case the KB is empty, and thus even the symptoms and troubleshooting procedures similar to S and T are missing. We generate all these configurations for the questions studied in the paper.

4.2 Setup

Environment. For the experiments, we run the application on a virtual machine via Azure (a *Standard NC4as T4 v3*) that use an Nvidia Tesla T4 GPU, with 16GB of VRAM, and an AMD EPYC 7V12 CPU.

Model	Parameters	Correct	Correct Exceeding Expectations	Incorrect or Incomplete
Llama	3B	2	0	2
Mistral	7B	1	1	2
Qwen	7B	1	2	1
Phi	3B	0	0	4
Orca-mini	3B	0	0	4
Zephyr	7B	0	0	4

Table 2: Results of the early assessment of the LLMs.

LLMs. To identify the LLMs to be used for the experiment, we performed an early assessment of six LLMs of different sizes and nature, as summarized in columns *model* and *parameters* of Table 2. We formulated four questions, considering four different topics and phrasing the questions differently, to explore how variations in structure, tone, and contextual detail may influence the clarity and specificity of the required troubleshooting procedure. Since we formulated the questions based on the available troubleshooting document, we also know the answer to each question.

To assess the answers provided by the LLMs to the questions, each answer has been independently inspected by two authors, distinguishing *correct answers*, which provide instructions equivalent to the troubleshooting procedures that must be applied to the considered case; *correct answers that exceed expectations*, which add relevant and accurate information extracted from the KB to the correct answer; and *incorrect or incomplete answers*, which do not fully describe the troubleshooting procedures that must be applied to the considered case. If the two inspectors disagree on the assessment of an answer, they analyze the answer and discuss until they find an agreement.

Table 2 summarizes the results for this initial assessment. We can state that Llama[1], Mistral[2], and Qwen[3] clearly provided the highest number of correct answers and the least number of incorrect answers. While [4], Orca-mini[5] and Zephyr[6] struggled with the considered task, only providing inaccurate answers.

Based on these results, we select for the larger experimental campaign described in the next section Qwen (the best performing model), Mistral (the second best performing model), and Llama (the best non 7B parameters model).

LLMs' Configuration. Regarding the temperature parameter, which affects the creativity of the model, we used the default value. Regarding top-p (i.e., nucleus sampling [14]), which is a decoding parameter in the range $[0, 1]$ that controls the randomness of the language model's output by limiting the token sampling to the smallest possible set of tokens whose cumulative probability exceeds the specified threshold, we considered three values: 0.2, 0.5, 0.9. The choice of the values derives from the goal of sampling the range of possible values, without hitting extreme cases (e.g., top-p equals 0 or 1).

For the chunk size (i.e., the size of the retrieved blocks), we considered three values: 400, 800, and 1000. The chunk sizes refer to the number of characters into which the documents are split before being embedded and indexed. We chose these values to study how different granularity levels in the retrieval process affect the quality of the generated answers.

The retrieval returns four blocks that are used to augment the question. Four blocks represent a good tradeoff between excessive information retrieved and the probability of retrieving the correct block, as confirmed in our experiments.

Experiment space. We study how all the parameters interact, as summarized in Figure 2, for a total of 135,000 questions assessed in this study: three LLMs, times three values of top-p, times three chunk sizes, times 100 questions, times 25 questions per category, and 20 repetitions with the full KB and 10 for the three cases with partial or no KB.

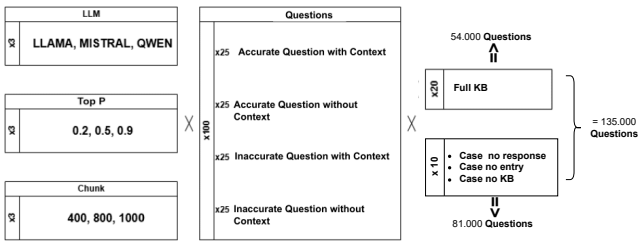


Figure 2: Total number of questions submitted to the RAG in our evaluation.

We use the answers to the Accurate Questions with context to answer RQ1. We use the answers to the alternative formulation of the questions to answer RQ2. We use the questions without a direct answer in the KB to answer RQ3.

[1]https://www.llama.com/
[2]https://mistral.ai/
[3]https://chat.qwen.ai/

[4]https://learn.michttps://huggingface.co/HuggingFaceH4/zephyr-7b-betarosoft.com/it-it/windows/ai/models/get-started-models-genai
[5]https://huggingface.co/pankajmathur/orca_mini_3b
[6]https://huggingface.co/HuggingFaceH4/zephyr-7b-beta

Judge	Gemma	Granite	Hermes	Avg	Mean
Correlation	0.47	0.48	0.34	0.59	**0.69**

Table 3: Assessment of the judges.

	p1	p2	p3	p4	p[1-4]	Never found
Retrieval: accurate question	34%	4%	1%	2%	41%	5/25
Retrieval: inaccurate question	10%	3%	2%	4%	19%	7/25

Table 4: Probability of retrieving the correct block for accurate and inaccurate questions.

4.3 Evaluation With the Judges

Large-scale experimentation requires automated assessment of the answers. For each question we ask, we know by design what the correct answer extracted from the available documentation is. However, the same answer can be stated in many different but equivalent ways. We thus adopted an LLM-as-a-jury paradigm [15] and selected three LLMs, distinct from the ones used in the RAG to avoid any bias, as judges for the answers. The three LLMs are Gemma (https://ai.google.dev/gemma?hl=it), Granite (https://www.ibm.com/granite) and Hermes (https://nousresearch.com/hermes3/). We selected these models since they perform well in evaluation and classification tasks, and their diversity in model architecture and training data ensures a more robust and unbiased judgment.

Each judge LLM is queried using the following template: "The text within the brackets is referred to as the 'reference' (" + [reference] + "). The text after the exclamation mark symbol is referred to as the 'comparison'. Your task is to assess the similarity between the content of the 'reference' and the content of the 'comparison', answering only with a number between '1' and '10' inclusive. If you answer '1', the content of 'reference' is completely different from the content of 'comparison', whereas if you answer is '10' the content of the 'reference' and the content of the 'comparison' are extremely similar. The check you have to make is not about the form, but about the content. It is not necessary that the exact same words are used, but the meaning must be the same. !" + [LLM response].

To decide how to employ the LLM judges, we first performed a small-scale quality control experiment to assess the judges in isolation and as an ensemble. In particular, we select 33 questions of various complexity and length, and submit the questions and the responses formulated by the RAG to the three LLM judges.

To produce the ground truth, two of the authors manually assessed the same set of responses with a score within the same scale used by the judges and then compute the correlation between the judges' scores and the ground truth scores to determine the best combination. We considered the three judges in isolation, and the average and mean scores of the responses produced by the three judges as a possible option. Table 3 summarizes the results.

We can notice that while the individual judges are relatively effective in judging the responses, the median answer of the three judges is the best performing option with a good level of correlation with the human judgment. Based on these results, we will assess the responses obtained with our large-scale evaluation presented in the next section using the mean value.

5 Results

This section describes the results obtained for each research question and discusses the potential threats to the validity of the results.

5.1 RQ1 - Accuracy

This RQ investigates the effectiveness of both the intermediate retrieval task and the actual RAG's response.

5.1.1 RQ1.1 - Retrieval Accuracy. To assess the accuracy of the retrieval process we measure how often the block that describes the troubleshooting procedure that must be actuated to respond to the question occurs in any of the first four blocks that are retrieved. In particular, Table 4 row *Retrieval (accurate questions)* reports the probability that the correct block appears in any of the first four blocks (columns *p1, p2, p3, p4*), the cumulative probability of appearing in any of these blocks (column *p[1-4]*), and the number of questions for which the retrieval systematically failed, never retrieving the correct block (column *never found*).

In most cases, the correct block, if retrieved, is in the first position (34% probability). Still, the other three positions are also often relevant, for a cumulative probability of retrieving the correct block equals 41%. Although not always retrieving the correct description, when the retrieval fails, it can still retrieve related troubleshooting procedures that might help the LLM, and then the operator, to formulate a feasible troubleshooting plan, as confirmed by the results in the next section. Further, in 80% of the questions, the right block is retrieved at least in some of the repetitions, while for 5 questions (20%) the right block could never be retrieved.

5.1.2 RQ1.2 - Response Accuracy. Figure 3 reports the results obtained with the three LLMs when the accurate question with context is used. The color is representative of the model used, and the shade of the color differentiates configurations.

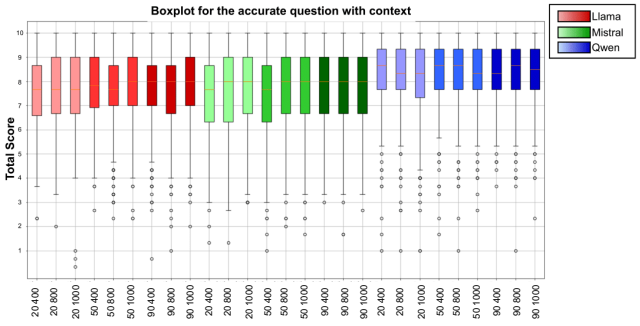


Figure 3: Boxplot representing the assessment of the answers to the accurate questions with context.

We can notice how the models performed generally well according to the jury, with the boxplots all shown in the top part of the diagram. In particular, Qwen performed better than the others, both

in terms of consistency of the results (shorter boxes) and in terms of the top scores reached.

The configuration of the LLM (i.e., top-p and the chunk size) does not impact the results, suggesting that its choice is not particularly critical for the studied task.

5.2 RQ2 - Sensitivity

This RQ investigates the sensitivity to the accuracy of questions of both the intermediate retrieval task and the actual response.

5.2.1 RQ2.1 - Retrieval Sensitivity. Table 4 row *Retrieval (inaccurate questions)* reports the probability that the correct block appears in any of the first four blocks (columns *p1, p2, p3, p4*), the cumulative probability of appearing in any of these blocks (column *p[1-4]*), and the number of questions for which the retrieval systematically failed, never retrieving the correct block (column *never found*), when an inaccurate question is used.

The correct block occurs in the first position with only a 10% chance. The cumulative probability of retrieving the correct block in any of the first four positions is 19%, less than half of the probability observed when accurate questions are used. The use of synonyms and changes to the used verbs, due to the many similar procedures stored in the KB, immediately complicated the retrieval task.

This result remarks how using the domain terminology as used in the troubleshooting procedure is important for the output of the retrieval task. Although the retrieval task is complicated by the inaccurate questions, the LLM may still have the chance of exploiting its knowledge and its generative capabilities to produce useful recommendations.

5.2.2 RQ2.2 - Response Sensitivity. Figures 4, 5, and 6 show the results for inaccurate questions with context, accurate questions without context, and inaccurate questions without context, respectively.

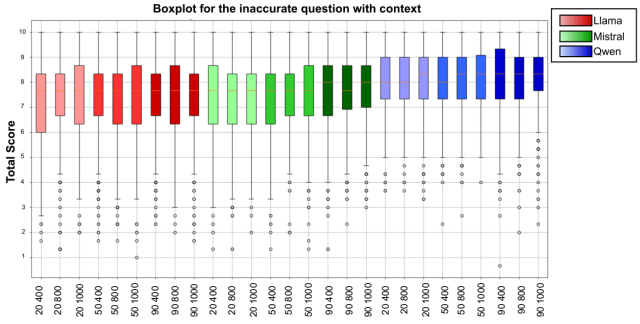


Figure 4: Boxplot representing the assessment of the answers to inaccurate questions with context.

The box plots show two main results. First, results do not strongly degrade with the inaccuracy of the questions and the lack of context, according to the jury. Of course, both a nicely formulated question and a reference to the context seem useful to obtain a correct answer, but neither of the two looks indispensable. Second, the better performance of Qwen is consistently confirmed across all configurations. This is likely because Qwen is trained on a diverse

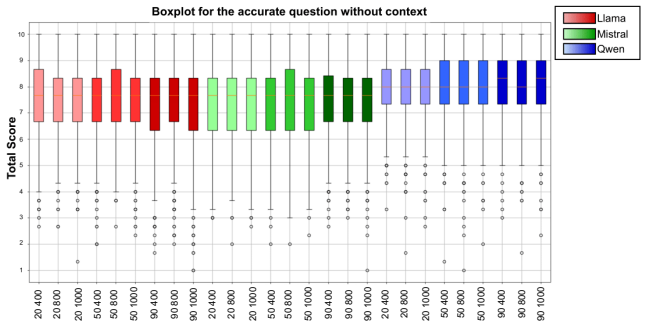


Figure 5: Boxplot representing the assessment of the answers to questions without context.

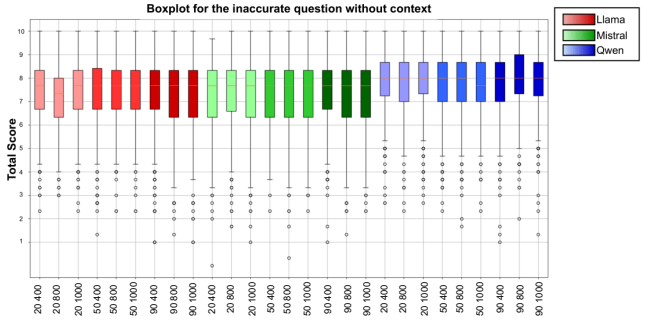


Figure 6: Boxplot representing the assessment of the answers to inaccurate questions with no context.

multilingual corpus, and it demonstrates strength in instruction-following tasks, which makes it more robust to poorly formulated questions and context sparsity.

It is interesting to note that Llama, although performing worse than Qwen, still performs well while using a smaller model. In fact, Llama is a 3B model, while Qwen is a 7B model. In case using a small model is important (e.g., due to resources on the available constraints), Llama might be a relevant option.

Again, the choice of the configuration has little impact on the results.

5.3 RQ3 - Derivation

This RQ investigates whether the RAG can be used to derive troubleshooting procedures that are not already present in the KB, exploiting the generative capability of the LLM. To do this, we measure the loss of similarity between the answers obtained with the full KB and the answers obtained with the three alternative configurations, for all the considered LLMs. We use BLEU [13] as a measure of similarity since it can capture the semantic closeness of two paragraphs.

Table 5: Results with incomplete KB

Scenario	no response	no entry	no KB
Δ% BLEU (avg)	-9.68%	-53.30%	-85.83%

Table 5 shows the average loss across all questions. When only the response is missing from the KB, the LLM can still suggest the correct troubleshooting procedure in the vast majority of cases, with a loss smaller than 10%. However, once the full entry with symptoms and the troubleshooting procedure is removed, the number of correct responses has a significant drop. Yet some questions could be answered by reusing and adapting troubleshooting procedures defined for other components and/or cases. This is a relevant capability that is useful to help operators address unspecified situations that require a timely resolution.

The drop is dramatic when the KB is empty, which, not surprisingly, indicates that the RAG is necessary to recommend troubleshooting procedures.

We further analyze these cases for the best configuration in the next section.

5.4 RQ4-Qualitative Assessment

To assess the results qualitatively, we inspected the outputs produced by Qwen, the best-performing model, in the context of RQ1 (accurate questions), RQ2 (alternative questions), and RQ3 (questions with no explicit answer in the KB), and analyzed the likely reasons for the mistakes. In particular, we categorized answers in three cases: *all steps*, which are answers that include all the steps that must be executed to complete the troubleshooting procedures; *partial*, which are answers that include only a subset of the steps that must be actuated; and *wrong*, which are answers unrelated to the actual procedure. Table 6 reports the results. For the all steps case, we also indicate in parentheses the percentage of cases where extra steps appear in the reported answer.

When asking questions that have a response in the KB (row full KB), the qualitative analysis reveals that using proper questions and context is important to obtain a high rate of useful responses (only 8% of the responses were wrong). The percentage of wrong responses grows to 36% when either the question is inaccurate or the context is not provided. Finally, more than half of the responses are wrong for inaccurate questions without context. Results also show that the presence of the sources used to formulate the answer is important for validating the answer and resolving any imprecision, for instance, to retrieve the missing steps for the incomplete answers or filter out the extra steps.

When considering scenarios where operators ask questions about troubleshooting procedures that have not been documented yet, not surprisingly, the accuracy of the responses quickly drops. However, it is an interesting result that for many cases (nearly one case out of two), the RAG was still able to output a useful response, sometimes a fully accurate procedure. In fact, if either the question is accurate or the context is provided, the range of wrong responses is from 52% to 68%. The extreme scenario where the question is inaccurate, the context is not provided, and the response is not in the KB is definitely too hard for the RAG, with 72% of the responses being wrong and 20% incomplete. Although the RAG cannot be systematically used to derive troubleshooting procedures that are not in the KB, it can indeed be a valid tool to provide initial suggestions to the operator who has to react quickly to any problem.

We also inspected the responses to identify the elements that likely influenced the results.

When inspecting the answers generated for questions in RQ1, we noticed that accuracy is lower for the questions that show terminological ambiguity or include complex multi-condition formulations. Questions such as "*What should I do if I detect compressed air leaks on pipelines?*" or "*What should I do if the assembly of a component appears loose or uneven?*" use vague or generic terms like "pipelines" and "component" that may confuse the RAG. As a result, the system may fail to retrieve the most relevant chunks of information or retrieve unrelated content that does not fully address the issue. Similarly, complex questions that include multiple conditions can introduce difficulties in both retrieval and generation. For instance, questions such as "*What should I do if the compressor has been turned off and the error message 'Oil' is displayed on the compressor control?*" envisage the understanding of the interplay between two events, namely, the shutdown of the compressor and the appearance of a specific error message, not handling the two events has two independent failure triggering conditions. When the relevant information is scattered or only implicitly connected, the system may struggle to synthesize a coherent and correct response. These issues highlight the importance of well-structured documentation and clear, unambiguous question phrasing when building and evaluating RAG systems.

When inspecting the answers generated for questions in RQ2, similarly to observations for questions in RQ1, a major problem lies in the use of ambiguous terminology. For example, in the question "*What should I do if the CIP pump leaks?*", the verb "leaks" is too generic and does not clearly indicate whether the issue concerns the mechanical seals, pipe fittings, or some other components. Here, the use of a vague term has likely negatively affected the capability of the model. Furthermore, the RAG has more difficulty in responding to questions that combine multiple conditions. A question such as "*What should I do if, while the E/P is running, the pump caps are leaking?*" blends the operational status of the equipment with a symptom of failure. In such cases, the RAG system may not always find a suitable chunk that responds to the described scenario. Unclear phrasing, generic conditions, or overly complex formulations limit the effectiveness of the semantic retrieval, leading to less accurate answers.

When inspecting the answers generated for questions in RQ3, we noticed the LLM is often able to adapt troubleshooting procedures that are strongly similar to the missing one, often compensating for the lack of information in the KB. Errors are mostly due to erroneous procedure reuses, erroneous combinations of multiple procedures into a single one, or the output of generic recommendations that do not consist of any procedure that operators can apply.

5.5 RQ5 - Performance

Since operators are supposed to query the RAG during problematic situations, we measured the time required by the RAG to provide an answer to the questions formulated by users.

In Table 7, we reported the time in seconds required by the RAG to provide an answer to questions w.r.t. the considered LLMs. Specifically, we reported for every LLM (i) the average (AVG) time in seconds taken by the RAG to answer the questions considered in the advanced test; (ii) the fastest time (MIN) and (iii) the slowest (MAX) one. Looking at the reported times, Llama and Mistral can

	Accurate with context			Inaccurate with context			Accurate without context			Inaccurate without context		
	All steps	**Partial**	**Wrong**	**All steps**	**Partial**	**Wrong**	**All steps**	**Partial**	**Wrong**	**All steps**	**Partial**	**Wrong**
full KB	64% (28%)	28%	8%	48% (12%)	16%	36%	44% (20%)	20%	36%	24% (12%)	20%	56%
no response	36% (20%)	12%	52%	28% (28%)	20%	52%	16% (12%)	16%	68%	12% (12%)	16%	72%
no entry	28% (20%)	12%	60%	40% (4%)	20%	64%	36% (8%)	12%	56%	20% (0%)	20%	72%
no KB	28% (4%)	12%	60%	16% (4%)	20%	64%	32% (0%)	12%	56%	8% (4%)	20%	72%

Table 6: Results of the manual inspection of the best-configuration's results in terms of completeness.

respond faster than Qwen, producing an answer in the range from about 2 to 7 seconds. Instead, Qwen, which is the most effective model in terms of accuracy and sensitivity, is slower in producing answers, responding in an average time of about 12 seconds.

LLM	AVG	MIN	MAX
Llama	3,6 secs	2,9 secs	4,2 secs
Mistral	4,52 secs	3,5 secs	7,5 secs
Qwen	12,2 secs	10,5 secs	14,2 secs

Table 7: Time required by the RAG to provide an answer to questions w.r.t. the considered LLMs.

In a nutshell, in case the priority is obtaining responses quickly (e.g., in a few seconds), because a procedure must be immediately selected, or the answer is the basis for some follow-up steps (e.g., additional questions), Llama might be the preferred option. Instead, if obtaining accurate answers is the priority, at the cost of waiting some more time (between 9 and 10 secs in our experiments), Qwen is the best choice among the investigated LLMs.

5.6 Threats to Validity

In this section we discuss potential threats to the validity of the conducted evaluation.

Construct Validity. Using LLM-based judges to assess the accuracy of generated answers may generate a concern related to their judgment, which might reflect limitations or inconsistencies in their capabilities. To mitigate this threat, we completed an initial assessment of the judges in our context (see Section 4.3), identifying the combination that provides the best result. Further, we manually inspected the responses, producing an accurate qualitative assessment of the best-performing model, eliminating the issue of judges for the most promising configuration of the approach. Another aspect that may affect reproducibility is version drift of the used LLMs. Our evaluation based on multiple LLMs reduces the dependency of the results on a single model version.

Internal Validity. In our setup, we study the impact of three parameters (LLM, chunk size, top-p) on the results, partially considering their interactions. Further experiments would be necessary to systematically consider every possible combination of these parameters, although the evidence reported in this paper suggests that their impact on the results is very limited.

While we designed question variants to represent different levels of precision and contextual reference, there might be unintended differences in difficulty or phrasing that influence the outcome beyond what we intended to measure. To mitigate this threat, we discussed cases of questions that have been hard to answer, so that these could be taken into consideration in future research.

External Validity. Our experiments are based on a set of 25 distinct topics and 100 questions, which, despite being diverse and relevant in the studied domain, cannot represent the full variety of real-world user queries. The study in an industrially relevant context posed limitations in the range and number of cases that could be investigated. Yet our findings could be the basis for follow-up work in similar domains.

Conclusion Validity. The assignment of a numerical score to an answer provides granularity, but it might not capture all dimensions of answer quality. For this reason, the paper reports a detailed qualitative analysis of the responses. Although, we did not involve practitioners from Fincantieri in this phase due to organizational and operational constraints, we cross-validated and reviewed the interpretations to reduce bias and ensure consistency.

6 Lesson Learned

Finally, we discuss the lessons learned from our experiments.

Lesson 1 – RAG is a valid tool to suggest troubleshooting procedures, which, however, must be validated and completed by experts. RAG revealed a useful tool to provide useful recommendations that contain all the steps, or at least some of the steps, that must be actuated to resolve an incident (see row Full KB in Table 6 column *accurate with context*). As in many other contexts, operators cannot blindly trust generative AI, and the operator's intervention is required to critically revised the procedure and finally decide the steps to execute. In this respect, the set of sources attached to the responses is a valid support to quickly cross-validate the responses.

Lesson 2 – Training operators on the domain terminology might improve the capability to quickly respond to incidents. Results show that some responses might be imprecise, especially when queries are inaccurate and lack contextual information (see row Full KB in Table 6 column from second to fourth). A consequent recommendation is to train operators on the main terms that must be used when looking for troubleshooting procedures, so that they can formulate proper requests for the RAG.

Lesson 3 - Accuracy is resilient to question-inherent variations but sensitive to ambiguity. The experiments demonstrate that LLMs like Qwen often maintain good performance even when questions are poorly formulated or lack explicit context. However, terminological ambiguities and complex multi-condition questions still pose challenges, leading to decreased response accuracy. Ensuring question clarity and minimizing vagueness can significantly

improve system reliability. This issue can likely be mitigated by employing prompt engineering techniques or implementing strategies to automatically rewrite prompts.

Lesson 4 – The generative capability of RAGs may help face undocumented situations. The generative capabilities of RAGs, and LLMs in particular, allow them to infer some plausible troubleshooting steps even when procedures are missing or incomplete in the KB (see rows from second to fourth in Table 6). Although this behavior is promising for handling undocumented scenarios, it also introduces the risks of hallucination or overgeneralization, especially when questions are not properly formulated. In safety-critical environments, such responses must be carefully validated, and systems should be designed to clearly distinguish between reliably retrieved content and unreliable inferred content.

Lesson 5 – Smaller models offer speed, larger models offer accuracy. Our experiments reveal a trade-off between response time and response quality (Table 7). Smaller models like Llama respond faster and may be suitable for time-sensitive tasks, while larger models like Qwen provide more accurate and context-aware responses at the cost of increased latency. This suggests that the choice of model should be guided by the operating context, depending on whether speed or accuracy is the priority.

Lesson 6 - Semantic search in isolation is weaker than RAG. The semantic retrieval task was able to retrieve the correct block a relatively high number of times (Table 4). In our experiments, the role of the LLM was essential to derive the troubleshooting procedure, compensating for some inaccuracies in the retrieval task. In addition, the RAG offers a conversational interface that is particularly effective for operators handling critical situations.

7 Related Work

Multiple classes of approaches have been defined to address failure and failure symptoms. For instance, systems might be designed to be resilient to failures [4, 17, 19, 27], so that failures have no harmful or critical consequences especially in sensitive domains such as healthcare [2]. Other approaches, not designed for resiliency, studied how to exploit the implicit redundancy present in some systems to dynamically generate workarounds [6]. Finally, once a failure has been observed, automatic program repair techniques have been investigated to recommend fixes to developers [12, 28]. Complemental to these studies, this paper investigates the case systems cannot workaround failures automatically, but require the user intervention to react to failure symptoms.

Our work adapts the principles of Recommendation Systems for Software Engineering (RSSE) [11, 23] to a largely unexplored domain in the operational maintenance of CPSs. RSSEs are designed to overcome information overload and support users in decision-making and information-seeking activities by providing valuable information items for a specific software engineering task within a given context [23]. While recommendation systems in software engineering typically assist software developers by suggesting source code artifacts during the implementation and maintenance phases of the development lifecycle [9, 11], our system supports operators during the post-deployment phase. Instead of recommending code, it provides troubleshooting procedures extracted from technical manuals to help resolve runtime failures. This shift in focus from code to documentation addresses a recognized gap in the literature, where there are unexploited opportunities in the development of recommendation systems outside the source code domain [11, 23]. By leveraging RAG, our approach broadens the applicability of recommendation systems to support decision-making in the resolution of runtime failures within industrial CPS.

Recent advancements in troubleshooting within industrial environments have been driven by the integration of natural language processing and information retrieval techniques. These studies have explored these avenues to enhance accessibility, efficiency, and accuracy of maintenance operations [1, 22, 25].

Ren et al. [22] developed a voice-interactive fault diagnosis system for industrial robots, which demonstrates the potential of voice commands for facilitating information retrieval from extensive manuals. While they focus on voice interaction, our study enhances this by applying RAG techniques to improve the accuracy and responsiveness of troubleshooting information retrieval.

Kiangala and Wang [16] present an experimental hybrid AI chatbot that combines customized AI and generative AI to enhance human–machine interaction within factory troubleshooting scenarios under Industry 5.0. Their approach emphasizes adaptive, AI-driven dialogue systems capable of resolving complex troubleshooting queries, leading to reduced factory downtime. Our work contributes to this field by demonstrating how RAG can enhance the retrieval of relevant data, thus improving the interaction between humans and machines in industrial settings.

Su et al. [25] introduce an innovative approach to enhance maintenance and troubleshooting efficiency in aerospace applications. Their method leverages integrated information flow modeling and ontology, along with ant colony optimization, to detect faults. Unlike Y. Su et al.'s method, which relies on structured data and ontological frameworks, our approach leverages LLMs to interpret unstructured queries. This enables operators to intuitively engage with procedures without needing to construct ad hoc ontologies.

The findings presented in the work by Algeo et al. [1] emphasize the critical need for integrated maintenance strategies in industrial settings. The study uses a deep learning approach for retrieving solutions from historical technical assistance reports within predictive maintenance frameworks, significantly reducing downtime and improving operational efficiency. The integration of such technologies aligns with our research focus, as it supports the need for robust data-driven decision-making processes that can be complemented by RAG techniques to improve information retrieval.

8 Conclusions

Large industrial CPSs are complex systems that require operators to readily resolve unexpected behaviors and failures. The troubleshooting strategies that must be applied are often documented in natural language manuals that are hard and slow to search.

In this paper, we investigated the use of RAG to quickly identify the right procedure to resolve a problem. We studied this problem for the manuals documenting a large naval CPS available at Fincantieri. Results show that RAG can indeed help operators, with some caveats, such as the documentation of the sources used to produce answers. From our experience, we distilled a lesson learned that can be used as a foundation for additional work in the area.

In the future, we plan to extend our work to additional systems and to develop additional strategies to help operators using conversational interfaces to timely react to problems. This way, we want to test other CPS documentation in order to explore how RAG responds to multi-modal troubleshooting instructions.

Acknowledgments

This work was supported by the ATOS project, funded by the MUR under the PNRR- CN - HPC - ICSC program (CUP: H43C22000520001).

References

[1] Antonio L Alfeo, Mario GCA Cimino, and Gigliola Vaglini. 2021. Technological troubleshooting based on sentence embedding with deep transformers. *Journal of Intelligent Manufacturing* 32, 6 (2021), 1699–1710.

[2] Lameck Amugongo, Pietro Mascheroni, Steven Brooks, Stefan Doering, and Jan Seidel. 2025. Retrieval augmented generation for large language models in healthcare: A systematic review. *PLOS Digital Health* 4 (06 2025). doi:10.1371/journal.pdig.0000877

[3] Radhakisan Baheti and Helen Gill. 2011. Cyber-physical systems. *The impact of control technology* 12, 1 (2011), 161–166.

[4] João R. Campos, Ernesto Costa, and Marco Vieira. 2023. Online Failure Prediction Through Fault Injection and Machine Learning: Methodology and Case Study. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*. 451–461.

[5] João R Campos, Ernesto Costa, and Marco Vieira. 2025. Predicting Failures in Complex Systems. *Computer* 58, 5 (2025), 57–64.

[6] Antonio Carzaniga, Alessandra Gorla, Nicolò Perino, and Mauro Pezzè. 2015. Automatic Workarounds: Exploiting the Intrinsic Redundancy of Web Applications. *ACM Transactions on Software Engineering and Methodologies* 24, 3, Article 16 (May 2015), 42 pages. doi:10.1145/2755970

[7] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17754–17762.

[8] Karl Cunningham and Michael Kovacic. 2022. Troubleshooting: The "Acceptable" Energized Work. In *2022 IEEE IAS Pulp and Paper Industry Conference (PPIC)*. 65–70. doi:10.1109/PPIC52995.2022.9888912

[9] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T Nguyen, and Riccardo Rubei. 2021. Development of recommendation systems for software engineering: the CROSSMINER experience. *Empirical Software Engineering* 26, 4 (2021), 69.

[10] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2 (2023), 1.

[11] Marko Gasparic and Andrea Janes. 2016. What recommendation systems for software engineering recommend: A systematic literature review. *Journal of Systems and Software* 113 (2016), 101–113.

[12] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. 2019. Automatic Software Repair: A Survey. *IEEE Transactions on Software Engineering* 45, 1 (2019), 34–67. doi:10.1109/TSE.2017.2755013

[13] Sakib Haque, Zachary Eberhart, Aakash Bansal, and Collin McMillan. 2022. Semantic similarity metrics for evaluating source code summarization. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. 36–47.

[14] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural TExt Degeneration. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[15] Renjun Hu, Yi Cheng, Libin Meng, Jiaxin Xia, Yi Zong, Xing Shi, and Wei Lin. 2025. Training an LLM-as-a-Judge Model: Pipeline, Insights, and Practical Lessons. In *Companion Proceedings of the ACM on Web Conference 2025*. Association for Computing Machinery, 228–237. doi:10.1145/3701716.3715265

[16] Kahiomba Sonia Kiangala and Zenghui Wang. 2024. An experimental hybrid customized AI and generative AI chatbot human machine interface to improve a factory troubleshooting downtime in the context of Industry 5.0. *The International Journal of Advanced Manufacturing Technology* 132, 5 (2024), 2715–2733.

[17] Kevin Leach, Christopher S. Timperley, Kevin Angstadt, Anh Nguyen-Tuong, Jason Hiser, Aaron Paulos, Partha Pal, Patrick Hurley, Carl Thomas, Jack W. Davidson, Stephanie Forrest, Claire Le Goues, and Westley Weimer. 2022. START: A Framework for Trusted and Resilient Autonomous Vehicles (Practical Experience Report). In *Proceeding of the International Symposium on Software Reliability Engineering (ISSRE)*. 73–84.

[18] Christian Lovis and Benoît Debande. 2015. Troubleshooting: What Can Go Wrong and How to Fix It. In *Practical Guide to Clinical Computing Systems*. Elsevier, 111–136.

[19] Leonardo Mariani, Mauro Pezzè, Oliviero Riganelli, and Rui Xin. 2020. Predicting failures in multi-tier distributed systems. *Journal of Systems and Software* 161 (2020), 110464.

[20] Kim Martineau. 2023. What is retrieval-augmented generation? *IBM Blog* (2023).

[21] David E Mertz. 2024. The Challenges of Troubleshooting. In *2024 IEEE IAS Electrical Safety Workshop (ESW)*. IEEE, 1–6.

[22] Zecheng Ren, Zengnan Yu, Wenyi Zhang, and Qujiang Lei. 2024. Streamlining Industrial Robot Maintenance: An Intelligent Voice Query Approach for Enhanced Efficiency. *IEEE Access* 12 (2024), 121864–121881. doi:10.1109/ACCESS.2024.3452269

[23] Martin Robillard, Robert Walker, and Thomas Zimmermann. 2009. Recommendation systems for software engineering. *IEEE software* 27, 4 (2009), 80–86.

[24] Maria Teresa Rossi, Leonardo Mariani, and Oliviero Riganelli. 2025. From PREVENTion to REACTion: Enhancing Failure Resolution in Naval Systems. In *The 36th IEEE International Symposium on Software Reliability Engineering (ISSRE)*.

[25] Yan Su, Xue Rui Liang, Hui Wang, Jin Jun Wang, and Michael Gerard Pecht. 2019. A Maintenance and Troubleshooting Method Based on Integrated Information and System Principles. *IEEE Access* 7 (2019), 70513–70524. doi:10.1109/ACCESS.2019.2915327

[26] Peipei Xia, Li Zhang, and Fanzhang Li. 2015. Learning similarity with cosine similarity ensemble. *Information sciences* 307 (2015), 39–52.

[27] Mingyue Zhang, Zhi Jin, Jian Hou, and Renwei Luo. 2022. Resilient Mechanism Against Byzantine Failure for Distributed Deep Reinforcement Learning. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*. 378–389.

[28] Quanjun Zhang, Chunrong Fang, Yuxiang Ma, Weisong Sun, and Zhenyu Chen. 2023. A Survey of Learning-based Automated Program Repair. *ACM Transactions on Software Engineering and Methodology* 33, 2, Article 55 (Dec. 2023), 69 pages.

[29] Weixu Zhang, Yifei Wang, Yuanfeng Song, Victor Junqiu Wei, Yuxing Tian, Yiyan Qi, Jonathan H. Chan, Raymond Chi-Wing Wong, and Haiqin Yang. 2024. Natural Language Interfaces for Tabular Data Querying and Visualization: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (2024), 6699–6718. doi:10.1109/TKDE.2024.3400824

[30] Yun Zhu, Jia-Chen Gu, Caitlin Sikora, Ho Ko, Yinxiao Liu, Chu-Cheng Lin, Lei Shu, Liangchen Luo, Lei Meng, Bang Liu, et al. 2024. Accelerating inference of retrieval-augmented generation via sparse context selection. *arXiv preprint arXiv:2405.16178* (2024).