Benchmarking Large Language Models for Root Cause Analysis in Train Control Software Testing

1st Rahmanu Hermawan
Mälardalen University
Västerås, Sweden
rahmanu.hermawan@mdu.se

2nd Alessio Bucaioni
Mälardalen University
Västerås, Sweden
alessio.bucaioni@mdu.se

3rd Eduard Enoiu
Mälardalen University
Västerås, Sweden
eduard.paul.enoiu@mdu.se

4th Wasif Afzal
Mälardalen University
Västerås, Sweden
wasif.afzal@mdu.se

***Abstract*—Software quality assurance is critical in safety-critical domains like railway systems, where failures can have catastrophic consequences. In this context, train control and management systems play a central role, and their software must undergo rigorous validation. Alstom Rail Sweden AB employs a digital twin infrastructure to simulate and validate train control and management systems software. While this setup significantly improves system-level testing, the root cause analysis of test failures remains a manual and time-consuming bottleneck.**

This study explores the potential of large language models to automate root cause analysis by interpreting execution logs generated during digital twin-based testing. We benchmark seven state-of-the-art large language models, Aion-1.0, DeepSeek R1, DeepSeek V3 0324, Mistral Small 3.1 24B, GPT o3-mini, Gemini 2.5 Pro Experimental, and QwB 32B, using zero-shot chain-of-thought prompting to assess their ability to reason about fault patterns in real-world industrial test execution logs. The logs, sourced from Alstom's digital twin-based testing environment, capture complex operational behaviour typical of embedded, safety-critical systems.

Our results show that Gemini 2.5 Pro Experimental achieved the best performance with 66.7% accuracy and strong reasoning quality in this domain, contributing to the future research agenda to improve the accuracy prediction.

***Index Terms*—root cause analysis, log analysis, llm**

I. Introduction

Software plays a pivotal role in advancing safety, automation, and efficiency within the railway industry, underpinning everything from signalling systems to onboard diagnostics. Among its core technological pillars is the Train Control and Management System (TCMS), which orchestrates and oversees essential subsystems, including propulsion, braking, door control, and passenger information. By enabling real-time data exchange and decision making, the TCMS underpins the operational efficiency, safety, and reliability of modern railway systems.

Given the safety-critical nature of TCMS, it is important to ensure its software quality. Failures in such systems can have catastrophic consequences, including collisions or derailments. The complexity of TCMS software, coupled with rigorous regulatory and safety requirements, demands disciplined software engineering practices, with particular emphasis on quality assurance. An important part of software quality assurance is software testing, which systematically verifies that software behaves as specified and performs reliably across a range of conditions [1]. In the railway context, testing is conducted at multiple levels to ensure conformance to stringent safety and reliability standards.

Recently, the field has witnessed a shift towards augmenting, or even replacing, traditional testing practices with Digital Twin (DT) based methodologies [2]. A DT is a dynamic, high-fidelity virtual replica of a physical system, continuously synchronised with real-time operational data [3]. In the railway domain, DTs open new opportunities for advanced testing and validation by emulating real-world conditions, predicting failures, and optimising performance, all without requiring physical or actual assets. DT-based testing enables near real-time scenario validation, predictive diagnostics, and proactive fault detection [4].

Alstom Rail Sweden AB (Alstom) uses a DT infrastructure to execute and validate the TCMS software. The DT simulates the train's onboard software components, creating a controlled, virtualised environment for system-level testing without the need for physical prototypes. Alstom engineers manually derive test cases and configure the DT by integrating required subsystems, ensuring that TCMS components interact correctly and as intended.

Despite these advances, key bottlenecks persist, particularly in automating test result interpretation [5], [6]. When a test passes, the process proceeds seamlessly. However, in the event of a failure, engineers must conduct a detailed manual review of execution logs to identify the root cause. This manual analysis is labour-intensive, time-consuming, and at odds with industry imperatives such as rapid release cycles and operational agility [7]. Engineers must dissect verbose logs, trace execution flows, and infer causal relationships to pinpoint faults. This particular debugging process inflates the testing effort and hampers the overall validation timeline, posing a significant challenge for organisations striving to enhance efficiency and maintain competitiveness.

To address this challenge, we propose an approach to

automate the analysis of failed test execution logs using LLMs for faster Root Cause Analysis (RCA). Building on the DT infrastructure, the test log data is generated during the TCMS software testing at Alstom. We explore the feasibility and performance of LLMs in diagnosing the root causes of failed test execution logs. Specifically, we benchmark seven state-of-the-art LLMs: Aion-1.0, DeepSeek R1, DeepSeek V3 0324, Mistral Small 3.1 24B, GPT o3-mini, Gemini 2.5 Pro Experimental, and QwB 32B, using the zero-shot Chain-of-Thought (CoT) technique proposed by Kojima et.al [8], by simply guiding the LLM to think step-by-step or adding "think step-by-step" string before providing the answer. With this technique, we are evaluating LLMs' ability to infer fault explanations from real-world railway software logs. These logs, provided by Alstom, contain the nuanced operational traces typical of safety-critical embedded systems, posing a realistic and high-stakes testing ground for automated reasoning. Our study delivers quantitative evidence on the effectiveness of LLMs for this task.

The remainder of this paper is organised in the following: in Section II we describe the background of this research and review existing studies that relate to our work. In Section III, we present the research methodology when doing this work. In Section IV, we present the experiment results and discuss them in Section V Finally, in Section VI, we conclude the work.

II. Background and related work

In this section, we first provide the background for our research, followed by a review of related works.

A. Background

Alstom is a global leader in railway mobility, offering a comprehensive portfolio that includes rolling stock, digital and integrated systems, and associated services. Its site in Västerås, Sweden, focusses on the development and delivery of electric traction systems and TCMS. These traction systems integrate advanced sensors and control mechanisms to ensure optimal performance, energy efficiency, and safety. One of the primary challenges at Alstom lies in improving the efficiency of the testing process, particularly by automating the design and execution of test cases early in the development lifecycle **??**. A promising avenue to address this is the adoption of DT technology for testing purposes. DTs support the identification of both requirement-based test cases and critical scenarios, such as invalid states or conflicting requirements, which can be used iteratively across development cycles.

As in many industrial settings, testing at Alstom remains a cost-intensive activity. This is largely due to manual tasks such as creating and executing test cases, generating test reports, and analysing failed test execution logs. These manual workflows result in extended feedback loops and slower iteration cycles. Leveraging the existing available DT at Alstom and automating parts, or potentially the entirety, of the manual workflows could substantially reduce the testing cycle time. For instance, if we can support test engineers in analysing failed test execution logs, it would eliminate the need for them to manually inspect the logs in depth.

Figure 1 illustrates Alstom's *current manual debugging process* following a failed test. When a test fails, engineers follow a structured but time-consuming procedure. The process begins with reviewing the test logs. If the root cause is apparent, it is resolved and the test is re-executed. If not, the engineer sequentially inspects the signal values using proprietary diagnostic tools, reviews the test script for potential errors, and checks the configuration for possible misconfiguration. These steps are repeated iteratively until the problem is diagnosed and corrected, after which the test is re-run.

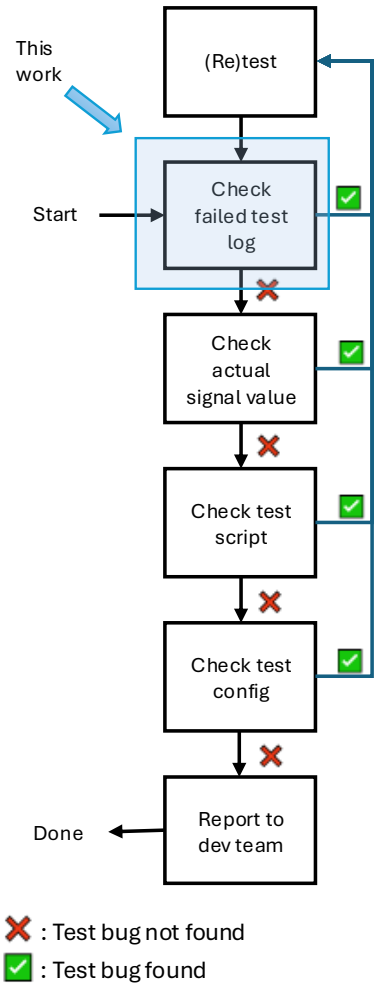

Fig. 1. Current manual failure debugging process at Alstom

Our ongoing work focusses specifically on automating this manual analysis phase of software test execution logs obtained from Alstom's internal systems. We focus on this

step because this is the foundation step where we can potentially generalise for the next steps, such as checking the actual signal's value, test script, and test configuration.

B. Related work

LLMs have been successfully applied in several tasks of software engineering, including code generation **??**, model transformations **??**, and self-healing code **??**. Even though several studies have demonstrated that LLMs can effectively analyse system logs, particularly for identifying the root causes of failures, the LLMs utilised did not have adequate reasoning capabilities. Now, when the LLM versions are heavily improved and enhanced, with reasoning capabilities, using this type of LLM for RCA is very interesting.

RCA is inherently complex, time-consuming and prone to errors, as it often requires manual examination of log traces. From a broader perspective, log analysis, much like RCA, is a demanding and labour-intensive task. Given its complexity, there is a growing trend toward automating this process using artificial intelligence (AI), particularly through machine learning (ML) and LLM.

Numerous studies leveraging ML techniques have demonstrated significant results. Du et al. [9] introduced DeepLog, a model designed to learn patterns from normal system logs and identify anomalies by detecting deviations from these learnt patterns. DeepLog employs a deep neural network based on Long Short-Term Memory (LSTM) to treat log sequences similarly to natural language, enabling effective pattern modelling. In another study, Zhang et al. [10] proposed LogRobust, which captures the semantic features of log events and represents them as semantic vectors. To detect anomalies, LogRobust utilises an attention-enhanced bidirectional LSTM (Bi-LSTM), allowing the model to understand contextual dependencies within log sequences and to automatically assess the significance of different log events. Their approach achieved high accuracy and demonstrated strong robustness when tested on Hadoop systems and real-world Microsoft online services.

Recognising the potential of LLMs for analytical tasks, Komal et al. [11] proposed an LLM-powered on-call system, utilising models such as GPT-4 and IBM's WatsonX, to automate RCA for cloud incidents in industrial environments that prioritise privacy. Their method incorporates multimodal data sources, including logs, metrics, traces, and alerts (LMTA), and aggregates key runtime diagnostic signals such as developer-defined alerts with corresponding thresholds and offsets. This integrated data is then used to predict the root cause category of incidents. The system achieved up to 97% accuracy in RCA. In a related effort, Shan et al. [12] developed a two-stage LLM-based approach, implemented as a tool named LogConfigLocalizer, to help users identify configuration errors in system logs. The first stage involves detecting log messages indicative of configuration issues (anomaly identification), followed by a second stage where these messages are used to localise the error (anomaly inference). Their tool demonstrated an impressive accuracy of 99.91%.

Chen et al. [13] introduced RCACopilot, an LLM-driven on-call assistant employing a few-shot learning paradigm to automate RCA in cloud settings. With GPT-4.0 as the language model, RCACopilot matches incidents with appropriate handlers based on alert type, aggregates relevant diagnostic data, predicts root cause categories, and generates interpretative explanations. The tool has been used by Microsoft for over four years and achieves an RCA accuracy of 76%. In another contribution, Han et al. [14] presented LasRCA, a hybrid framework combining an LLM with a lightweight ML-based classifier. During inference, the classifier performs an initial RCA stage, and the LLM refines the analysis using external resources such as troubleshooting guides, fault exemplars, and classifier confidence scores. Both approaches demonstrate the efficacy of LLMs in supporting RCA tasks.

However, unlike our work, these prior studies did not exploit the advanced reasoning capabilities of cutting-edge LLMs such as Gemini 2.5 Pro and DeepSeek R1. Leveraging reasoning-enhanced LLMs introduces the possibility of achieving high RCA accuracy in a zero-shot CoT setting, reducing the need for labelled data and extensive training. In addition, these studies primarily focused on system logs, leaving test execution log analysis largely unexplored. A failure in the software test execution could be due to misconfiguration or signal timeout. In this study, we specifically investigate the use of LLM for analysing test execution logs, addressing this gap in existing research.

III. Research methodology

This section outlines the methodology used to benchmark the performance of LLMs. To assess their effectiveness in performing RCA on software test execution logs, we conducted an empirical study involving seven LLMs: Aion-1.0, DeepSeek R1, DeepSeek V3 0324, Mistral Small 3.1 24B, GPT o3-mini, Gemini 2.5 Pro Experimental, and QwB 32B. We chose them due to their proven reasoning ability, which means that the LLM is able to cognitively derive a logical conclusion based on the given information [15]. Each model analysed a set of 15 software test

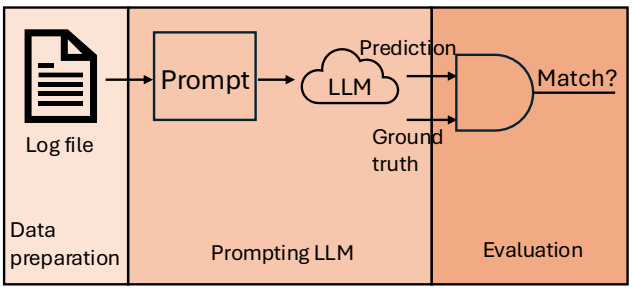

Fig. 2. Overview of the experiment pipeline

execution logs. The predicted root cause of each model was manually compared with a predefined ground truth. The predefined ground truth was deduced and the root causes of each failure log were manually determined based on an in-depth analysis by Alstom experts. For each correct prediction, the model received one point. These scores were then aggregated to identify which LLM achieved the highest score, which is reflective to the accuracy. The overall experimental pipeline is illustrated in Figure 2.

a) Data preparation: A total of 15 software test execution logs were prepared for analysis. The original logs we got from Alstom included extraneous information, such as test step counts and signal names, that other infrastructures of other domains may not have. To ensure generalisability, the logs were preprocessed to retain only generic log components such as (1) test step, (2) execution result, (3) timestamp and (4) message content. The refined logs were then structured in a JSON-based textual format to facilitate clarity, model interpretability, and future extensibility as shown by Listing 1. Furthermore, an automated obfuscation process was applied to anonymise sensitive information, such as project names, signal identifiers, test case labels, and file paths.

```
{
    "main_information": {
        "main_script_name": ▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
    ▒▒__131350_TrDmdRmp▒▒▒▒▒▒▒▒▒▒▒▒▒▒[]",
        "overall_execution_result": "failed",
        "date": "2024-04-08T14
    :15:17.236+02:00",
        "duration": "00:09:18.5110919"
    },
    "test_steps": [
        {
            "test_substep": "INFO",
            "execution_result": "InProgress",
            "timestamp": "2024-04-08T14
    :15:17.236+02:00",
            "message": "The testcase is
    running on framework version 2023.1.4..."
        },
        ...
    ]
}
```

Listing 1. Excerpt of test execution log

b) Prompting the LLM: We designed a fixed prompt template that incorporates zero-shot CoT reasoning by clearly specifying the role, input information, task description, and expected output. This technique is operationalised through an explicit task instruction, *"Provide a step-by-step reasoning to analyse the failure"*, which guides the LLM in inferring root causes from the implicit patterns within each log. The prompt text also included the preprocessed log along with a predefined list of the most frequent possible root causes, according to the insights gathered from Alstom test engineers. These causes included: (1) misconfiguration: failures due to incorrect test setups, (2) signal timeout (assertion failures): failures caused by signals not meeting expected conditions within a time limit, (3) expired bugfix: failures due to outdated or irrelevant fixes, (4) aborted test: failures resulting from premature test termination, and (5) unknown cause: cases where the failure could not be determined from the log alone. In classification tasks, the difficulty of accurately assigning a new instance to the correct class generally increases as the number of possible classes grows [16]. As this is an initial study, we deliberately limited the set of root causes to evaluate LLM performance under a less complex classification scenario. This approach allows us to establish a baseline and set more realistic expectations for future work involving a broader set of root causes. The complete prompt template for prompting the selected LLMs is provided in Listing 2.

```
You are a debugging assistant. Given the
    following information:

1. A test execution log
2. A list of possible root causes

Your task is to:
- Provide a step-by-step reasoning to analyze
    the failure
- Select the most likely root cause (only one)
     from the list
- Suggest a reasonable action or next step to
    resolve it
- Output the result in JSON format as shown
    below

---
Log File:
{"<log-file-name>"}

Possible Root Causes:
{ "misconfiguration", "signal_timeout",
"test_aborted", "expired_bugfix", "unknown" }

Respond strictly in the following JSON format:
{{
   "File name": "{file_name}",
   "Step-by-steps reasoning": "1. ...\n2. ...\
    n3. ...",
   "Root cause": "<select one from above>",
   "Action suggestion": "..."
}}
```

Listing 2. Prompt for instructing the LLM

Each model was accessed through the OpenRouter web interface[1], a platform that provides unified access to more than 440 publicly available LLMs. Similarly to the common way to interact with LLM, such as ChatGPT or Google NotebookLM, every time we prompt the LLM through the OpenRouter chatbox, the selected LLM responds to the prompt in the chat. The responses are manually exported and downloaded as a JSON file.

c) Evaluation: In this final step, we evaluate every root cause prediction made by each LLMs with a ground

[1]https://openrouter.ai/

truth of the root cause using quantitative criteria. Together with Alstom, each log was manually analysed to determine the root cause and serve as the ground truth, referred to as the oracle. To enhance readability and interpretability, the LLMs were instructed to respond in a structured JSON format. Additionally, we asked the models to include step-by-step reasoning in their responses to support future analysis of the LLMs' reasoning capabilities. We qualify if the response and the ground truth are an exact match or not. When the prediction and the ground truth are a match, we give a score of one for every match. As mentioned, we have prepared a list of possible root causes to be chosen from. When an LLM predicts the root cause is "test aborted", such as `"Root cause": "test_aborted",` we compare the exact root cause prediction string with the ground truth. If the string is exactly matched, we give the score of one to the LLM, and zero otherwise. We accumulated the scores from each log-prediction to get the complete evaluation score. In addition, since we instructed the LLM to also include its step-by-step reasoning, we judge its reasoning steps by giving a score between one and five based on a Likert scale. The Likert scale is chosen because it is known as the most appropriate way to measure attitude, behaviour, or opinion such as step-by-step reasoning [17]. In this way we can potentially relate the root cause prediction and step-by-step reasoning. For example, when an LLM's step-by-step reasoning is considered in line with Alstom's expert way of thinking, we give a score of three, four, or five, but one or two otherwise.

```
...
{
  "File name": "log-15.json",
  "Step-by-steps reasoning": "1. The `
    overall_execution_result` is \"failed\".\
    n2. The `duration` reported in `
    main_information` is \"00:00:00.0000000\",
     which is highly unusual for a test that
    has clearly logged steps over several
    minutes (timestamps range from 15:30:44 to
     15:37:54).\n3.
  ...",
  "Root cause": "test_aborted",
  "Action suggestion": "Investigate the test
    execution environment and controlling
    system (e.g., test runner, CI/CD pipeline)
     for reasons why the test execution might
    have been terminated prematurely around
    15:37:54 on 2024-05-15. Check for manual
    interruptions, system resource issues,
    crashes, or external stop commands.
    Additionally, investigate the 'package
    checksum mismatch' warning for the
    UserCode package ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒`_EM.zip` as a
     potential secondary issue."
}
```

Listing 3. Excerpt of LLM response

d) Implementing the pipeline: Prior to prompting the logs into the LLMs, an automated obfuscation procedure using a Python script was performed to preserve privacy. The obfuscating process uses an automated script based on modifiable and expandable regular expressions of the pattern of project name, signal names, test case names, and file path information. When the pattern matches, we replace the strings with "▒" as seen in Listing 2. With this modifiable obfuscation script, in the future, we can reuse the script to obfuscate more strings from other logs. After obfuscation, for this work-in-progress and to get a preliminary result as fast as possible, each log was manually embedded within the prompt text and manually submitted to OpenRouter through its web chat GUI interface. The LLMs returned diagnostic responses, which included step-by-step reasoning and a summary encoded in JSON format to regulate and simplify the response format as shown by Listing 3. In practice, the workflow for interacting with an LLM via OpenRouter involves the following steps: (1) log in through the OpenRouter website, (2) navigate to the "*Chat*" tab, (3) select and add desired language models, (4) input the prompt into the chat interface, and (5) export and download the generated output as JSON file, once the model has completed its response. From these JSON outputs, the predicted root cause was manually extracted and stored in a prepared spreadsheet file. In the spreadsheet file, we prepared a formula to automatically compare the root cause prediction and the ground truth. For each correct prediction, the LLM got a score of one. Considering it a tedious effort to quantify the correct prediction, we used a Python script to extract and calculate the score, including the percentage, and display it in a bar chart.

IV. Experimental results and analysis

This section presents and discusses the performance of each evaluated LLM in predicting the root causes of software test failures.

TABLE I
Number of correct prediction and reasoning score by the selected language models

Model name	Num. of correct prediction	Avg. of reasoning score
Aion-1.0	4	3.27
DeepSeek R1	5	3.80
DeepSeek V3 0324	5	3.73
GPT-o3 Mini	1	3.93
Gemini 2.5 Pro Experimental	**10**	**4.21**
Mistral Small 3.1 24B	5	3.13
QwB 32B	4	3.67

Table I shows the raw number of correct predictions and the average reasoning score of each LLM. From the table, we see that Gemini 2.5 Pro Experimental correctly predicts 10 root causes out of 15 logs with a reasoning score of 4.21 out of 5. While GPT-o3 Mini successfully predicted only one root cause, with a reasoning score of 3.93.

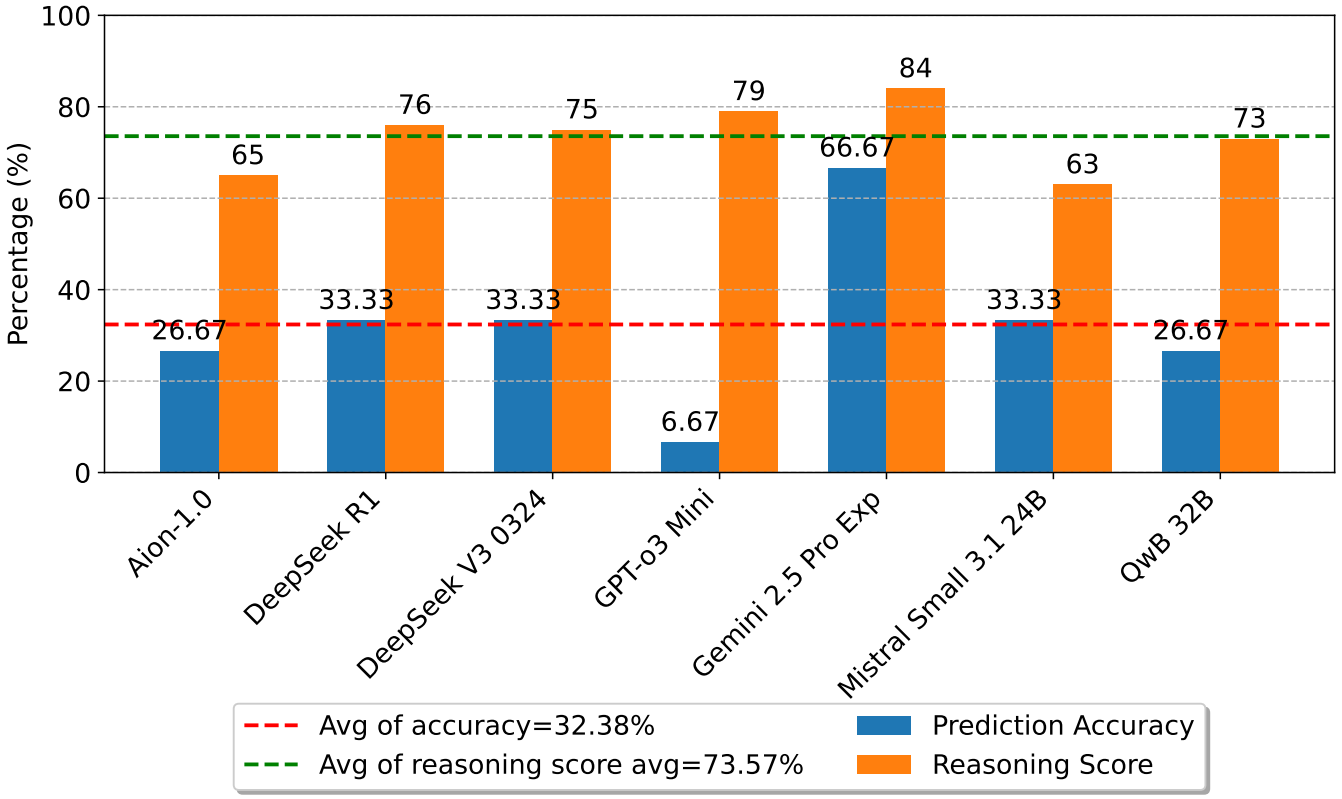

Fig. 3. Number of correct prediction by the selected language models

Figure 3 illustrates the comparative performance of the selected LLMs. Each LLM performance is illustrated in terms of accuracy and reasoning score. The accuracy percentage was calculated as the number of correct predictions divided by the total number of logs (15), multiplied by 100. For instance, when Gemini 2.5 Pro Experimental correctly predicted 10 root causes, its accuracy was calculated as $\frac{10}{15} \times 100\% = 66.67\%$. The reasoning score in Figure 3 is the percentage of the average of the LLM when doing the RCA for each log. As mentioned in Section III, we used a Likert scale to evaluate reasoning quality, where scores of 3, 4, or 5 indicated that the step-by-step reasoning was closely aligned with the explanation given by Alstom experts. The percentage score was calculated using the formula $\frac{\text{Given Score}}{\text{Max Score}} \times 100\%$. For example, if Gemini 2.5 Pro Experimental received a score of 4, its reasoning score would be $\frac{4}{5} \times 100\% = 80\%$. This process was repeated for each root cause prediction. Among the models, Gemini 2.5 Pro Experimental achieved the highest accuracy, correctly identifying the root cause in 10 out of 15 cases (66.7%). In contrast, GPT-03 Mini showed the lowest performance, with only one correct prediction (6.7%). Three models, DeepSeek R1, DeepSeek V3 0324, and Mistral Small 3.1 24B, achieved moderate performance, each correctly identifying 5 out of 15 root causes (33.3%). Both Aion-1.0 and QwB 32B made four correct predictions, resulting in an accuracy of 26.7%.

Figure 3 shows that Gemini 2.5 Pro Experimental achieves the highest RCA accuracy, 66.67%. In addition to its strong predictive capability, Gemini 2.5 Pro Experimental also demonstrates solid reasoning performance, with a reasoning score of 84%. These results highlight the potential of LLM to assist human analysts by not only identifying the root cause but also providing traceable and meaningful justifications, thereby supporting more reliable and interpretable RCA.

V. Discussion

Although the overall accuracy is relatively low, the reasoning score achieved by the LLM is rather high. This indicates that while LLMs excel at logical inference, they may not be suitable for fully automated RCA. Relying solely on LLMs for RCA is ineffective and may introduce new issues, such as prolonging the software development process or misidentifying bugs. For instance, an LLM might incorrectly identify a “signal timeout” or “test aborted” as the root cause, prompting test engineers to repeat the RCA process manually to determine the correct cause. This issue, ultimately will also increase the cost and prolong the time-to-market. Given that we are using zero-shot Co) prompting, one possible reason for the low accuracy is ineffective prompt design. Although zero-shot CoT has the potential to handle a range of root cause

predictions without requiring example-based reasoning, the complexity of the log content may limit its effectiveness in this context. Therefore, it may be worthwhile to explore alternative prompting strategies, such as few-shot CoT or standard CoT, which could better guide the model in reasoning through such intricate scenarios. Another possible low accuracy cause is due to the lack of fine tuning. The selected LLMs are *vanilla*, meaning that they are not tuned to a specific dataset of (failed) test execution logs. For instance, when an RCA prediction should be "test aborted" instead of "signal timeout", the LLM was misled by the presence of "WaitForCondition" string, as presented by Listing 4 at the end-section of the log.

```
{
    "main_information": {
        "main_script_name": ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
    _BI_110518_BgWtCrrFt▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒[]",
        "overall_execution_result": "failed",
        "date": "2024-05-15T15
    :08:04.152+02:00",
        "duration": "00:00:00.0000000"
    },
    "test_steps": [
    ...
    ...
       {
            "test_substep": "WaitForCondition
    ",
            "execution_result": "InProgress",
            "timestamp": "2024-05-15T15
    :13:41.956+02:00",
            "message": "Waiting (10000 ms) for
     signal (iCCUD3_BGC_ELoBgWgtM2B2) to reach
     the specified condition (Is.Equal) by a
    value of (True)."
        }
    ]
}
```

Listing 4. Excerpt of aborted test execution log

This string incorrectly led the LLM to determine "signal timeout" as the root cause. While the LLM's step-by-step reasoning (Listing 5) was logically consistent in detecting a "signal timeout" clue, it failed to consider that the test duration was 0—an important clue indicating the test did not complete and was likely aborted. With proper training and fine-tuning in a representative dataset, LLM's predictions could become significantly more accurate.

```
...
"Step-by-steps reasoning": [
    "1. The overall execution result is '
    failed'.",
    ...
    "3. Test steps 1 through 7 execute
    successfully, verifying weight correction
    factors under different bogie load
    conditions.",
    ...
    "8. The test verifies the presence of
    events 'BGC_ELoBgWgtDB1' (related), '
    BGC_ELoBgWgtM1B1', and 'BGC_ELoBgWgtM1B2'
    successfully (based on the `
    WaitForCondition` or `
    WaitForExpressionAtomic` steps passing).",
    "9. The test proceeds to wait for the
    event 'BGC_ELoBgWgtM2B2' to become active
    using a `WaitForCondition` step (signal `
    iCCUD3_BGC_ELoBgWgtM2B2` == `True`) with a
     timeout of 10000 ms.",
    "10. The log ends abruptly while this `
    WaitForCondition` step (`timestamp:
    \"2024-05-15T15:13:41.956+02:00\"`) is
    still marked as `InProgress`.",
    "11. This indicates the condition (`
    iCCUD3_BGC_ELoBgWgtM2B2 == True`) was not
    met within the 10-second timeout period,
    causing the test step to fail and likely
    halting the overall test execution."
],
"Root cause": "signal_timeout",
...
```

Listing 5. Excerpt of LLM's reponse when incorrectly determining a root cause of aborted test

When LLM correctly predicts a root cause, the step-by-step reasoning is robust. Listing 6 presents a log excerpt where LLM correctly predict a "signal timeout" root cause. From Listing 7, we see the step-by-step reasoning is solid to determine that the root cause is "signal timeout". The LLM successfully detects "message: Returned after 10050 ms."—a strong indication of "signal timeout".

```
{
    "main_information": {
        "main_script_name": ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
    _BI_110518_BgWtCrrFt▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒[]",
        "overall_execution_result": "failed",
        "date": "2024-05-28T09
    :01:45.858+02:00",
        "duration": "00:16:09.5782313"
    },
    ...
            "test_substep": "
    WaitForExpressionAtomic",
            "execution_result": "failed",
            "timestamp": "2024-05-28T09
    :07:43.932+02:00",
            "message": "Returned after 10050
    ms."
        },
    ...
    ]
}
```

Listing 6. Excerpt of log that contains signal timeout failure

```
"Step-by-steps reasoning": [
    "1. The overall test execution result is '
    failed'.",
    ...
    "5. The step failed with the message '
    Returned after 10050 ms.', indicating the
    10-second timeout was exceeded.",
    ...
    "7. The failure occurred because the
    expected signal state was not reached
    within the defined timeout period.",
```

```
    "8. This pattern directly corresponds to a
     signal timeout."
],
"Root cause": "signal_timeout",
```

Listing 7. Excerpt of LLM's reponse when determining a signal timeout failure

Given their strong reasoning capabilities in identifying potential failure causes, a more practical approach can be realised by using LLM in a human-in-the-loop setting. This hybrid method, referred to as LLM-agent, leverages the complementary strengths of human expertise and machine intelligence. Supporting this direction, Chen et al. [13] introduced RCACopilot, an LLM-agent-based tool that has been effectively used within Microsoft for four years to assist specifically solve cloud incidents. But RCA-Copilot, might not be tuned onto TCMS test log execution and used an older generation of LLM without reasoning capability. Our proposed solution will potentially enable Alstom's test engineers to achieve faster and efficient fault resolution by following the LLM's step-by-step reasoning. In addition, along with step-by-step reasoning, we also collected LLM's actionable suggestion such as seen in Listing 3, this solution is offering more concrete and actionable guidance for test engineers to take the next action related to the failure resolution. This will potentially shift the LLM's role from mere reasoning and analysis support to a more comprehensive decision aid. Furthermore, the test engineer could potentially brainstorm or discuss with LLM about the next steps in solving the failure. Figure 4 illustrates the debugging process when our proposed solution is integrated into Alstom's existing debugging process by enabling faster and more efficient failure resolution of a test execution, the implementation of semi-automated RCA, from analysis failed execution logs to checking signal values, test scripts, and configuration. The test engineer will benefit from the LLM's step-by-step reasoning and action suggestion to make a decision whether to repeat the test or report the bug to the development team. When our proposed solution is improved to be a fully automatic RCA system, Alstom's DT infrastructure will potentially get a benefit such as a shorter feedback loop between the testing and development phases, allowing automatic feedback to developers for failure resolution. Consequently, this can enhance the overall speed of the software development cycle, ultimately contributing to faster time-to-market.

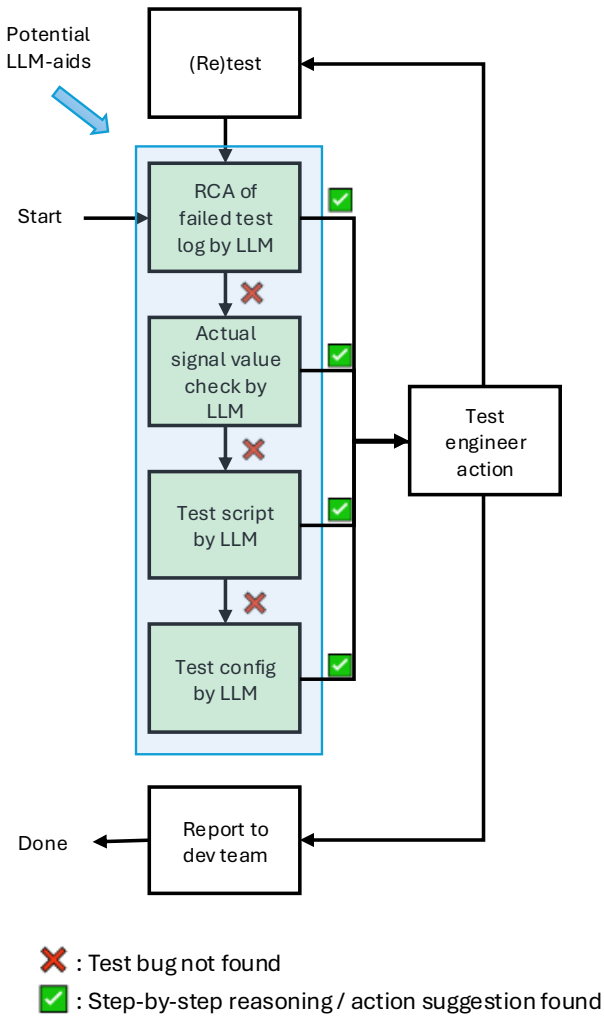

Fig. 4. Semi-automatic failure debugging process at Alstom when LLM's RCA is integrated

A. Threats to validity

Although this study represents an initial investigation into the capabilities of LLMs to perform RCA on real industrial log files, conducted in close collaboration with our industrial partners, there remain potential threats to validity that may limit the generalisability of our findings beyond the specific context of this research. Concerning internal validity, the pipeline was manually implemented, which may have introduced the risk of human error. Specifically, the ground truth labels for each log were established through manual analysis that is potentially introducing subjectivity or inconsistencies in the analysis process. Regarding external validity, the dataset used in this study was limited in both size and diversity, as it was sourced from a single industrial partner and consisted of only 15 log instances. As a result, the findings may not generalise to other domains, log formats, or operational contexts. Furthermore, the evaluated LLMs represent only a specific subset of currently available models, and their performance may not reflect that of other existing or future-generation LLMs. Conclusion validity may have been affected by the small dataset size, and caution is warranted when interpreting the results. Although one LLM outperformed others in this study, the limited number of examples restricts our ability to draw statistically significant conclusions. As noted by Fei et al. [18], expanding the dataset in both size and diversity would enhance the reliability of our findings and allow for more robust

assessments of LLM performance in RCA.

VI. Conclusion and Future Work

According to the current state of research, this appears to be the first peer-reviewed study to explore the feasibility of using LLMs to automate RCA for failed test execution log in safety-critical railway software testing, specifically focussed on execution logs generated by Alstom's DT infrastructure. Given the complexity and criticality of TCMS, and the growing industrial interest in DT-based validation, our goal was to assess whether LLMs could assist engineers by interpreting test failures and identifying plausible fault explanations. To this end, we benchmarked seven contemporary LLMs, Aion-1.0, DeepSeek R1, DeepSeek V3 0324, Mistral Small 3.1 24B, GPT-03 Mini, Gemini 2.5 Pro Experimental, and QwB 32B, using the zero-shot CoT prompting technique, which promotes step-by-step reasoning. We evaluated the models against a curated dataset of 15 real-world test logs, using two metrics: prediction accuracy (correct identification of root cause) and reasoning score (alignment of step-by-step logic with expert explanations). Our results show that Gemini 2.5 Pro Experimental achieved the best performance with 66.7% accuracy and strong reasoning quality. Other models reached moderate accuracy (33.3%), while GPT-03 Mini lagged behind at 6.7%. With an average accuracy of 32.38%, current LLMs, with minimal optimisation, show potential, but remain inconsistent and with different model, different results might potentially be gathered. While structured prompting can support the RCA of failed test execution log, LLMs are not yet reliable enough to replace manual analysis by human in safety-critical contexts.

There are several avenues for future exploration. First, expanding the dataset to include larger and more diverse log instances, spanning different TCMS subsystems and failure types, would improve the generalisability and statistical power of the findings. In this direction, we have prepared the test execution log as generic as possible as a log model, which potentially represents diverse test execution logs from other domains. Therefore, we also consider evaluating the proposed solution in other domains. Second, optimising prompts and fine-tuning LLMs on domain-specific log data could significantly enhance reasoning accuracy and reduce hallucinations or irrelevant outputs. Third, exploring hybrid approaches that combine LLM-based reasoning with symbolic rules or anomaly detection systems may result in more reliable and explainable RCA in particular for failed test execution log. Finally, adopting human-in-the-loop strategies, where LLMs assist rather than replace experts, could accelerate log analysis while maintaining critical human oversight for safety-critical decisions.

Data availability

A publicly available replication package is provided at the following repository: https://zenodo.org/records/15753920.

Acknowledgement

This work is supported by the Swedish Agency for Innovation Systems through the project "Secure: Developing Predictable and Secure IoT for Autonomous Systems" (2023-01899), and by the Key Digital Technologies Joint Undertaking through the project "MATISSE: Model-based engineering of digital twins for early verification and validation of industrial systems" (101140216).

The authors would like to thank Zulqarnain Haider and the entire team at Alstom Rail Sweden AB in Västerås for their valuable support throughout this work.

References

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*, second edition. ed. Cambridge: Cambridge University Press, 2016.

[2] R. J. Somers, J. A. Douthwaite, D. J. Wagg, N. Walkinshaw, and R. M. Hierons, "Digital-twin-based testing for cyber–physical systems: A systematic literature review," *Information and Software Technology*, vol. 156, p. 107145, Apr. 2023. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584922002543

[3] E. Ferko, A. Bucaioni, and M. Behnam, "Architecting Digital Twins," *IEEE Access*, vol. 10, pp. 50 335–50 350, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9770073/

[4] A. Löcklin, M. Müller, T. Jung, N. Jazdi, D. White, and M. Weyrich, "Digital Twin for Verification and Validation of Industrial Automation Systems – a Survey," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Sep. 2020, pp. 851–858. [Online]. Available: https://ieeexplore.ieee.org/document/9212051/?arnumber=9212051

[5] P. E. Strandberg, W. Afzal, and D. Sundmark, "Decision making and visualizations based on test results," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3239235.3268921

[6] ——, "Software test results exploration and visualization with continuous integration and nightly testing," *International Journal on Software Tools for Technology Transfer*, vol. 24, pp. 261–285, 2022.

[7] J. Wang, G. Chu, J. Wang, H. Sun, Q. Qi, Y. Wang, J. Qi, and J. Liao, "LogExpert: Log-based Recommended Resolutions Generation using Large Language Model," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 42–46. [Online]. Available: https://dl.acm.org/doi/10.1145/3639476.3639773

[8] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2022.

[9] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1298. [Online]. Available: https://doi.org/10.1145/3133956.3134015

[10] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–817. [Online]. Available: https://doi.org/10.1145/3338906.3338931

[11] K. Sarda, Z. Namrud, I. Watts, L. Shwartz, S. Nagar, P. Mohapatra, and M. Litoiu, "Augmenting automatic root-cause identification with incident alerts using llm," in *2024 34th International Conference on Collaborative Advances in Software and COmputiNg (CASCON)*, 2024, pp. 1–10.

[12] S. Shan, Y. Huo, Y. Su, Y. Li, D. Li, and Z. Zheng, "Face it yourselves: An llm-based two-stage strategy to localize configuration errors via logs," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '24: 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. Association for Computing Machinery, 2024-09-11, pp. 13–25.

[13] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xu, "Automatic Root Cause Analysis via Large Language Models for Cloud Incidents," in *Proceedings of the Nineteenth European Conference on Computer Systems*, ser. EuroSys '24. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 674–688. [Online]. Available: https://dl.acm.org/doi/10.1145/3627703.3629553

[14] Y. Han, Q. Du, Y. Huang, J. Wu, F. Tian, and C. He, "The Potential of One-Shot Failure Root Cause Analysis: Collaboration of the Large Language Model and Small Classifier," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. Sacramento CA USA: ACM, Oct. 2024, pp. 931–943. [Online]. Available: https://dl.acm.org/doi/10.1145/3691620.3695475

[15] J. Huang and K. C.-C. Chang, "Towards reasoning in large language models: A survey," in *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 1049–1065. [Online]. Available: https://aclanthology.org/2023.findings-acl.67/

[16] P. D. Moral, S. Nowaczyk, and S. Pashami, "Why is multiclass classification hard?" *IEEE Access*, vol. 10, pp. 80 448–80 462, 2022.

[17] A. Joshi, S. Kale, S. Chandel, and D. Pal, "Likert scale: Explored and explained," *British Journal of Applied Science & Technology*, vol. 7, pp. 396–403, 01 2015.

[18] W. Fei, X. Niu, G. Xie, Y. Zhang, B. Bai, L. Deng, and W. Han, "Retrieval meets reasoning: Dynamic in-context editing for long-text understanding," 2024. [Online]. Available: https://arxiv.org/abs/2406.12331