Evaluating LLMs and Prompting Strategies for Automated Hardware Diagnosis from Textual User-Reports

Carlos Caminha, Maria de Lourdes M. Silva, Iago C. Chaves, Felipe T. Brito, Victor A. E. Farias, Javam C. Machado

Laboratório de Sistemas e Banco de Dados (LSBD)
Departamento de Computação / UFC – Fortaleza – CE – Brazil

{carlos.caminha, malu.maia, iago.chaves, felipe.timbo,victor.farias, javam.machado}@lsbd.ufc.br

Abstract. *Computer manufacturers offer platforms for users to describe device faults using textual reports such as "My screen is flickering". Identifying the faulty component from the report is essential for automating tests and improving user experience. However, such reports are often ambiguous and lack detail, making this task challenging. Large Language Models (LLMs) have shown promise in addressing such issues. This study evaluates 27 open-source models (1B–72B parameters) and 2 proprietary LLMs using four prompting strategies: Zero-Shot, Few-Shot, Chain-of-Thought (CoT), and CoT+Few-Shot (CoT+FS). We conducted 98,948 inferences, processing over 51 million input tokens and generating 13 million output tokens. We achieve f1-score up to* 0.76*. Results show that three models offer the best balance between size and performance: mistral-small-24b-instruct and two smaller models, llama-3.2-1b-instruct and gemma-2-2b-it, that offer competitive performance with lower VRAM usage, enabling efficient inference on end-user devices as modern laptops or smartphones with NPUs.*

1. Introduction

Advancements in electronic fabrication have enabled large-scale production of computer components, but faults over time remain common [Queiroz et al. 2016a, Queiroz et al. 2016b, Pereira et al. 2020]. This has increased the need for robust diagnostic systems to ensure reliability and efficiency. However, manually diagnosing these problems can be time-consuming and inefficient, making it essential for these systems to automatically identify the faulty component. By doing so, the manufacturer could also initiate an automated process to identify the suspected components, enhancing the troubleshooting process and reducing the need for manual intervention. Automating this process would enhance efficiency, lower support costs, and improve the user experience. However, this task is highly challenging because user-reported issues are typically described in free text, e.g., "*my connection keeps timing out when I try to access the internet*", which could be ambiguous, unstructured, incomplete, and lacks technical precision [Silva et al. 2025].

A promising solution to this problem lies in the Large Language Models (LLMs), which continue to evolve and significantly transform various domains, demonstrating their remarkable ability to perform a wide range of tasks, from natural language understanding to solving complex problems [Hadi et al. 2023]. Recent advancements have empowered these models to produce human-level responses, supporting a wide range of tasks

such as programming [Nam et al. 2024], time series forecasting [Bastos et al. 2025], synthetic data generation [Karl et al. 2024], classification [Abburi et al. 2023], information extraction [Almeida and Caminha 2024], and even complex problem-solving [Rasal 2024].

Leveraging LLMs to analyze user-reported problem descriptions and infer potential causes presents a viable solution. These models can rapidly process ambiguous textual reports, extract relevant symptoms, and match them with likely hardware issues based on patterns learned.

For hardware manufacturers and service providers, open-source LLMs can play a key role in adopting diagnostic tools. While proprietary models such as GPT-4 and Gemini offer advanced capabilities, their reliance on external cloud services raises concerns about data privacy, competitive intelligence, and vendor dependency. Utilizing open-source models allows companies to tailor customized solutions for their proprietary diagnostic pipelines without exposing sensitive data to third parties. This approach is beneficial for businesses of all sizes, ensuring that even smaller manufacturers can integrate advanced AI models into their products without high licensing costs or data security risks.

For the effective implementation of LLM-based diagnostics, it is essential to conduct a comprehensive evaluation of open-source models across different families and parameter count. Various geopolitical and strategic factors influence model selection, as organizations may prefer specific families due to regulatory requirements, intellectual property restrictions, or regional market policies. For instance, some companies may prioritize locally developed models, such as Qwen or DeepSeek, while others may opt for Meta's LLaMA or Google's Gemma due to existing partnerships and technological preferences.

In terms of parameter count, larger models tend to perform better with the trade-off of requiring more computational resources. Some companies may favor performance and deploy large models in servers with GPUs. On the other hand, given the growing integration of Neural Processing Units (NPUs) in modern devices, some organizations might prefer to deploy compact models capable of operating efficiently in edge devices. Understanding what is recommended for each case is crucial.

In this work, we tackle, by textual interaction with a computer user, the problem of predicting the computer component that is possibly causing an issue reported by the user. Our approach employs large language models using prompting engineering to map a textual user report to a possibly defective computer component. We conduct an extensive benchmark of open-source LLMs with 27 open-source models ranging from 1 billion to 72 billion parameters, along with two proprietary models. The inclusion of proprietary models helps assess the extent to which they outperform open-source models and whether the difference justifies choosing closed-source solutions for this application. Additionally, we conducted experiments using four different prompt engineering strategies: Zero-Shot, Few-Shot, Chain-of-Thought (CoT), and Chain-of-Thought combined with Few-Shot (CoT+FS).

The contributions of this paper are listed below:

1. We develop an LLM-based strategy for classifying textual user reports for possibly faulty computer components: video card, storage, network, motherboard,

memory, CPU/FAN/Heatsink, battery, and audio.
2. We conduct an extensive experimental evaluation of LLMs from various families and sizes to recommend models that offer the best balance between performance and model size.

The remainder of this paper is structured as follows. Section 2 reviews related work on LLM-based automated diagnosis. Section 3 describes our methodology, detailing the dataset used, the open-source models evaluated, and the prompting strategies employed in the experiments. Section 4 presents the experimental setup and results, analyzing the performance of different models and prompting techniques. Finally, Section 5 summarizes our findings and outlines future directions for improving LLM-based hardware fault diagnosis.

2. Related Work

Despite the growing interest in LLM research, their application in hardware failure detection for automated diagnostics remains limited. Since automated diagnostics rely on accurately identifying the category of a potentially faulty component within the data, we define the scope of our related work as the use of LLMs for automated diagnosis. Standard techniques in this domain include prompt engineering [Wang et al. 2024, Makram and Mohammcd 2024], fine-tuning [Zheng et al. 2024], and knowledge distillation [Nathani et al. 2024].

[Li et al. 2023] propose a strategy for adapting LLMs, specifically Llama 2 with 7 billion parameters, to specialized fields such as fault detection and automated diagnostics in industrial machinery. Their approach involves constructing a knowledge graph tailored to the industrial domain to enhance fault diagnosis. This knowledge graph is then used to fine-tune LLMs, improving their ability to perform the target task accurately. Similarly, [Tao et al. 2025] employs LLMs, specifically the open-source ChatGLM2 with 6 billion parameters, for bearing fault diagnostic. They convert features, such as vibration signals, into textual data, enabling the application of LLMs to this problem. In line with the previous work, the authors fine-tuned the pre-trained model using their first contribution, the textual data. On the other hand, similar to this work, [Silva et al. 2025] addresses the problem of predicting faulty hardware components using smaller models, between 22 million and 407 million parameters. The authors made an empirical comparison between different language models employing zero-, one-, and few-shot prompting strategies.

3. Methodology

This section outlines the methodology employed in our work. We consider our task a classification problem, where we aim to label a report that contains complaints about hardware issues. For instance, the model receives as input the following text "My connection keeps timing out when I try to access the internet." and needs to return the "Network" label. We use prompt strategies in LLMs to solve this classification problem.

We leveraged LLMs in combination with prompting strategies to improve their performance on the classification task. To assess the effectiveness of the approach, we evaluated the model's predictions using a dataset specific to the target task, and then measured the quality of predictions using the F1-Score, a well-known metric for classification problems. Additionally, we compare various approaches, varying LLM families, LLM

sizes, and prompting strategies to make an empirical evaluation and define the best LLM recommendation for this kind of task considering the balance of model's size and performance. Our methodology follows the described pipeline that is also summarized in Figure 1.

Section 3.1 explains how we selected LLMs for the empirical evaluation and describes their characteristics, Section 3.2 details the prompting techniques that were combined with the LLMs, and Section 3.3 presents the evaluation metric used to measure the predictions' quality of the approaches.

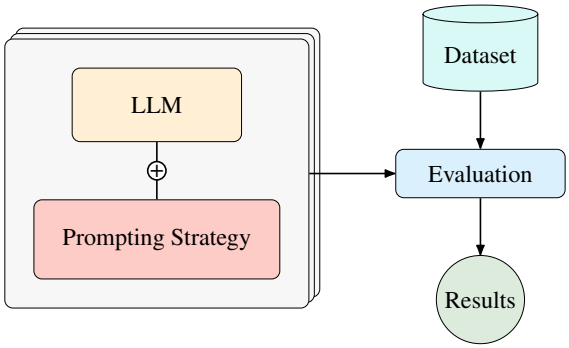

Figure 1. Pipeline for evaluating LLMs in the task of faulty hardware diagnosis.

3.1. Open Source Large Language Models

To conduct this study, we selected LLMs based on their performance ranking provided by the Chatbot Arena[1] up until January 1, 2025. The selection process follows two constraints: (i) Models families that provided versions with a maximum size of 72 billion parameters; (ii) Pre-quantized models in the GPT-Generated Unified Format (GGUF) with 4-bit quantization.

The first constraint ensures comparability and efficiency in test executions, while the second optimizes memory usage (VRAM) during model inference. All selected models are open-source and available on the Hugging Face platform[2]. Table 1 provides an overview of the LLMs used in this study, including their source platform path, parameter size in billions, and VRAM consumption in GB. The VRAM usage is measured based on inference with a context length of eight thousand tokens.

3.2. Prompting Techniques

We employed four distinct prompting strategies for each LLM: (i) Zero-Shot, (ii) Few-Shot, (iii) Chain-of-Thought (CoT), and (iv) CoT combined with Few-Shot. Each prompt contains specific instructions for the model to identify faulty hardware components based on user-provided descriptions, with the output being a dictionary indicating the predicted component. The prompts used to define these strategies are illustrated in Figure 2. The frame labeled "Common Prompt" is present in all prompts and outlines the target task of predicting faulty hardware components, asking the LLM to respond to user queries. The frame labeled "Chain of Thoughts" provides a step-by-step reasoning for the solution. The

[1] `https://chat.lmsys.org/`
[2] `https://huggingface.co/`

Table 1. Overview of opensource LLMs used in this study, detailing their source platform path, parameter size (in billions), and VRAM consumption during inference with a context length of eight thousand tokens.

Hugging Face path	Size (B)	VRAM (GB)
lmstudio-community/deepseek-r1-distill-llama-70b	70	45.0
lmstudio-community/deepseek-r1-distill-qwen-32b	32	22.2
lmstudio-community/deepseek-r1-distill-qwen-14b	14	11.4
lmstudio-community/deepseek-r1-distill-llama-8b	8	7.8
lmstudio-community/deepseek-r1-distill-qwen-1.5b	1.5	3.9
lmstudio-community/gemma-2-27b-it	27	19.2
lmstudio-community/gemma-2-9b-it	9	8.4
lmstudio-community/gemma-2-2b-it	2	4.2
lmstudio-community/llama-3.3-70b-instruct	70	45.0
lmstudio-community/meta-llama-3-8b-instruct	8	7.8
lmstudio-community/llama-3.2-3b-instruct	3	4.8
lmstudio-community/llama-3.2-1b-instruct	1.2	2.8
lmstudio-community/qwen2.5-72b-instruct	72	46.2
bartowski/qwen2.5-32b-instruct	32	22.2
lmstudio-community/qwen2.5-14b-instruct	14	11.4
lmstudio-community/qwen2.5-7b-instruct-1m	7	7.2
lmstudio-community/qwen2.5-1.5b-instruct	1.5	4.1
lmstudio-community/phi-4	14	11.4
lmstudio-community/phi-4-mini-instruct	3.8	5.3
TheBloke/zephyr-7b-beta	7	7.2
TheBloke/stablelm-zephyr-3b	3	4.8
stabilityai/stablelm-zephyr-1_6b	1.6	4.0
lmstudio-community/mistral-small-24b-instruct-2501	24	17.4
lmstudio-community/mistral-7b-instruct-v0.3	7	7.2
TheBloke/yi-34b	34	23.4
MaziyarPanahi/yi-9b	9	8.4
TheBloke/yi-6b	6	6.6

"Few-Shot" frame includes examples of how to respond to user queries. The prompting strategies are summarized as follows:

i. The Zero-Shot strategy uses only the prompts labeled as "Common Prompt";
ii. Few-Shot strategy combines the "Common Prompt" and "Few-shot" frames;
iii. CoT strategy merges the frames labeled as "Common Prompt" and "Chain of Thoughts";
iv. The strategy that combines few-shot and CoT strategies contains all the frames described in Figure 2.

3.3. Evaluation Metrics

In this study, we evaluate the performance of LLMs in a classification task aimed at predicting faulty hardware components. Given the nature of our dataset and class imbalance, we adopt F1-score as the primary evaluation metric for balancing precision and recall and handle imbalanced data. This metric combines precision and recall in a single metric and it is computed as: $F1\ Score = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$, where precision and recall are defined

Common Prompt

```
You are an intelligent assistant capable of identifying faulty computer hardware components based on descriptions
provided by their owners.  Your main function is to analyze user reports of computer problems and determine which
hardware component is most likely causing the issue.
Consider the following hardware components:
* Audio:  Related to sound, speakers, headphones, and microphone issues.
* Battery:  Related to charging problems, battery life, or power issues if applicable.
* CPU/Fan/Heatsink:  Related to processing, overheating, or fan operation problems.
* Memory:  Related to random crashes, blue screen errors (BSOD), or data corruption.
* Motherboard:  Related to general computer malfunctions, boot failures, or problems with ports and connectors.
* Storage:  Related to hard drives (HDDs) or solid-state drives (SSDs), such as slowness, read/write errors, or
corrupted files.
* Video Card:  Related to graphics problems, such as visual artifacts, poor gaming performance, or issues with
multiple monitors.
* Network:  Related to internet connection problems, either wired (Ethernet) or wireless (Wi-Fi).

When a user reports a problem, analyze their description and identify the most likely faulty hardware component.
```

Chain of Thoughts

```
First, explain your reasoning step-by-step, considering the user's description and the possible components that could
be responsible.  Then, based on your reasoning, provide your final answer in the specified format.  Respond ONLY with
a Python dictionary where the key is "component" and the value is the name of the appropriate hardware component from
the list above.  Do not generate any additional text in your response under any circumstances *after the dictionary*.
Ensure the component suggested is STRICTLY one from the provided list, and indicate only ONE component.  The component
name must MATCH EXACTLY one of the options listed above, with no variations in capitalization, spacing, or wording.
Do not add or remove spaces, change the case, or alter the names in any way.
```

Few-shot

```
Here are a few examples of how to respond to user queries:
User query:  My computer keeps crashing randomly, and sometimes I see a blue screen with an error message.  Response:
Let's analyze the problem.  The user reports random crashes and blue screen errors (BSOD). These symptoms are most
commonly associated with faulty RAM, which is listed as "Memory".  Other components like the motherboard or storage
*could* cause crashes, but BSODs are strongly indicative of memory issues.
{{"component":  "Memory"}}

User query:  I can't hear any sound coming from my speakers, and I've already checked the volume controls.  Response:
The user reports no sound from the speakers, and the volume controls are not the issue.  This points directly to a
problem with the audio output system.  The relevant component is "Audio".
{{"component":  "Audio"}}

User query:  My computer is running very slowly, and it takes a long time to open files or save documents.  Response:
The user describes slow performance, specifically when opening and saving files.  This strongly suggests a problem
with the storage device, where the files are read from and written to.  The component is "Storage".
{{"component":  "Storage"}}
```

Common Prompt

```
Following is the user query that you detect the faulty component:
User query:  {user_query}
Response:
```

Figure 2. Overview of prompting strategies—Zero-Shot, Few-Shot, Chain-of-Thought (CoT), and Few-Shot with CoT—to guide LLMs in identifying faulty hardware components. Each strategy integrates specific prompt frames to structure model responses based on user-provided descriptions, with outputs formatted as dictionaries predicting the faulty component.

based on true positive (TP), false negative (FN), and false positive (FP) predictions, and are computed as: $\textit{Precision} = \frac{TP}{TP+FP}$ and $\textit{Recall} = \frac{TP}{TP+FN}$.

We also evaluate the models by jointly analyzing their performance and size using the Pareto frontier, a well-established concept from multi-objective optimization [Lotov and Miettinen 2008]. The Pareto frontier captures the set of models that represent the best possible trade-offs between competing objectives. Since larger models tend to perform better than smaller ones but highly increase computational and memory costs, the competing objectives are model performance, computed by F1-score, and model size. The Pareto frontier helps to find smaller models that achieve high performance. By plotting model size on the X-axis and F1-score on the Y-axis in a 2D space, we identify the Pareto-optimal set: models for which no other model exists that is both more accurate and smaller. The Pareto frontier is thus created by connecting these non-dominated models, enabling us to visualize and compare the most efficient options with respect to the

trade-off between accuracy and model complexity.

4. Experimental Evaluation

This section details our experimental evaluation of the selected open-source large language models and prompting strategies for classifying hardware faults from textual user-reports, aiming to determine the most accurate and efficient model-strategy combinations.

4.1. Dataset

We evaluated our proposed combinations for classifying user-reports of hardware faults using the FACTO dataset [Silva et al. 2024], which consists of 853 user-generated textual reports on hardware diagnosis. The dataset includes three main sources: surveys conducted with IT professionals, automated collection from specialized online forums, and synthetic generation. Each entry provides a description of a specific problem (content), the affected hardware component (label), and the source of information (source).

4.2. Evaluation Setup

To evaluate the language models listed in Table 1, we employed the `llama.cpp`[3] library. Models were limited to a context window of 8,192 tokens and used the `Q4_K_M` quantization strategy, a 4-bit per parameter INT4 quantization combining group quantization with mean compensation. This approach significantly reduces memory consumption without substantial performance degradation. All model layers were fully loaded onto the GPU to optimize inference speed. The experiments were performed on a local workstation equipped with a NVIDIA RTX6000 ADA GPU featuring 48 GB of VRAM.

Ensuring consistency in model outputs is critical for automated hardware fault diagnosis since structured outputs enable automated processes and integration into diagnostic pipelines. However, LLMs may generate outputs with varying levels of formatting quality, potentially complicating automated extraction tasks. To illustrate this issue, we present representative examples of outputs generated by the evaluated models in Figure 3. It shows six examples of outputs from the component extraction task based on textual user-reports. The three examples in the first row, (a), (b), and (c), illustrate correct identification of the faulty hardware component, either as a direct JSON dictionary or embedded within correctly structured text. The second row, examples (d), (e), and (f), highlights cases where the models failed to produce the expected output. These cases emphasize the importance of maintaining consistent output formatting for reliable automated processing.

[3] `https://github.com/ggerganov/llama.cpp`

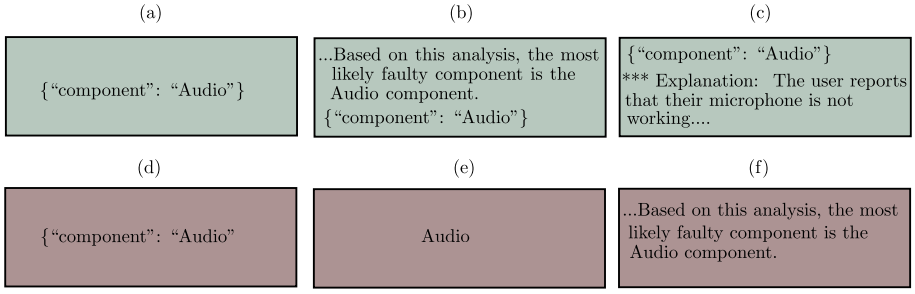

Figure 3. Examples of successful (a–c) and unsuccessful (d–f) outputs from the faulty hardware component extraction task using textual user-reports.

4.3. Results and Discussion

We evaluated the four prompting strategies described in Figure 2 using the LLMs detailed in Table 1. In total, 98,948 inferences were executed, processing 51,641,460 input tokens and generating 13,259,092 output tokens. This extensive variation in model sizes, architectures, and prompting techniques provides a solid foundation for analyzing the impact of model size and prompting strategy on inference quality.

Figure 4 presents a comparative analysis of the F1-Score results obtained for the four prompting strategies. Each plot corresponds to one of these strategies, where the horizontal axis represents model size (in billions of parameters), and the vertical axis shows the corresponding F1-Score values. Error bars indicate the standard deviation of the F1-Score, estimated using the bootstrap resampling method [Efron and Tibshirani 1986]. The GPT models do not appear in the figure because their respective sizes in billions of parameters are not known.

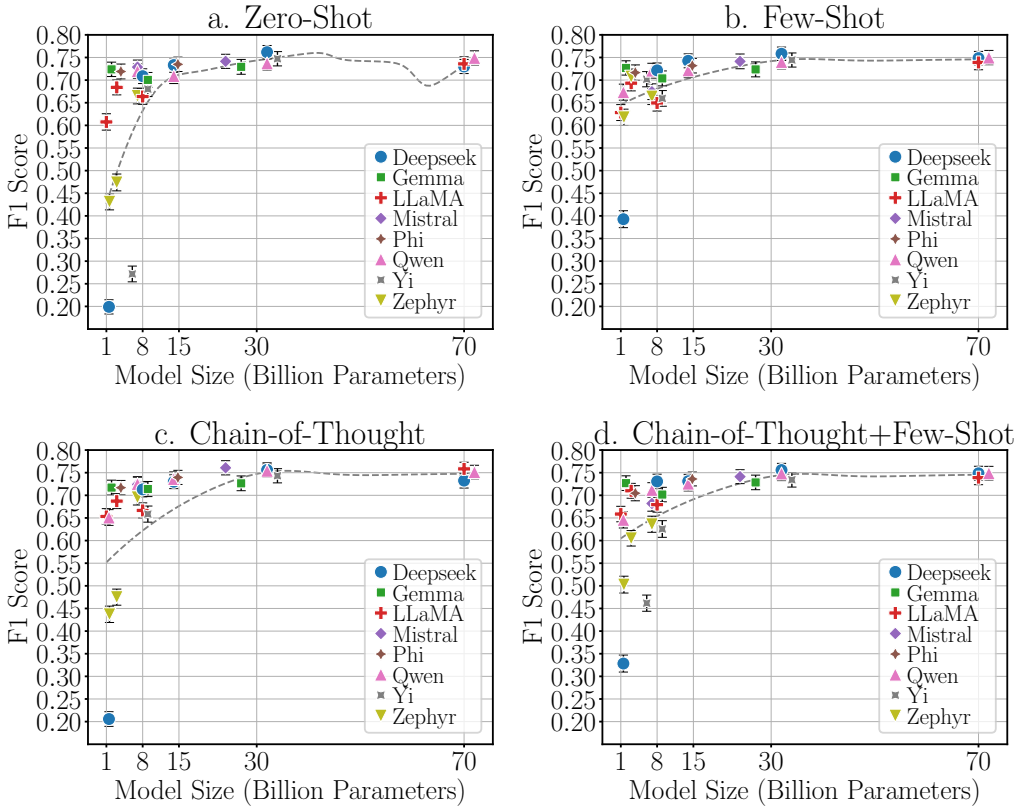

Figure 4. Comparison of F1-Scores across four prompting strategies in relation to model size. The plots show the performance of various open-source LLMs, with error bars indicating the standard deviation of the F1-Score. Dashed lines represent non-parametric regressions.

Specifically, for each model, 1,000 random resamples with replacement were performed on the 853 inference examples from the FACTO Dataset, generating a distribution of F1-Score values. Dashed lines represent non-parametric regressions estimated using the Nadaraya-Watson method [Nadaraya 1964]. This analysis reveals a general performance improvement as model size increases. However, growth levels off near 30 billion parameters, showing limited gains beyond that point.

We also evaluated the optimal selection of LLMs over all possible prompting strategies by analyzing the trade-off between model size and F1-score. In Figure 5, the red dashed line represents the Pareto frontier [Ishizaka and Nemery 2013], which identifies the set of models that achieve the best balance between size and performance. Models located on this frontier are considered optimal in terms of efficiency and effectiveness. Our analysis reveals that *llama-3.2-1b-instruct*, *gemma-2-2b-it*, and *mistral-small-24b-instruct-2501* are among the most optimal models. Notably, *gemma-2-2b-it* demonstrates exceptional performance, achieving an F1-score of 0.7269 with only 2 billion parameters, compared to the F1-score of 0.7608 achieved by *mistral-small-24b-instruct-2501*, which utilizes 24 billion parameters. This highlights the efficiency of *gemma-2-2b-it* in delivering competitive performance with significantly fewer parameters.

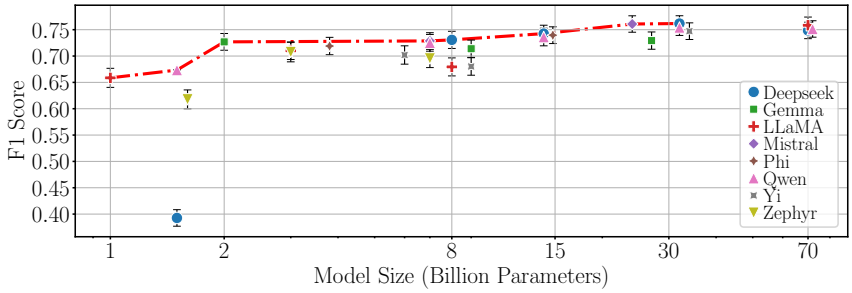

Figure 5. Pareto frontier (red dashed line) analysis of model size versus F1-score, highlighting the optimal trade-off between performance and efficiency. Models on the frontier achieve the best balance between size and performance.

Table 2 presents the F1-Score results, including the standard deviation, for the tested LLMs across the four prompting strategies. Bold values indicate the highest F1-Scores considering the error bars. In general, CoT often yields the best performance, particularly for larger models such as *gpt-4o*, *llama-3.3-70b-instruct*, and *mistral-small-24b-instruct-2501*, as well as for some mid-sized models like *qwen2.5-32b-instruct* and *qwen2.5-14b-instruct*. In certain cases, such as *deepseek-r1-distill-llama-70b*, the Few-Shot strategy (alone or combined) achieves the highest scores.

For smaller models (especially those under $\approx 7B$ parameters), performance varies significantly. In particular, very small models like *qwen2.5-1.5b-instruct* and *yi-6b* exhibit extremely low F1-Scores (below 0.02) in the ZS and CoT strategies, which lack explicit examples (FS). A detailed analysis reveals that these models often fail to generate structured outputs, such as a dictionary containing the "component" attribute, even when correctly identifying the faulty component within unstructured text. This strongly suggests that Few-Shot prompting is essential for smaller models to ensure properly formatted outputs. In contrast, Gemma models demonstrate consistently strong performance across all strategies and sizes, indicating greater robustness to different prompting approaches.

Other results indicate that for the task of hardware fault diagnosis based on user-generated textual descriptions, larger models such as *llama-3.3-70b-instruct*, *mistral-small-24b-instruct*, and *qwen2.5-32b-instruct* achieve the highest performance. However, considering the feasibility of deploying compact models in real-world applications, knowledge distillation from larger models to smaller versions emerges as a promising strategy.

Table 2. F1-Score results with standard deviation for the evaluated LLMs across four prompting strategies. Bold values indicate the highest F1-Scores considering the error bars for each model.

LLM	ZS	FS	CoT	CoT+FS
gpt-4o	0.730 ± 0.016	0.731 ± 0.016	**0.767 ± 0.015**	0.746 ± 0.015
gpt-4o-mini	0.709 ± 0.017	0.709 ± 0.016	**0.752 ± 0.015**	0.729 ± 0.016
deepseek-r1-distill-llama-70b	0.730 ± 0.016	**0.749 ± 0.016**	0.732 ± 0.015	**0.749 ± 0.015**
deepseek-r1-distill-qwen-32b	**0.762 ± 0.015**	**0.758 ± 0.015**	**0.756 ± 0.015**	**0.756 ± 0.015**
deepseek-r1-distill-qwen-14b	**0.733 ± 0.016**	**0.743 ± 0.015**	**0.73 ± 0.016**	**0.731 ± 0.015**
deepseek-r1-distill-llama-8b	0.709 ± 0.016	**0.722 ± 0.016**	0.713 ± 0.016	**0.731 ± 0.016**
deepseek-r1-distill-qwen-1.5b	0.199 ± 0.016	**0.393 ± 0.018**	0.206 ± 0.016	0.328 ± 0.019
gemma-2-27b-it	**0.729 ± 0.016**	**0.724 ± 0.016**	**0.726 ± 0.016**	**0.728 ± 0.016**
gemma-2-9b-it	**0.7 ± 0.017**	**0.704 ± 0.016**	**0.714 ± 0.016**	**0.702 ± 0.016**
gemma-2-2b-it	**0.724 ± 0.016**	**0.727 ± 0.016**	**0.717 ± 0.017**	**0.727 ± 0.016**
llama-3.3-70b-instruct	0.736 ± 0.016	0.739 ± 0.015	**0.758 ± 0.015**	0.739 ± 0.016
meta-llama-3-8b-instruct	**0.664 ± 0.018**	0.650 ± 0.017	**0.667 ± 0.017**	**0.679 ± 0.017**
llama-3.2-3b-instruct	0.684 ± 0.016	0.693 ± 0.017	0.687 ± 0.017	**0.710 ± 0.017**
llama-3.2-1b-instruct	0.608 ± 0.018	0.628 ± 0.018	**0.653 ± 0.018**	**0.659 ± 0.017**
qwen2.5-72b-instruct	**0.749 ± 0.015**	**0.75 ± 0.016**	**0.751 ± 0.015**	**0.749 ± 0.016**
qwen2.5-32b-instruct	0.737 ± 0.015	**0.740 ± 0.016**	**0.754 ± 0.015**	**0.748 ± 0.015**
qwen2.5-14b-instruct	0.709 ± 0.016	**0.722 ± 0.016**	**0.736 ± 0.016**	**0.726 ± 0.016**
qwen2.5-7b-instruct-1m	**0.72 ± 0.016**	**0.721 ± 0.016**	**0.725 ± 0.016**	**0.712 ± 0.016**
qwen2.5-1.5b-instruct	0.000 ± 0.000	**0.673 ± 0.018**	0.651 ± 0.017	0.646 ± 0.018
phi-4	**0.735 ± 0.016**	**0.732 ± 0.016**	**0.74 ± 0.015**	**0.736 ± 0.016**
phi-4-mini-instruct	**0.719 ± 0.016**	**0.717 ± 0.017**	**0.717 ± 0.016**	**0.705 ± 0.016**
zephyr-7b-beta	0.665 ± 0.017	0.664 ± 0.018	**0.696 ± 0.017**	0.636 ± 0.017
stablelm-zephyr-3b	0.474 ± 0.019	**0.708 ± 0.017**	0.475 ± 0.018	0.605 ± 0.017
stablelm-2-zephyr-1_6b	0.431 ± 0.018	**0.618 ± 0.018**	0.437 ± 0.018	0.503 ± 0.019
mistral-small-24b-instruct	0.742 ± 0.015	0.741 ± 0.016	**0.761 ± 0.015**	0.741 ± 0.016
mistral-7b-instruct-v0.3	**0.729 ± 0.016**	0.674 ± 0.017	**0.724 ± 0.016**	0.682 ± 0.017
yi-34b	**0.747 ± 0.016**	**0.744 ± 0.015**	**0.743 ± 0.016**	**0.734 ± 0.016**
yi-9b	**0.680 ± 0.017**	0.660 ± 0.018	0.658 ± 0.018	0.626 ± 0.018
yi-6b	0.272 ± 0.017	**0.702 ± 0.016**	0.013 ± 0.005	0.462 ± 0.019

5. Conclusion

This study presented a comparative evaluation of LLMs addressed to the problem of detecting faulty hardware components, highlighting the importance of prompting strategies in optimizing failure prediction and the impact of model size on predictive performance. The Chain-of-Thought approach, particularly when combined with Few-Shot prompting, proved the most effective by enhancing structured reasoning in model responses. While the tested dataset covers only eight hardware component categories, the observed trends suggest generalizability to other failure types with similar diagnostic patterns. These findings can help manufacturers develop more efficient AI-driven diagnostic systems, reducing downtime, minimizing repair costs, and improving overall user experience. Between the evaluated models, we found that for faulty hardware prediction tasks, the *mistral-small-24b-instruct* combined with the CoT strategy yielded the best results.

As a secondary contribution to this work, we can deliver potential candidates to knowledge distillation to build a specialized smaller LLM that hypothetically performs similarly to a larger model. This distillation process transfers knowledge from

larger LLMs to smaller ones, improving performance while reducing computational requirements. Among the candidates for distillation, although *gemma-2-2b-it instruct* is one of the smallest models evaluated, it delivers a competitive performance compared to larger models. Additionally, larger models such as *llama-3.3-70b-instruct*, *mistral-small-24b-instruct*, and *qwen2.5-32b-instruct* demonstrated a strong performance in the target task, making them promising candidates for knowledge distillation. Since 1B and 2B-parameter models have extremely low VRAM consumption, they enable efficient execution on modern laptops equipped with Neural Processing Units (NPUs). This allows efficient inference with minimal computational resource usage.

For future work, we propose fine-tuning compact models specifically for hardware diagnostics and developing knowledge distillation pipelines to transfer expertise from larger models. A key challenge is generating high-quality labeled examples from high-performing models to improve diagnostic generalization. As open-source LLMs and NPUs continue to advance, failure prediction diagnosis will become more accessible, enabling real-time, on-device diagnostics without relying on cloud services.

Acknowledgments

This research was partially funded by Lenovo, as part of its R&D investment under Brazilian Informatics Law.

References

Abburi, H., Suesserman, M., Pudota, N., Veeramani, B., Bowen, E., and Bhattacharya, S. (2023). Generative ai text classification using ensemble llm approaches. *arXiv preprint arXiv:2309.07755*.

Almeida, F. C. and Caminha, C. (2024). Evaluation of entry-level open-source large language models for information extraction from digitized documents. In *Symposium on Knowledge Discovery, Mining and Learning (KDMiLe)*, pages 25–32. SBC.

Bastos, Z., Freitas, J. D., Franco, J. W., and Caminha, C. (2025). Prompt-driven time series forecasting with large language models. In *Proceedings of the 27th International Conference on Enterprise Information Systems*, pages 309–316.

Efron, B. and Tibshirani, R. (1986). Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75.

Hadi, M. U., Qureshi, R., Shah, A., Irfan, M., Zafar, A., Shaikh, M. B., Akhtar, N., Wu, J., Mirjalili, S., et al. (2023). A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints*, 3.

Ishizaka, A. and Nemery, P. (2013). *Multi-criteria decision analysis: methods and software*. John Wiley & Sons.

Karl, A. L., Fernandes, G. S., Pires, L. A., Serpa, Y. R., and Caminha, C. (2024). Synthetic ai data pipeline for domain-specific speech-to-text solutions. In *Simpósio Brasileiro de Tecnologia da Informação e da Linguagem Humana (STIL)*, pages 37–47. SBC.

Li, Y., He, Y., Lian, R., and Guo, Q. (2023). Fault diagnosis and system maintenance based on large language models and knowledge graphs. In *2023 5th international conference on robotics, intelligent control and artificial intelligence (RICAI)*, pages 589–592. IEEE.

Lotov, A. V. and Miettinen, K. (2008). Visualizing the pareto frontier. In *Multiobjective optimization: interactive and evolutionary approaches*, pages 213–243. Springer.

Makram, M. and Mohammcd, A. (2024). Ai applications in medical reporting and diagnosis. In *2024 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, pages 185–192. IEEE.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142.

Nam, D., Macvean, A., Hellendoorn, V., Vasilescu, B., and Myers, B. (2024). Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Nathani, M., Soni, R., and Mishra, R. (2024). Knowledge distillation in mixture of experts for multi-modal medical llms. In *2024 IEEE International Conference on Big Data (BigData)*, pages 4367–4373. IEEE.

Pereira, F. L. F., Chaves, I. C., Gomes, J. P. P., and Machado, J. C. (2020). Using autoencoders for anomaly detection in hard disk drives. In *2020 international joint conference on neural networks (IJCNN)*, pages 1–7. IEEE.

Queiroz, L. P., Rodrigues, F. C. M., Gomes, J. P. P., Brito, F. T., Brito, I. C., and Machado, J. C. (2016a). Fault detection in hard disk drives based on mixture of gaussians. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 145–150. IEEE.

Queiroz, L. P., Rodrigues, F. C. M., Gomes, J. P. P., Brito, F. T., Chaves, I. C., Paula, M. R. P., Salvador, M. R., and Machado, J. C. (2016b). A fault detection method for hard disk drives based on mixture of gaussians and nonparametric statistics. *IEEE Transactions on industrial informatics*, 13(2):542–550.

Rasal, S. (2024). Llm harmony: Multi-agent communication for problem solving. *arXiv preprint arXiv:2401.01312*.

Silva, M. d. L. M., Mendonça, A. L. C., Neto, E. R. D., Chaves, I. C., Brito, F. T., Farias, V. A. E., and Machado, J. C. (2025). Classification of user reports for detection of faulty computer components using nlp models: A case study.

Silva, M. d. L. M., Mendonça, A. L. C., Neto, E. R. D., Chaves, I. C., Caminha, C., Brito, F. T., Farias, V. A. E., and Machado, J. C. (2024). Facto dataset: A dataset of user reports for faulty computer components. In *Dataset Showcase Workshop (DSW)*, pages 91–102. SBC.

Tao, L., Liu, H., Ning, G., Cao, W., Huang, B., and Lu, C. (2025). Llm-based framework for bearing fault diagnosis. *Mechanical Systems and Signal Processing*, 224:112127.

Wang, L., Bi, W., Zhao, S., Ma, Y., Lv, L., Meng, C., Fu, J., Lv, H., et al. (2024). Investigating the impact of prompt engineering on the performance of large language models for standardizing obstetric diagnosis text: comparative study. *JMIR formative research*, 8(1):e53216.

Zheng, S., Pan, K., Liu, J., and Chen, Y. (2024). Empirical study on fine-tuning pretrained large language models for fault diagnosis of complex systems. *Reliability Engineering & System Safety*, 252:110382.