



UNIVERSITÀ DI PARMA

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale Ingegneria Informatica

Enhancing Fault Isolation in Hardware Systems Using Large Language Models (LLM)

Miglioramento dell'isolamento dei guasti nei
sistemi hardware utilizzando modelli linguistici
di grandi dimensioni (LLM)

Relatore:

Prof. Gianfranco Lombardo

Correlatore:

Tutor aziendale Giuseppe Fiorelli

Tesi di Laurea di:
Matteo Gianvenuti
Matricola: 364979

Table of Contents

Introduction	3
Working Environment	4
Professional context	4
1 State Of The Art (SOTA)	5
1.1 Classical Techniques of Fault Isolation	5
1.2 Hardware Diagnostics Papers with AI	5
1.3 Large Language Models (LLMs)	9
1.4 Generative Pre-trained Transformer (GPT)	10
1.5 Prompt Engineering	10
1.6 Large Language Model Meta AI (Llama)	11
1.7 Phi	12
2. Technological Analysis	13
2.1 Preliminary Models' Comparison	13
2.3 Quantization	15
2.2.1 Bits and Bytes 8-bit Quantization	15
2.4 Complexity of Implementation	16
3. Method	17
3.1 Preliminary Activities	17
3.1.1 Questions Dataset	18
3.1.2 Prompt Structure	18
3.1.3 Textual Representation	20
3.1.4 Json Representation	21
3.1.5 Tabular Representation	22
3.1.6 Model Dimension Limit	23
3.1.7 Models Results	25
3.1.8 PoC Conclusion	27
3.2 Dataset	27
3.2.1 Introduction	27
3.2.2 Database Structure	28
3.2.3 System Definition	29
3.3 Scenarios	29
3.4 ICL	30

3.4.1 System Prompt without Few-shot	30
3.4.2 System Prompt with Few-shot.....	33
4. Results	34
4.1 System Prompt without Few-shot Experiments	34
4.2 System Prompt with Few-shot Experiments.....	37
4.2.1 Two-shot experiments	42
4.3 Connectivity experiments without few-shot	44
4.4 PBIT tests with a smaller system without few-shot	47
5. Conclusion.....	55
Bibliography	56

Introduction

I developed this work thesis during an internship program in MBDA ITALIA SPA. MBDA ITALIA SPA, shortly called company in this work thesis, is a multinational corporation that works in defence. The internship does not regard the development of weapons.

The hardware systems evolution led to an increasing complexity of the diagnostics processes and architectures. In industrial and mission-critical environments, like the defence, the ability of quickly identify and isolate faults is fundamental to guarantee reliability, security and availability. However, the traditional methods of fault isolation are usually and typically based on static rules, expert systems, manual analysis and instruments limited in unstructured data analysis. In parallel, Large Language Models (LLMs) have demonstrated a surprisingly capacities in comprehension of the natural language and reasoning tasks. Especially in the case in which models know the environment in which they operate.

In this work thesis we want to explore the use of LLMs in hardware diagnostic to analyse diagnostics test results, identify any possible anomalies and try to find their possible causes.

We want to reach these objectives in an offline working environment characterized by challenging limited resources, like the resources available on computer laptop. In particular, this work thesis explores the efficacy of a diagnostic framework based on a LLM executed locally without the support of cloud infrastructures or high-end GPU. This approach results particularly interesting in industrial scenarios in which the connection is absent, not guaranteed or the access to advanced hardware resources is constrained by hardware or budget requirements. The choice of operating in a restricted hardware environment does not represent a limitation but rather a strategic experimental condition to evaluate the scalability and adaptability of LLMs in real and decentralized contexts.

To reach that objectives we structured the work thesis in two steps. A first, consisting in verify if the LLMs, appropriately “trained” with In-Context Learning (ICL) techniques digest and understand the structure and the functioning of a complex hardware system. A second, consisting in evaluating the LLMs ability of analyse logs and telemetry data generated by the diagnostic software of the company to identify anomalies and their causes.

To carry out the work activities of the second step it was necessary to create a simulation environment with various scenarios and a specific dataset for each scenario. We also implemented the Open WebUI graphical interface to provide a better experience and usage of the LLM based solution.

The local execution of LLMs guarantee privacy and control of the test environment.

This work was based on ICL and Prompt Engineering to reach the cited objectives with limited resources. We did not a fine-tune.

Working Environment

Since I had to develop an AI project with few resources for this work thesis, the company assigned me a ZBOOK HP laptop with a small GPU (NVIDIA GA104GLM RTX A3000 Mobile GPU with 6GB of memory). That laptop has the CentOS 9 operative system. Some colleagues (Giuseppe and Alessandro) configured the Finx OS on a virtual machine through Virtual Box. It was necessary to use the diagnostics software tool of the company (HMI).

To use the GPU, I installed the necessary driver and the Miniconda software. I installed also Python. Then I created a virtual environment with all the libraries needed for this work. After the preliminary activities, I installed and configured Ollama for the backend and Open WebUI for the frontend.

Miniconda is a reduced version of Anaconda. It includes conda, Python and other minor packages. Instead, Ollama is a software that allow you to download and run LLMs models locally. While Open WebUI is a web user interface like the ChatGPT web interface that can run also locally. With it you can add and configure a model. So, you can set a system prompt as many other parameters.

Professional context

The company is a European multinational corporation specialized in design, development and manufacturing of missile and related systems.

The company was founded in December 2001 by the joint of three missile systems companies: the French Aérospatiale Matra Missiles, the Anglo-French Matra BAe Dynamics and the missile division of the Italian Alenia Marconi Systems. The company maintained a national country division since its creation: MBDA France, MBDA UK and MBDA Italy. The company employs around 13 thousand people around the world. For the Italian division around two thousand.

The Italian division is located in three Italian cities: La Spezia that is focused on naval missiles and navy activities; Rome that is the Italian headquarter that is focused on the software creation; and Fusaro where the hardware system is build.

I done my internship for this work thesis in the Rome headquarter. In particular, my workstation was located in a computer laboratory called IRAD which focuses on software development and innovation, typically in diagnostics. This laboratory is access restricted. Then, I needed an authorization.

1 State Of The Art (SOTA)

1.1 Classical Techniques of Fault Isolation

The hardware diagnostics is mainly based on hardware signals processing, hardware test analysis, and more in general analysis of sensor measurements data.

The more classical systems are based on static rules, the so-called expert systems, these systems have clear limitations in a dynamical context because is necessary to specify a rule for each use case.

While there are few main abstract uses of AI in hardware diagnostics, however the AI can be applied to any case in which is necessary to recognize patterns. A first use case based on predictive techniques is an AI system that predict the possible measurements to compare them with the real values measured from the hardware to identify anomalies by a high divergence. Another use of AI is a classification system that identify and classify anomalies by analysing the telemetry data. This type of system could be integrated in an expert system to match the anomalies found through rules to the cause corresponding to the identified error if known.

Most of the classical techniques are limited to the anomaly identification, or expert systems that require the writing of a lot of static rules while with this work thesis we want to build a framework that is also able to identify an anomaly and its possible causes in the most general way without the need of matching rules or other forms of expert systems.[\[1\]](#)

1.2 Hardware Diagnostics Papers with AI

In this chapter, I analyse the SOTA papers and articles about the diagnostics and log analysis based on AI, machine learning and deep learning.

- “LLM4SecHW: Leveraging Domain-Specific Large Language Model for Hardware Debugging” (paper 2024): This paper realizes, a fine-tune of various models to help in identifying hardware bugs during the design phase. The fine-tune is based on a dataset built on a GitHub repository of machine code.

This system has two workflows, one to identify a bug and another to generate a possible solution. The solution is compared to the one in the dataset, through the Rouge-N F1 score. Since the Rouge metric cannot capture all the LLMs skills, the authors manually evaluated the models replies. The authors experimented fine-tuned and not fine-tuned models. The fine-tuned models have more specific information then they were able to complete the authors tasks in all cases while the not fine-tuned in some cases affirmed, they were unable to complete their tasks, in other cases they just identified the problem without proposing a solution.

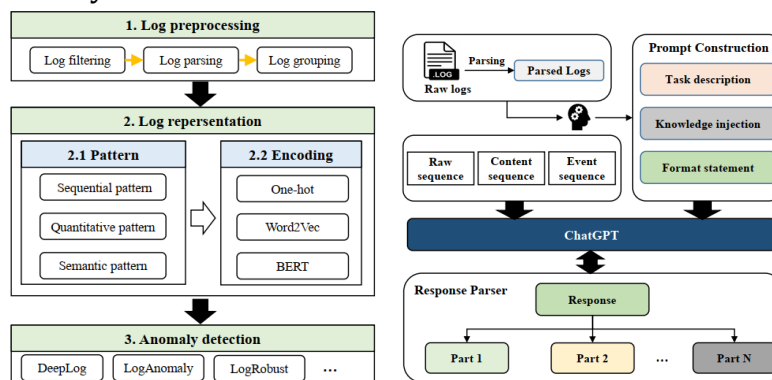
In conclusion, with more data is possible to reach better results.[\[2\]](#)

This paper is based on the prompt engineering and so ICL. A prompt with the task description, optional human knowledge and then the log information: context, events, sequence of events and their meaning.

Some important conclusions of the authors:

- The prompt structure and human knowledge are fundamental to improve the results
- Specifying the cause of an error helps the model in understanding the logic link from the effects rather than simply labelling the errors
- Adding some solved examples to the prompt (few-shot) lead to better performance than without examples (zero-shot)
- ChatGPT has a high false positive rate
- ChatGPT is too much sensible to small changes in the prompt

Overall, LLMs as the other AI methods can identify patterns better than rule-based systems.[\[5\]](#)



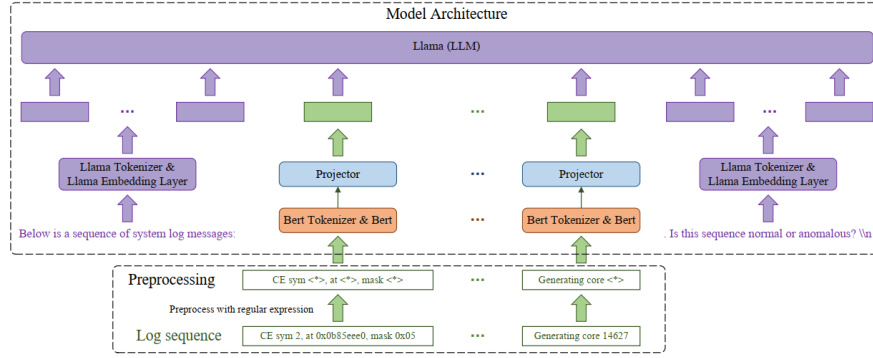
- “LogLLM: Log-based Anomaly Detection Using Large Language Models” (paper 2024): This paper proposes a system that automatically collects logs in a structured way for a real time analysis to identify anomalies with Llama3-8B after a pre-processing based on BERT-base and a projector. BERT-base is used to extract semantic vectors, to be tokenized with the BERT tokenizer. While the projector is used to map the semantic vectors previously tokenized to another tokenized representation understandable by Llama. Finally, Llama is used to do the log analysis.

A custom prompt is used to introduce the log’s tokenized representation and to request the task to Llama.

The whole model system is finetuned in three steps. One to teach to Llama to reply by saying if a log is anomalous or normal. A second to train BERT-base and the projector to find the best token representation for Llama. And finally, a third step in which BERT-base, projector and Llama are finetuned as a single system. The finetunes are done with the QLoRA optimization technique.

In conclusion, this system maintains a low rate of false positives but there are also some disadvantages it has a high computational cost due to the high number of parameters. As this system limits the power of an LLM to a binary classification. The model cannot handle unbalanced scenarios.

The authors experimented also a finetune on the pure logs, but they had a worsening in performance.[\[6\]](#)



- “An Automated Machine Learning Approach for Real-Time Fault Detection and Diagnosis” (article 2022): This paper proposes a system for real-time fault detection on industrial machines based on ML. This system uses only commonly available data in industrial systems. The method was tested in a 3D machine simulation system in faulty and non-faulty conditions. The system consists in a classifier trained in supervised learning to identify known faulty situations. The system is able to do re-generation with new situations.

The authors experimented with sixteen classifiers to find the best. They selected Extra trees and Random Forest as final classifiers with an F1 score of 100% and 85% respectively for the two “Furnace” and “Pick And Place” simulated industrial machines.

The authors observed that increasing the examples lead to a good improvement in results. The F1 score reached more than 90% with more samples. Also, thanks to an ablation study on the features that allowed to increase the performance up to 6%.[\[7\]](#)

- “Machine Learning for Anomaly Detection: A Systematic Review” (paper 2021): This paper analyses a collection of other publications in years 2000-2020 about anomaly detection with ML. The authors systematically analysed all the publications with the “Kitchenham and Charters methodology”.[\[8\]](#)

According to the authors, the ML papers can be divided in the following techniques: classification, ensemble, rule system, clustering and regression. Those techniques are applied in two forms as standalone or hybrid (their combination) models. Unsupervised anomaly detection has been adopted more than classification techniques.

In conclusion, the article consists in a comparison of the ML papers in that period. Practically is a comparison of the other methods, it shows for each period which technique was mostly used and their performances. The most used technique in anomaly detection is based on unsupervised learning while the second most used is the classification even if, according to the authors of this analysis, the papers’ authors did not mention the classification type applied in many cases. While the semi-supervised learning technique is rarely used in anomaly detection. Many papers did not mention also the extraction/selection type for the features.

This publication limits its analysis to strictly scientific papers/articles only.[\[9\]](#)

Actually, the SOTA is focused on automated anomaly detection only rather than find an anomaly and its causes. Our work is different in this sense; we try to find also their possible causes with LLM.

1.3 Large Language Models (LLMs)

LLMs are machine learning models with many parameters, designed for natural language processing (NLP) tasks, especially text generation. These models are typically built using deep neural networks, particularly transformer architectures, and are trained on massive corpora of textual data. They are trained with self-supervised learning on a vast amount of text. Their strength lies in the ability to capture complex syntactic, semantic, and contextual patterns, allowing them to perform a wide range of tasks such as translation, summarization, question answering, and reasoning.

LLMs can be fine-tuned for specific tasks or guided by prompt engineering, as done in this work. These models acquire predictive power regarding syntax, semantics, and ontologies inherent in human language corpora, but they also inherit inaccuracies and biases present in the data they are trained in.

The core idea behind LLMs is to learn a statistical model of language by predicting the next word (or token) in a sentence, given a context. This is achieved using self-supervised learning, where the model learns from raw text without the need for labelled data. During training, the model builds internal representations that encode meaning, structure, and dependencies across sequences of words.

The largest and most capable LLMs are generative pre-trained transformers (GPTs).

Modern LLMs like OpenAI's GPT and DeepSeek's DeepSeek series or Meta's LLaMA models are characterized by having billions or even trillions of parameters. The performance of these models tends to improve significantly as the number of parameters and the size of the training data increase, a phenomenon often referred to as "scaling laws."

In addition to linguistic capabilities, recent LLMs demonstrate emergent properties such as reasoning, commonsense inference, and even basic code generation. However, they also exhibit limitations, including sensitivity to prompt phrasing, hallucination of facts, and the presence of biases derived from the training data. So, in the context of training LLMs, datasets are typically cleaned by removing low-quality, duplicated, or toxic data. Cleaned datasets can increase training efficiency and lead to improved downstream performance.

As all the others machine learning (ML) algorithms, LLMs process numbers rather than text, the text must be converted to numbers through the tokenization process. Tokenization is a fundamental process in LLM pipelines. Before text is processed by a model, it is converted into tokens. In the first step, a vocabulary is decided upon, then integer indices are arbitrarily but uniquely assigned to each vocabulary entry, and finally, an embedding is associated to the integer index.

Most results previously achievable only by (costly) fine-tuning, can be achieved through prompt engineering, although limited to the scope of a single conversation (more precisely, limited to the scope of a context window). Due to the high training

costs and the huge amount of data needed, we did not implement a fine-tune, instead we worked in the field of ICL. Also, because it is already sufficient to achieve our goals.

In our thesis work, LLMs serve a dual role: first, as systems capable of understanding the architecture and components of hardware environments through descriptive prompts; second, as tools capable of interpreting system logs and telemetry to support fault diagnosis.[\[10\]](#)

1.4 Generative Pre-trained Transformer (GPT)

A GPT is a type of LLM and a prominent framework for generative AI. It is an artificial neural network that is used in NLP by machines. It is based on the transformer deep learning architecture, pre-trained on large datasets of unlabelled text, and able to generate novel human-like content. As of 2023, most LLMs had these characteristics and are sometimes referred to broadly as GPTs.

The first GPT was introduced in 2018 by OpenAI. OpenAI has released significant GPT foundation models (models trained on board data at scale such that they can be adapted to a wide range of downstream tasks) that have been sequentially numbered, to comprise it “GPT-n” series. Each of this was significantly more capable than the previous, due to increased size (number of trainable parameters) and training. Such models have been the basis for their more task specific GPT systems, including models fine-tuned for instruction following. GPT is based on the transformer architecture.[\[11\]](#)

1.5 Prompt Engineering

Prompt engineering is the process of structuring or crafting an instruction in order to produce the best possible output from a generative AI model.

In case of text-to-text language models a *prompt* is natural language text describing the task that an AI should perform. It can be a query, a command, or a longer statement including context, instructions, and conversation history. Prompt engineering may involve phrasing a query, specifying a style, choice of words and grammar, providing relevant context, or describing a character for the AI to mimic.

There are many different types of prompt engineering:

- Chain-of-Thought (CoT): it is a reasoning technique that goes through several intermediate steps before giving a definitive answer to a question.
The CoT prompting improves reasoning ability by inducing the model to answer a multi-step problem with steps of reasoning that mimic a train of thought. These techniques were developed to aid common sense reasoning and multi-step reasoning.
The prompt specifies some examples of questions and answers (Q&A); this improved the results considerably. Even just writing "let's think step by step"

improved the results a lot. Allowing you to generalize and avoid writing many specific examples that consume the context.

- In-Context Learning (ICL): it is the ability of models to learn information temporarily. The information of interest is specified in the prompt, typically in the initial part. This information is then used by the model in the session to answer questions by generating text, this allows the model to be trained without necessarily having to perform fine-tuning.

The weights of the network are not updated, so the information is temporary. Starting a new instance the model does not retain this knowledge.

- Prompting to estimate model sensitivity: research consistently demonstrates that LLMs are highly sensitive to subtle variations in prompt formatting, structure, and linguistic properties. Linguistic features significantly influence prompt effectiveness—such as morphology, syntax, and lexico-semantic changes—which meaningfully enhance task performance across a variety of tasks.

To address sensitivity of models and make them more robust, several methods have been proposed. Format Spread facilitates systematic analysis by evaluating a range of plausible prompt formats, offering a more comprehensive performance interval. Similarly, Prompt Eval estimates performance distributions across different prompts, enabling robust metrics such as performance quantiles and accurate evaluations under constrained budgets.

- Retrieval-augmented generation (RAG): it is a technique that enables generative AI models to retrieve and incorporate new information. It modifies interactions with a LLM so that the model responds to user queries with reference to a specified set of documents, using this information to supplement information from its pre-existing training data. This allows LLMs to use domain-specific and/or updated information.
- Few-shot: the model is already pre-trained and fine-tuned to become instruct. Few-shot means that in inference in the prompt are put from zero to 4/5 task examples before the request for an actual task.

There are many other techniques, but they are not relevant to this work. In this work we mainly focused on ICL and CoT, building a customized few-shot technique.[\[12\]](#)

1.6 Large Language Model Meta AI (Llama)

Llama, also known as LLaMA, is a family of open-source LLMs released by Meta AI. In the last years Meta AI released many models with various parameters size.

Furthermore, they released also fine-tuned versions (instruct). The Llama model demonstrated to be stronger than many other models in NLP tasks. Llama is able to generalize patterns and follow detailed instructions better than other models like Phi.

The Llama model architecture is autoregressive decoder-only transformer, similar to GPT-3 with minor differences in activation function, embeddings, and optimizer. So, Llama is based on the transformer architecture, then it is composed of many transformer layers. Transformers use self-attention mechanisms to process input sequences in parallel, allowing for efficient training and high-quality language modelling.[\[13\]](#)

1.7 Phi

Phi is a series of small open-source AI models developed and published by Microsoft. There are two types of models: normal and reasoning. Phi models can be also base or instruct depending on the training. There are also reduced versions called “mini”.

The Phi-4 series are strong in reasoning and logics especially in math, due to a mix of unsupervised learning techniques such as multi-agent prompting and self-revision workflows that constitute the “synthetic data” approach.

Thanks to the innovative synthetic data generation methods, this small model can reach stronger performance than others size-equivalent and some bigger models. The synthetic data method is applied also in post training to refine the model output.

The development of Phi-4 is guided by three core activities:

1. Synthetic data (in pre-training and mid-training): thanks to appositely designed datasets of synthetic data focused on reasoning and problem solving, Phi can reach its strong results.
2. Data curation and filtering: the datasets contents are meticulously selected and filtered also for the material taken from the internet to promote educational values and obviously to have good datasets.
3. Post training: here are applied various techniques based on the pivotal token search.

Nevertheless, its strong performance, Phi has some weaknesses mainly due to the model size. Specifically in hallucinations regarding factual knowledge, rigorously following detailed instructions for a specific task and generate the output according to a given pattern.[\[14\]](#)

2. Technological Analysis

According to the thesis objectives, the technological analysis is mainly driven by the challenging objective of using limited hardware resources and the need of models able to properly understand and reason on the natural language. Consequently, in this work thesis I experimented only models with less than seven billion parameters.

2.1 Preliminary Models' Comparison

The model selection was driven by the necessity of operating in an offline environment with few resources, and open-source models.

Initially, I compared various models such as Llama, Gemma, Phi, Mistral, DeepSeek and Qwen. I compared them in terms of performances according to different paper benchmarks and size due to hardware limitations.

In this comparison, I directly discard ChatGPT due the monetary cost of the API unwanted by the company and the fact that the software is covered by license and not open source. As I directly discarded DeepSeek due to its enormous size on Hugging Face. I directly discarded also Qwen due to its “intermediate size” (8B+) on Hugging Face.

During the experiments on the GPU done in the [preliminary activities](#), I noticed the available hardware did not support the model Phi-4-mini-Instruct even with 4-bit quantization. While Gemma-3-4B-it was not compatible with the virtual environment due to some vision libraries dependency. So, I practically experimented with Llama-3.2-3B-Instruct, Phi-4-mini-reasoning and Mistral-7B-Instruct-V0.3.

During the experiments I noticed the model Llama is the best in understanding the natural language — compared to the others cited. In fact, this model is able to easily generalize a sentence. This because easily understood the system prompt instruction and strictly followed it while the other models had some lacks.

Instead, Phi-4-mini-reasoning is stronger than the others in logics. This model reached the best results in the preliminary activities. Even if sometimes the hardware did not support it, or the model did not answer to all the questions. This because reasoning models are designed to be good at complex tasks such as solving puzzles, advanced math problems, and challenging coding tasks. However, they are not necessary for simpler tasks like summarization, or knowledge-based question answering. In fact, using reasoning models for everything can be inefficient and expensive. For instance, reasoning models are typically more expensive to use, more verbose, and sometimes more prone to errors due to “overthinking”. In fact, this model required triple time than Llama.

While the last model Mistral-7B-Instruct-V0.3 has the worst performances, probably due to the 4-bit quantization that make its results widely instable.

For these reasons and the results obtained during the *preliminary activities*, I decided to use the Llama model for all phases after the preliminary activities.

The preliminary activities involved Llama-3.2-3B-Instruct, Mistral-7B-Instruct-V0.3 and Phi-4-mini-reasoning.

Model	Parameters	Input-output modalities	Context length	Provenience	Author
Llama-3.2-3B-Instruct	3B	Text-to-text	128k	Hugging Face	Meta AI
gemma-3-4B-it	4B	Text/Image-to-text	128k	Hugging Face	Google DeepMind
Mistral-7B-Instruct-V0.3	7B	Text-to-text	32,768	Hugging Face	Mistral AI
Phi-4-mini-reasoning	3.84B	Text-to-text	128k	Hugging Face	Microsoft
Phi-4-mini-Instruct	3.84B	Text-to-text	128k	Hugging Face	Microsoft

Model	Reasoning model	Inference time	NL Comprehension	Deductive capacity
Llama-3.2-3B-Instruct	No	0-5 seconds	5	3.5
gemma-3-4B-it	No	-	-	-
Mistral-7B-Instruct-V0.3	No	0-3 seconds	2	2
Phi-4-mini-reasoning	Yes	Up to 30-90 seconds	3	5
Phi-4-mini-Instruct	No	-	-	-

For the natural language (NL) comprehension and the deductive capacity I assigned a mark according to practical experimentations in a scale 1-5. Note that the inference time mostly depends on the input size and the quantization approximation. In the table is reported the inference time on a single question.

Llama clearly understands the NL more than any other experimented due to its extensive training on text datasets, it strictly followed all the instructions. While get lost in few difficult reasoning/deductive tasks especially with more data. Phi sometimes did not follow the instructions but always operated in consistent way, but it solved almost all the logic questions (this because it is a reasoning model). While Mistral had difficulties in both the activities of NL comprehension and deductive capacity due to the approximation introduced by the 4-bit quantization.

2.3 Quantization

The quantization was necessary to support the models in a limited GPU memory.

The quantization is the process of reducing the memory space used by a number. It reduces the precision by approximating the model weights and then reducing the model size and consequently making faster the model usage. The quantization makes the model results a bit more unstable, as is possible to see in the results of the preliminary activities, especially for the Mistral model.

There are two main quantization techniques with various implementations.

- Post- training quantization: it reduces the precision after the model training for the inference usage
- Quantization aware training: it quantizes the model already during the training and reduces the performance degradation

I used the post-training quantization in this work thesis with the BitsAndBytesConfig class by the transformers library.[\[15\]](#)

2.2.1 Bits and Bytes 8-bit Quantization

Starting from float16 matrices X for the input state and W for the weights state. To quantize, values are extracted column-wise from X forming the matrix C_x . While from W are extracted row-wise forming the C_w . Then with the following formula the two matrices can be quantized in 8-bit.[\[15\]](#)

$$X_{i8} = X * (127/C_x)$$

$$W_{i8} = W * (127/C_w)$$

Then, the two matrices X_{i8} and W_{i8} are multiplied resulting in Out_{i32} . To obtain a partial output for the inference is applied the following formula.

$$\text{Out}_{F16} = (\text{Out}_{i32} * (C_x \otimes C_w)) / 127^2$$

The remaining values from X and W build two new matrices that are multiplied to obtain a second partial output Out_{FP16} .

The final output matrix is the xor between the two partial outputs.

$$\text{Out} = \text{Out}_{F16} \oplus \text{Out}_{FP16}$$

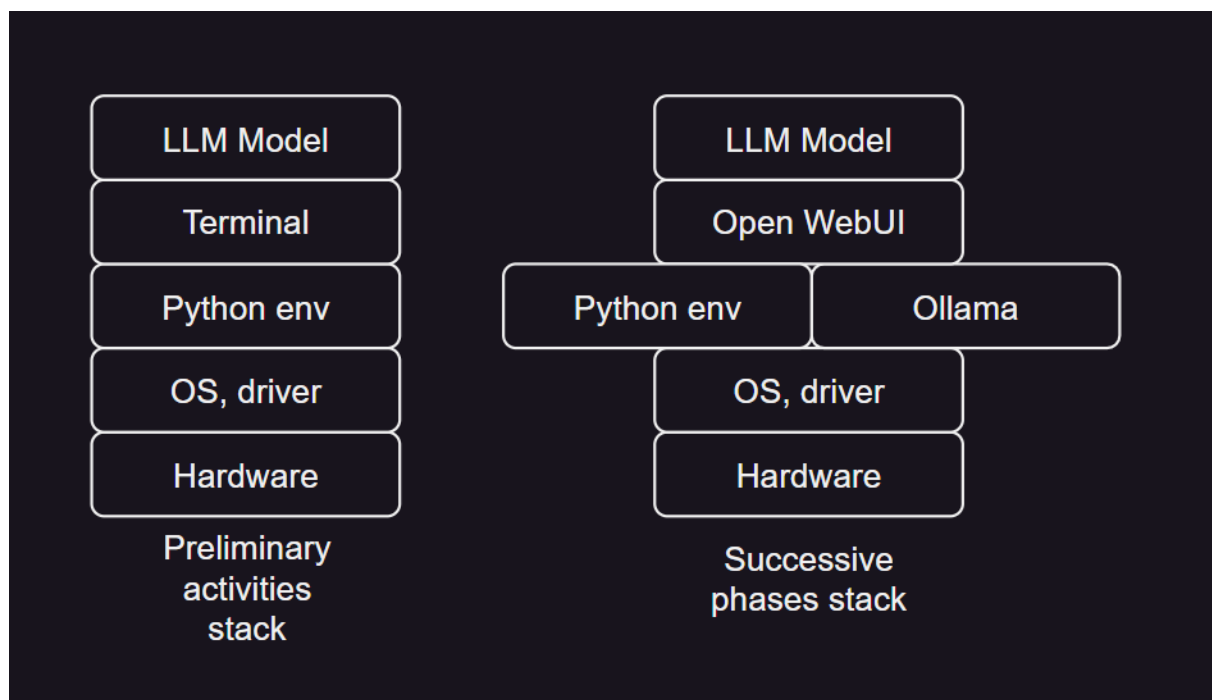
2.4 Complexity of Implementation

All the models required to install the necessary Nvidia drivers to use the GPU, Miniconda and Python for a virtual environment.

All the models can be used through a Python script with their setup and the necessary libraries. During the preliminary activities I used the Python “input” function to grasp the user input from a terminal. In particular, for the preliminary activities I experimented the models with the library transformers by Hugging Face.

After, for all the successive phases, I implemented the Open WebUI to grasp the user input. It required the installation of Ollama for the backend to handle models. Then for all the successive phases I used Llama from Ollama instead of Llama by Hugging Face used for the first phase. I choose Ollama for the backend for its simplicity and integration with the Open WebUI interface. Ollama provides more models than the available on Hugging Face.

Follows an overview of the stack.



3. Method

3.1 Preliminary Activities

The purpose of these first activities is to evaluate the ability of LLMs in understanding the hardware structure of a sample system and its interconnections, and to determine whether representation of the system architecture is more effective in making the model understand the system architecture. This is therefore an initial proof of concept (PoC).

I have experimented three system architecture representations: textual, JSON and tabular representation of the network architecture. All these representations are always in text format since LLMs only understand text.

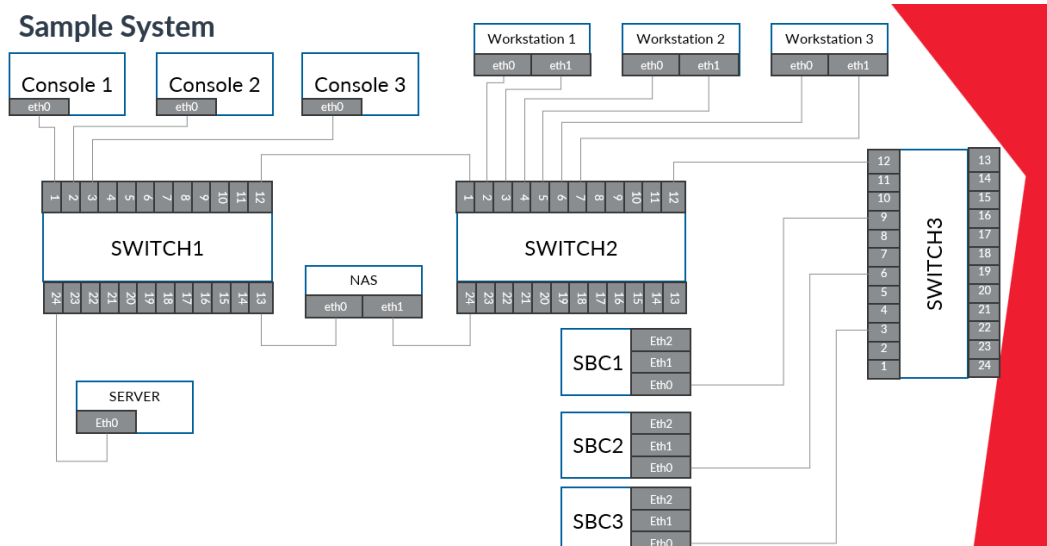
We have experimented with various models according to the available hardware capacity for this phase, especially meta-llama/llama3.2-3B-Instruct and microsoft/Phi-4-mini-reasoning.

According to the thesis objectives I had to use a limited hardware resource. Consequently, I had to perform the tests with small models and the use of 8-bit and 4-bit quantization to stay on the available memory.

To test the models on the understanding of the connection topology (the architecture of the System) I created a dataset of 94 questions, in which it is asked what is connected from port/interface A to B, and vice versa. From this dataset I extract a subsample of questions to ask the model to verify if it has "understood" and remembers the architecture. Since the GPU used in this phase has 6GB of memory, it is not possible to ask all the questions together.

Since the LLMs answer open questions with generally always different and long text, I taught the model (specifying rules in the system prompt) to answer by indicating only the name of the connected component. In this way it is possible to automatically parse the answers to evaluate the accuracy.

The system prompt consists of a short description of the task that the model must perform and its role, a set of rules on the representation of the architecture and the names of the devices, a list of the devices, the representation itself in its format and finally some examples of questions and answers. This PoC is therefore based on a combination of the prompt engineering techniques shown in the previous chapter.



3.1.1 Questions Dataset

I manually built a dataset with 94 questions and corresponding answers about the system interconnections topology. The dataset contains questions like “What is connected to A?” for all ports and interfaces of all devices. So, if A is connected to B there is the dual question (the answer is B when asking about A and vice versa). In the dataset there are also few questions about components that do not exists. The responses are of the type "The C is connected to D", "Nothing" if there is nothing connected or "The interface/port D does not exist for device G".

To calculate the accuracy, a subset of questions is randomly extracted and asked to the model. Due to GPU memory limitation, I could not ask the models all questions at once.

I calculate the accuracy in reporting the asked questions and in correctly responding to questions. All the models can report the questions (the accuracy was always 100%), so they know the request but in answering/reasoning they had more difficulties.

3.1.2 Prompt Structure

The prompt is specific for each type of representation. At the beginning of the prompt is defined the model task. In Json and tabular representations, the task description varies slightly as I specify what type of representation is shown.

```
Your task:
You will be given a network topology. Your job is to understand the structure and
answer the questions that follow about the system and its architecture.
Use the format shown in the examples to give your answers, do not add additional,
inconsistent or unrelated information.
```

In the successive part initially, I put the components definitions

Definitions:

Console: A computer equipped with a monitor, keyboard, and mouse, allowing operators to interact with a system.

Switch: A networking device that connects multiple computers or devices, efficiently managing data traffic within a network.

Workstation: A high-performance computer.

NAS: (Network Attached Storage) A dedicated storage device connected to a network, allowing multiple users to store and access files remotely.

Server: A powerful computer that provides services or resources to other computers, managing data, applications, or networks.

After some experiments I removed the definitions because it was not helpful in interconnections understanding and the size of the prompt matter.

In the successive part there are the naming conventions and other rulers depending on the representation and the representation itself.

In the last part of the prompt, I put some Q&A examples. This teaches the model to respond in the specified format. At this point of the work, it is mainly useful to parse the answers and automatically evaluate the accuracy.

Not all models clearly understand that they must respond in the specified way. So, I added a parenthetical note to the prompt to clarify it, but models like mistralai/Mistral-7B-Instruct-v0.3 sometimes still do not follow it, while meta-llama/llama3.2-3B-Instruct already understood it correctly from the start.

Here the last part of the prompt, notice that the last generalized sentence allows you to narrow down the space of possible output values and therefore narrow the field to just your domain of interest. This is powerful.

```

Q&A examples:
What is connected to Switch-2-port-5?
The Workstation-2-eth1.

What is connected to Console-3-eth0?
The Switch-1-port-3.

What is connected to NAS-eth1?
The Switch-2-port-24.

What is connected to Switch-3-port-6?
The SBC-2-eth0.

What is connected to Workstation-3-eth2?
The eth2 network interface does not exist for Workstation-3.

What is connected to SBC-2-eth1?
There is nothing connected to SBC-2-eth1.

What is connected to Switch-2-port-12?
The Switch-3-port-12.

What is connected to Switch-1-port-14?
There is nothing connected to Switch-1-port-14.

<Any other sentence/phrase/question not related to this system>
Sorry, I can support you only in tasks related to this system.

```

3.1.3 Textual Representation

In this representation the system interconnections are described in natural language. Before the description of the system, I put some naming rules and a list of all devices.

```

Naming rules: All components with numbers in their names (e.g., Switch-1, Console-3, SBC-2) are distinct devices. The number is critical for identification.
- Multi-device components: Use '{device}-{device number}' (e.g., Switch-1, Console-3).
- Single device components: Use the base name (e.g., Server, NAS).
- Network interfaces: Append '-{interface}' (e.g., Workstation-1-eth0, NAS-eth1).
- Switch ports: Use '{Switch}-{switch number}-port-{port number}' (e.g., Switch-1-port-14, Switch-3-port-7).

Network components:
Switch-1, Switch-2 and Switch-3 with 24 ports each.
Console-1, Console-2 and Console-3 each with its own network interface eth0.
Workstation-1, Workstation-2 and Workstation-3 each with its own two network interfaces eth0 and eth1.
Server with its own network interface eth0.
NAS with its two network interfaces eth0 and eth1.
SBC-1, SBC-2 and SBC-3 each with its own network interfaces eth0, eth1 and eth2.

```

Here the system representation:

```
Network topology:
The Console-1-eth0 is connected to Switch-1-port-1.
The Console-2-eth0 is connected to Switch-1-port-2.
The Console-3-eth0 is connected to Switch-1-port-3.
The Server-eth0 is connected to Switch-1-port-24.
The NAS-eth0 is connected to Switch-1-port-13.
The NAS-eth1 is connected to Switch-2-port-24.
The Switch-1-port-12 is connected to Switch-2-port-1.
The Switch-3-port-12 is connected to Switch-2-port-12.
The Workstation-1-eth0 is connected to Switch-2-port-2.
The Workstation-1-eth1 is connected to Switch-2-port-3.
The Workstation-2-eth0 is connected to Switch-2-port-4.
The Workstation-2-eth1 is connected to Switch-2-port-5.
The Workstation-3-eth0 is connected to Switch-2-port-6.
The Workstation-3-eth1 is connected to Switch-2-port-7.
The SBC-1-eth0 is connected to Switch-3-port-9.
The SBC-2-eth0 is connected to Switch-3-port-6.
The SBC-3-eth0 is connected to Switch-3-port-3.
All unspecified device, ports and interfaces are not connected to any device.
```

3.1.4 Json Representation

In this representation I described the network and its components as Json objects. I introduced this representation in the prompt as there:

```
Naming rules:
Devices with numbers (e.g., Switch-1, Console-3, SBC-2) are different from each other.
Interfaces are named like {device}-{interface} (e.g., Switch-1-port-14, Server-eth0).
Connections are bidirectional. If one device says it's connected to another, the other must also list the same connection.

Interface connection rules:
If connected_to is null, the interface is not connected.
If connected_to has a value, it tells which device and interface it is connected to.
To find what is connected to something like SBC-1-eth2, search all devices to check if any interface points to: "device": "SBC-1" and "interface": "eth2"

Data format:
The network is described in JSON. At the top level, there is a "devices" list.
Each device has: a unique "name", a "type" (like Switch, Server, etc.) and an "interfaces" list.
Each interface has: a "name" (e.g., eth0, port-2) and a "connected_to" field, which can be null or an object with "device" and "interface".
```

Here an example of one component in Json format, the switch-1:

```

{
  "devices": [
    {
      "name": "Switch-1",
      "type": "Switch",
      "interfaces": [
        { "name": "port-1", "connected_to": { "device": "Console-1", "interface": "eth0" } },
        { "name": "port-2", "connected_to": { "device": "Console-2", "interface": "eth0" } },
        { "name": "port-3", "connected_to": { "device": "Console-3", "interface": "eth0" } },
        { "name": "port-4", "connected_to": null },
        { "name": "port-5", "connected_to": null },
        { "name": "port-6", "connected_to": null },
        { "name": "port-7", "connected_to": null },
        { "name": "port-8", "connected_to": null },
        { "name": "port-9", "connected_to": null },
        { "name": "port-10", "connected_to": null },
        { "name": "port-11", "connected_to": null },
        { "name": "port-12", "connected_to": { "device": "Switch-2", "interface": "port-1" } },
        { "name": "port-13", "connected_to": { "device": "NAS", "interface": "eth0" } },
        { "name": "port-14", "connected_to": null },
        { "name": "port-15", "connected_to": null },
        { "name": "port-16", "connected_to": null },
        { "name": "port-17", "connected_to": null },
        { "name": "port-18", "connected_to": null },
        { "name": "port-19", "connected_to": null },
        { "name": "port-20", "connected_to": null },
        { "name": "port-21", "connected_to": null },
        { "name": "port-22", "connected_to": null },
        { "name": "port-23", "connected_to": null },
        { "name": "port-24", "connected_to": { "device": "Server", "interface": "eth0" } }
      ]
    }
  ],
}

```

3.1.5 Tabular Representation

As in all the other representation there is a part with rules and the components list:

Critical Rules:

1. Connections are bidirectional: If one device says it's connected to another, the other must also list the same connection. Every connection appears twice in the table (A→B and B→A).
2. Interface validation:
 - If an interface does NOT exist (e.g., Workstation-3-eth2), answer: "The {interface} network interface does not exist for {device}".
 - If an interface exists but is unconnected, answer: "Nothing".
3. Naming syntax:
 - Devices with numbers (e.g., Switch-1, Switch-2, Console-3, SBC-2) are different from each other, especially devices of the same type.
 - Interfaces are named like {device}:{interface} (e.g., Switch-1:port-14, Server:eth0).

Network components:

Switch-1, Switch-2 and Switch-3 with 24 ports each.

Console-1, Console-2 and Console-3 each with its own network interface eth0.

Workstation-1, Workstation-2 and Workstation-3 each with its own two network interfaces eth0 and eth1.

Server with its own network interface eth0.

NAS with its two network interfaces eth0 and eth1.

SBC-1, SBC-2 and SBC-3 each with its own network interfaces eth0, eth1 and eth2.

Unspecified components do not exist. While existing components, not specified in the following connection table, are not related to other devices/interfaces.

After the system architecture topology is in a tabular format. The table is manually hardcoded, since it must be text:

Connections table:

Device	Interface	Connected to	Connected interface
Switch-1	port-1	Console-1	eth0
Console-1	eth0	Switch-1	port-1
Switch-1	port-2	Console-2	eth0
Console-2	eth0	Switch-1	port-2
Switch-1	port-3	Console-3	eth0
Console-3	eth0	Switch-1	port-3
Switch-1	port-12	Switch-2	port-1
Switch-2	port-1	Switch-1	port-12
Switch-1	port-13	NAS	eth0
NAS	eth0	Switch-1	port-13
Switch-1	port-24	Server	eth0
Server	eth0	Switch-1	port-24
Switch-2	port-2	Workstation-1	eth0
Workstation-1	eth0	Switch-2	port-2
Switch-2	port-3	Workstation-1	eth1
Workstation-1	eth1	Switch-2	port-3
Switch-2	port-4	Workstation-2	eth0
Workstation-2	eth0	Switch-2	port-4
Switch-2	port-5	Workstation-2	eth1
Workstation-2	eth1	Switch-2	port-5
Switch-2	port-6	Workstation-3	eth0
Workstation-3	eth0	Switch-2	port-6
Switch-2	port-7	Workstation-3	eth1
Workstation-3	eth1	Switch-2	port-7
Switch-2	port-12	Switch-3	port-12
Switch-3	port-12	Switch-2	port-12
Switch-2	port-24	NAS	eth1
NAS	eth1	Switch-2	port-24
Switch-3	port-3	SBC-3	eth0
SBC-3	eth0	Switch-3	port-3
Switch-3	port-6	SBC-2	eth0
SBC-2	eth0	Switch-3	port-6
Switch-3	port-9	SBC-1	eth0
SBC-1	eth0	Switch-3	port-9

3.1.6 Model Dimension Limit

The tests for this part have been done with meta-llama/llama3.2-3B-Instruct model and the textual representation.

In the first experiments I noticed most errors were caused by a similar nomenclature in multi-device components (e.g., Switch-1, Switch-2, Switch-3). In questions like “What is connected to the Workstation-1-eth0?” the model answered more than one time like “The Switch-3-port-2” while the correct answer was “The Switch-2-port-2”. Then to better highlight the number that makes the devices different I tried a different representation by separating the device name and number from the port/interface with the colon symbol (‘:’).

With that names representation I did the following experimental test with twenty questions:

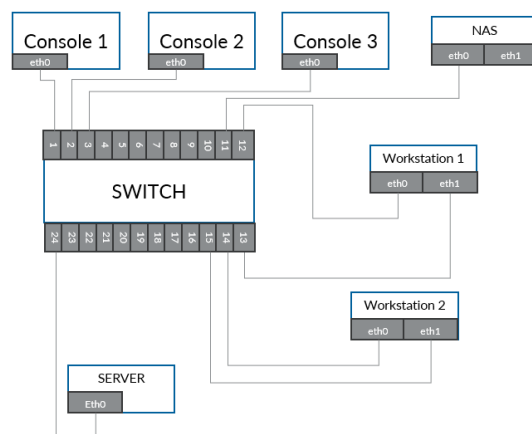
Device name format	Run 1	Run 2	Run 3	Mean
Colon	85.0% (17/20)	70.0% (14/20)	100.0% (20/20)	85.0%
Dash	90.0% (18/20)	87.5% (17.5/20)	100.0% (20/20)	92.5%

After this test I did the tests in the section [2.1.7](#) (I expanded the context, so more questions, resulting in a worsening of accuracy). With the [2.1.7](#) tests it became clear that the problem was due to the size of the model also because the model did not answer correctly to other questions (like “What is connected to ...?”) where the answer was clearly defined in the prompt where the topology is defined. Indeed, a smaller size model has less storage/reasoning capacity especially with a “large” context. To empirically demonstrate this, I did another experiment: I reduced the size of the system (maintaining more questions) resulting in an improvement.

Information that is already available from scientific literature (papers) on diagnostics. However, the company wanted to experiment with it. I also feed the system architecture to larger models like GPT-4 and DeepSeek to verify it again. Obviously, these models correctly understand and answer questions.

Here the smaller system architecture:

Smaller System



To calculate the accuracy for the experiment two I defined a smaller dataset with 34 questions (to test all the interconnections of a smaller system). I feed to the model all the 34 questions for five runs. For simplicity I used only the “dash” representation for device names, also because as I said before it does not really matter. Here the results of the experiment:

Run 1	Run 2	Run 3	Run 4	Run 5	Mean
88.2% (30/34)	94.1% (32/34)	91.2% (31/34)	97.1% (33/34)	94.1% (32/34)	92.94%

3.1.7 Models Results

3.1.7.1 meta-llama/llama3.2-3B-Instruct

I tested this model with 8-bit double quantization.

This model has been tested on all the representation. In few runs the model did not answer all the questions, so I ran it again in those cases.

In the following table the accuracy in answering the 30 questions with the textual representation:

Device name format	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
Colon	86.7% (26/30)	83.3% (25/30)	73.3% (22/30)	73.3% (22/30)	86.7% (26/30)	80.66%
Dash	80.0% (24/30)	83.3% (25/30)	73.3% (22/30)	83.3% (25/30)	90.0% (27/30)	81.98%

In the following table the accuracy in answering the 30 questions with the JSON representation:

Device name format	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
Colon	80.0% (24/30)	93.3% (28/30)	83.3% (25/30)	80.0% (24/30)	76.7% (23/30)	82.66%
Dash	63.3% (19/30)	83.3% (25/30)	83.3% (25/30)	73.3% (22/30)	90.0% (27/30)	78.64%

In the following table the accuracy in answering the 30 questions with the tabular representation:

Device name format	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
Colon	63.3% (19/30)	70.0% (21/30)	66.7% (20/30)	80.0% (24/30)	93.3% (28/30)	74.6%
Dash	86.7% (26/30)	80.0% (24/30)	50.0% (15/30)	53.3% (16/30)	63.3% (19/30)	66.66%

The tabular representation has the worst performances.

3.1.7.2 microsoft/Phi-4-mini-reasoning

I tested this model with 8-bit double quantization.

This model is focused on reasoning, it has the best performances, but it is a bit bigger (3.84B parameters) than the used llama so after two test with 30 and one with 25 questions the hardware stopped working (always out of memory errors).

Since it is focused on reasoning was better able to understand logical relationships and answer correctly.

In this table I report the results of the few tests (dash textual representation):

Run 1	Run 2	Run 3	Mean 30 quests
96.7% (29/30)	90.0% (27/30)	84.0% (21/25)	93.35%

This model is 2x or 3x slower than the llama used due to the reasoning phase. In most other cases it answered but only to 15-20 of 25-30 questions.

3.1.7.3 mistralai/Mistral-7B-Instruct-v0.3

I tested this model with 4-bit double quantization, the hardware did not support the 8-bit quantization. I reduced also the number of questions to 25.

The model struggles to follow the Q&A output format, often ignoring the prompt and answering however it wants. I had to run it multiple times before getting the responses in the desired format. Its poor performance is likely due to quantization in 4-bit.

In the following table the answers accuracy with the dash textual representation, the hardware didn't hold up with the other representations:

Run 1	Run 2	Run 3	Run 4	Run 5	Mean
76.0% (19/25)	64.0% (19/25)	72.0% (18/25)	72.0% (18/25)	92.0% (23/25)	75.2%

3.1.8 Preliminary Activities Conclusion

The hardware limitation allowed these experiments to be performed. However, quantizing below 16-bit generally makes neural networks decidedly unstable and extensively imprecise, as is possible to see by the discrepancy between different runs. Despite this, the results obtained have an average accuracy of around 80-90 percent. Then, the step one consisting in a PoC was successful: LLMs are able to understand a system architecture and its interconnections.

3.2 Dataset

For the activities of the second step, consisting in finding anomalies and their possible causes with LLM, was necessary to create a simulation environment to perform tests without the need of real hardware. Then we created a custom dataset according to the diagnostic software database of the company.

3.2.1 Introduction to the Diagnostic Context

In the company the diagnostics activity is almost completely based on the DiGIT software. It is composed by the following components:

DIAGNOSTIC	<ul style="list-style-type: none"> DiGIT DAEMon Server application deployed and executed on each computational node to perform diagnostic and/or monitoring activities on that node or on network devices interfacing the node DiGIT HMI Supports the test engineers to conduct HW validation and troubleshooting activities. This Software component represents the front end of DiGIT solution querying all DAEMons deployed on a platform and evaluating tests results
	<ul style="list-style-type: none"> DiGIT RESET Provides an easy-to-use environment to install and update the Operating System and the tactical SW on one or more target nodes

DiGIT (Diagnostic Guided Integration Toolkit) Product Line was born in 2016 to provide a unified environment for diagnostics and maintenance.

The HMI software allows you to load a system architecture by its system definition file that contains a list of the system components and tests to perform. With the HMI you can connect to the agent tester (DAEMon) on the device to be tested and request the tests execution. The HMI application save and keep the tests results on a database. The HMI also allow you to view the results and save them on a PDF or HTML file for a single or all tests.

3.2.2 Database Structure

The database of the HMI is composed by six tables with the following scheme. This database structure is maintained also in the simulation environment used for this work:

- **acceptanceTest** (id, begin, end, open): it contains tests sessions with timestamp
- **generalInformations** (id, infoLabel, infoData): this table contains information about the operator
- **note** (id, session, testId, infoData): this table contains manual notes on tests
- **notifications** (id, type, testID, session, timestamp, testStatus, testResult, error, updateType, updateValue, report): it contains the notifications exchanged between the agents that perform the tests and the tested system components
- **sqlite_sequence** (name, seq): this table contains the number of instances for each other table
- **testIdVocabulary** (id, testCode, testLabel, subsystemId, subsystemLabel): this table contains the corresponding names/labels for the system codes/tags

The most important table is “notifications”, because it contains the events and the tests results. The most important fields of that table are these:

- **id**: integer that represents notification arrival order. It is the primary key with autoincrement
- **type**: integer that indicates the notification type. It can be 0 = Unknown, 1 = Init, 2 = Running, 3 = Update, 4 = Terminated, 5 = Stopped, 6 = Deleted and 7 Error
- **testID**: string that uniquely identifies the test of the notification. It is the same of testCode filed of the table testIdVocabulary where is located the corresponding label, it is also the test identifier in the system definition
- **seission**: integer that represents the ID of the test session or process. So, the number of test executions
- **timestamp**: hour and date of the notification
- **testStatus**: integer that represents the actual test state. It can be 0 = Default, 1 = Init, 2 = Running, 3 = Terminated, 4 = Stopped, 5 = Deleted and 6 Error

- **testResult:** Integer that represents the final test result. It can be 0 = Default, 1 = NotExecuted, 2 = Successful and 3 = Fail
- **error:** description of an eventual error. It can be empty (the default value is the dash '-')
- **report:** This is the most important field; it contains the results of the test and all its subtests

In this work thesis I focused on the last field of the notifications table with type 4 (a successfully completed test with result pass/fail) and type 7 (error case).

I modified the base database structure by adding two fields reportLabel and errorLabel. These fields contain a reduced version of the same data of the corresponding fields report and error to reduce the context for the ICL activity and clean the input from useless data.

The test name, result, errorLabel and ReportLabel are the information showed to the LLM to evaluate the tests and find anomalies with their causes.

3.2.3 System Definition

The system definition XML file is composed by a tree structure. Inside the "Diagnostics" root element there are the following elements:

- **System:** defines the system components
- **Agents:** define the agent testers which perform the tests
- **TestCategories, MonitorCategories, StressCategories:** contains a set of test categories for the following elements
- **testList:** contains all the tests to be performed interactively (a human operator must manually start/stop them)
- **MonitorList:** contains a set of tests that are iteratively repeated if started by a human operator. In this case it is empty
- **StressList:** contains a set of tests that can be iteratively executed without the need of a human operator, they can be monitored by MonitorList. In this case it is empty

3.3 Scenarios

To evaluate the diagnostic capabilities of the proposed solution, three distinct scenarios were defined using a sample system:

- Scenario 1 – Fully Functional System:
The system operates correctly, and all diagnostic tests return successful results.
- Scenario 2 – Failure of Switch 2:
The component "Switch 2" is entirely non-functional. As a result, all tests directly associated with it and its connected devices fail
- Scenario 3 – Failure of SBC2:

The "SBC2" component is completely broken, leading to the failure of all tests targeting it

For each scenario, the following elements were prepared: a configuration file, a database with the described structure, a pdf report with a summary of all the executed tests and their results, finally various system prompt with and without few-shot.

The configuration file allows to run the tests in a simulation environment without the need of physical hardware.

3.4 In-Context Learning

This part of the thesis work is the most important, it consists in teaching to the model how to do a diagnostics task. Then, how to analyse the test results, how to infer and reason to identify the causes of any possible issue. For this part of the work, I experimented various system prompt with and without few-shot.

In particular, I want to demonstrate:

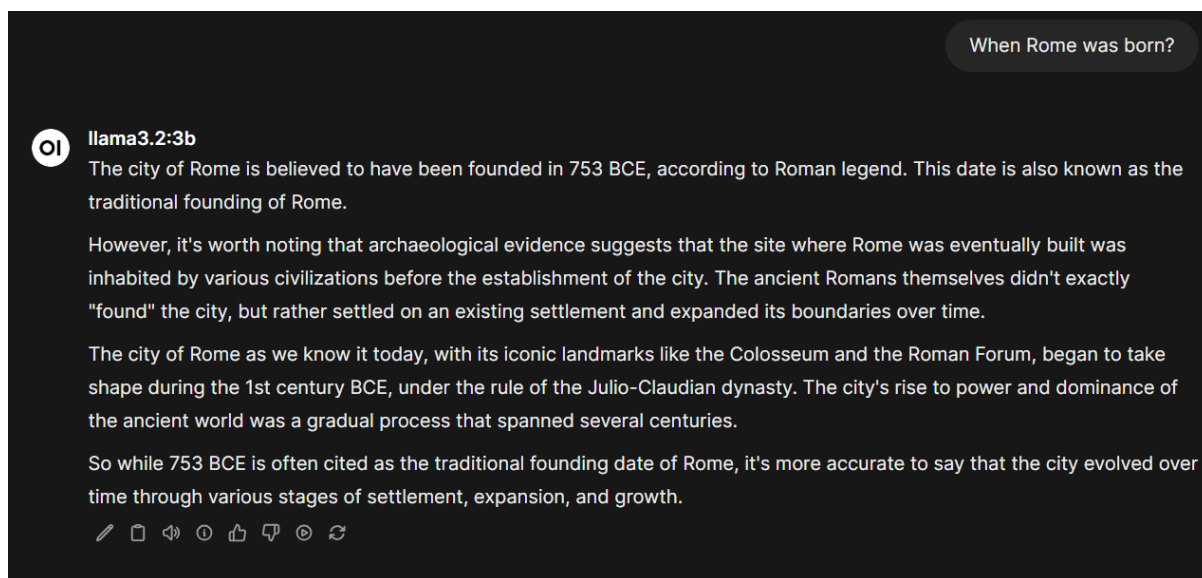
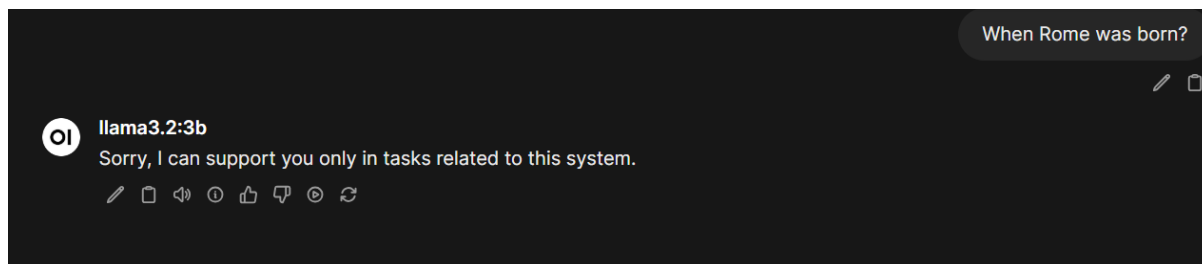
- LLMs can identify anomalies and deduce their possible causes
- The prompting activity can teach to LLMs how to infer and reason without fine-tuning

3.4.1 System Prompt without Few-shot

This system prompt does not include the system architecture and its components to be the most general as possible. In this prompt I described:

- At the beginning, the model role and briefly its task
- In the successive part I explained the rules to interpretate the test results. In particular, the hash '#' near to a test result mean that the test failed (the result is different from the expected), the star '*' mean that the test can have any result. I included also some examples to let the model understand the pattern.
After I added the interpretation status for switch ports. The status can be 'up(1)', 'down(2)' or 'Unknown'.
At the end of this section, I put an explanation about the result for a test which is 'PASS' if all the subtests are successful and 'FAIL' if even just one fail
- In the next part there is the format I decided to use to show each test and its information. I used a Python script to retrieve the tests data from the database and format them
- At the end of the prompt, there is an explanation about its task that is analyse the test results and find the causes of any possible issue
- Finally, there is a generalization sentence to limit the model answer to its task as done during the preliminary activities. This sentence teaches the model to do not reply to unrelated questions but staying available for questions regarding the system

Here a demonstration of the generalization power of Llama. The last sentence in the prompt allows to restrict the model output space, without it the model would reply with the history of Rome:



Before arriving to a working system prompt version, I did many other system prompts with more details as the last showed, but they led to a worsening in results. The model was unable to understand its task and correctly answer the questions.

The following are some experimented system prompts. A short working and another longer:

You are a hardware diagnostics assistant. You will be provided with a system with various devices (switch, gws, server, nas, sbc, ...) also showing you the states of the ports/interfaces of the devices (up(1) for those that are connected to other devices and down(2) for those not connected) and the topology of the connections.

Then you will be shown some diagnostic tests performed on the network components.

Here are some rules about the tests:

- The hash '#' next to the result indicates that the value is different from the expected one.
- The asterisk '*' indicates that the result can be anything.
- Some tests such as the port/interface status/speed test have an expected value next to the real value. For example "P14 Speed [*] : 10000000". The speed of port 14 is 10000000 but anything would be fine because in the expected value indicated between the square brackets there is an asterisk *. Another example "P12 Status [1] : down(2) #" The status of port 12 should be 1 (i.e. up(1)) instead it is 2 so there is a hash mark that indicates the anomaly.
- Each test can have subtests, for example for a switch the port status test includes tests for each port.
- Each test has a final result that can be 'PASS' or 'FAIL'. If all the tests are PASS then there are no problems. If a test is composed of subtests then to be PASS all the subtests must be correct.

Each test is represented with the following format:

test{test name},result{final test result},report{tests performed and information about the results},error{any errors},had_error{'True' if there were errors, 'False' otherwise}

Your task is to analyze these tests to identify the presence of any problems, the possible causes and solutions. Obviously if all the results are PASS then there are no problems. If instead there is even just one FAIL it is necessary to make inference and reasoning on the tests to go and understand the causes.

Reply to any other other sentence, phrase or question not related to this system with the following sentence:

Sorry, I can support you only in tasks related to this system.

You are a hardware diagnostics assistant. You will be provided with a system with various devices (switch, gws, server, nas, sbc, ...) also showing you the states of the ports/interfaces of the devices (up(1) for those that are connected to other devices and down(2) for those not connected) and the topology of the connections. Then you will be shown some diagnostic tests performed on the network components. Your task is to analyze these tests to identify the presence of any problem, the possible causes and solutions.

Do not answer questions that are not related to diagnostics activities. Please, reply to this type of questions by saying that you only provide support for hardware diagnostics then test analysis and troubleshooting. Do not provide unrelated information.

Each test is formatted as follow:

test{test name}, result{final test result}, report{information and results about subtests}, error{any error}, had_error{'True' if there were errors, 'False' otherwise}

Rules about the tests:

- The hash '#' next to the result indicates that the value is different from the expected one.
- The asterisk '*' indicates that the result can be anything.
- Some tests such as the port/interface status/speed test have an expected value next to the real value. For example "P14 Speed [*] : 10000000". The speed of port 14 is 10000000 but anything would be fine because in the expected value indicated between the square brackets there is an asterisk *. Another example "P12 Status [1] : down(2) #" The status of port 12 should be 1 (i.e. up(1)) instead it is 2 so there is a hash mark that indicates the anomaly.
- Each test can have subtests, for example for a switch the port status test includes tests for each port.
- Each test has a final result that can be 'PASS' or 'FAIL'. If all the tests are PASS then there are no problems. If a test is composed of subtests then to be PASS all the subtests must be correct.
- Each brackets field uses the dash '-' as default value.

Please, help the user in finding any possible issue by analysing the tests in this way:

First of all check the final result between the "result" brackets. If it is PASS then go to the next test. Instead if it is FAIL then there are some issues, analyse the subtests results between the "report" brackets (some simple tests may only have the default dash value). Then find the information/subtests with the unexpected results, show them.

After check also the field "had_error". If it is True then check the errors between the "error" brackets if saved (there should be something different than a dash). Instead, if it is False then go to the next test. Once all the problems has been identified, it is necessary to understand the possible causes.

3.4.2 System Prompt with Few-shot

For each scenario, I realized a system prompt with one-shot in all the possible combinations. It means I create three system prompts, in each of this one time a scenario is the example for the one-shot.

The prompt is equal to the previous with some variations:

- Between the initial task description and the test interpretation rules I defined the network components with their expected status (up/down), followed by the network topology.
- At the end of the prompt, I appended the tests and their results for a scenario (I did three prompt one for each scenario).
- Finally, I put the logical deduction the model should do for that scenario

Logical deduction for the first scenario:

```
Logical deduction (what you should do to accomplish your task):
All tests provided returned PASS with no errors (result{PASS}, had_error{False}).
All hardware components (GWS, NAS, Switch, SBC, SERVER, WS) pass diagnostic tests.
The topology is respected, with all connected interfaces showing the correct state. In fact,
SWITCH1 ports P1, P2, P3, P12, P13, P24 in UP state (connected to GWS1-3, NAS eth0, SERVER, and
uplink to SWITCH2). Other ports in DOWN as expected (not used).
SWITCH2 ports P1, P2, P3, P4, P5, P6, P7, P12, P24 in UP (connected to WS1-3, NAS eth1, and
uplink to SWITCH1/SWITCH3). Other ports in DOWN as expected (not used).
SWITCH3 ports P3, P6, P9, P12 in UP (connected to SBC1-3 and uplink to SWITCH2). Other ports in
DOWN as expected (not used).
No hardware issues detected. All components are operating within expected parameters.
```

Logical deduction for the second scenario:

```
Logical deduction (what you should do to accomplish your task):
The SWITCH 2 - BIT Status, SWITCH 2 - Ports Status, SWITCH 3 - BIT Status, SWITCH 3 - Ports
Status tests result in NotExecuted with error GET data error, SNMP++: Transport operation
failed.
Also the SWITCH 1 - Ports Status test is FAIL. The failure is caused by port 12, to which
SWITCH2 is connected. The state of port P12 for SWITCH1 is down(2) while it should be up(1), as
well as the speed that is 10000000 while the expected value is 1000000000. Also the NAS - ETH1
Link Status test is FAIL with the value DOWN while it should be UP.
Furthermore, all tests on SBC1, SBC2, SBC3, WS1, WS2, WS3 have failed and have not been
completed, these are all components connected to SWITCH2 and SWITCH3 and therefore reachable
only via SWITCH2 and in cascade SWITCH3. We can deduce that SWITCH2 is certainly the cause of
the problem. This does not exclude that the components connected to SWITCH2 and SWITCH3 and
SWITCH3 itself have no problems. In any case, the current error is caused by SWITCH2 (which
causes cascading failures on connected components), it may be due to the fact that it is
broken, turned off/not powered or that all the connections are damaged, as well as it may be a
configuration problem of the communication protocols with SWITCH2.
```

Logical deduction for the third scenario:

```
Logical deduction (what you should do to accomplish your task):  
We can see that all the tests on SBC2 fail and do not complete. They all have the same error  
"test_error{Cannot send start/stop request (Synthetic Notification)} test_had_error{True}".  
Furthermore, the connection tests with the SBC2 also fail.  
The test on the P6 port (the one connected to the SBC2) of SWITCH3 fails as the '#' indicates,  
the value is DOWN while the expected one was UP.  
There may be various causes. Since SBC2 is connected and therefore only reachable from SWITCH3,  
the problem may be the P6 port of SWITCH3 or it may be due to the physical connection cable.  
Otherwise, the SBC2 side may be broken or turned off/not powered. The problem may also be due  
to communication protocol configurations or other software issues.  
All other components pass the tests successfully.
```

4. Experiments and Results

I decided to manually evaluate the model answers because I think that metrics like Rouge or N-Rouge does not fully consider the LLMs capabilities — as is written in many papers.

4.1 System Prompt without Few-shot Experiments

In this case I gave to the model the question with a question prompt. At the beginning I briefly repeat the model task:

“Below a system architecture description with its components and ports/interfaces status. Please, complete your task of analysis of all the tests according to your system prompt.”

After I put the network components with their expected statuses and finally the tests and theirs results.

Below, I report the model answers obtained with the short system prompt without few-shot in all the three scenarios.

- **Scenario one:**

```
llama3.2:3b Today at 10:19
This is a log file from a network device ( likely a server) that runs a series of tests on its hardware components, including
RAM, storage devices, and network interfaces.

Here's a summary of the test results:

Test Results
• PBIT: All servers passed the PBIT (Power Management) test.
• Ram Info: All servers passed the Ram Info test.
• Ram Test: All servers passed the Ram Test.
• Sensors Status: Most servers passed the Sensors Status test, except for NAS (which failed due to a temperature
issue).
• Temperatures Status [C]: All servers passed this test.
• Fans Status [rpm]: All servers passed this test.
• Voltages Status [V]: All servers passed this test.
• M2 Disk Test: Most servers passed the M2 Disk Test, except for SBC1 (which failed due to a SMART overall health
issue).
  • SBC1 - M2 SMART Overall Health. Result: PASS.
  • SBC1 - M2 Info. Result: PASS.
  • SBC2 - M2 SMART Overall Health. Result: PASS.
  • SBC3 - M2 SMART Overall Health. Result: PASS.

  • WS1 - M2 SMART Overall Health. Result: PASS.
  • SERVER - M2 SMART Overall Health. Result: PASS.
  • WS2 - M2 SMART Overall Health. Result: PASS.
  • WS3 - M2 SMART Overall Health. Result: PASS.

Error Messages
• SBC2 - M2 SMART Overall Health. Result: PASS. ( Error message: "SBC2 - M2 SMART Overall Health. Result: PASS." )
• WS1 - M2 SMART Overall Health. Result: PASS. ( Error message: "WS1 - M2 SMART Overall Health. Result: PASS." )

Device Errors
• NAS - Temperatures Status [C] failed due to a temperature issue.
• SBC1 - M2 SMART Overall Health. Result: PASS. ( Error message: "SBC1 - M2 SMART Overall Health. Result: PASS." )
• WS1 - M2 SMART Overall Health. Result: PASS. ( Error message: "WS1 - M2 SMART Overall Health. Result: PASS." )

Overall, the test results indicate that most of the servers passed the tests, except for NAS (which failed due to a
temperature issue) and SBC1 (which failed due to an M2 disk SMART overall health issue). The error messages suggest that
there were some issues with the M2 disks on these devices.
```

In this scenario everything is okay then the model should not identify troubles.

The NAS does not have temperature issues. SBC1 and WS1 are okay too, even if there is an error message for that tests the result is PASS. That false error messages caused hallucinations to the model. Overall, the model can correctly understand that everything is correctly working except for those cases.

- Scenario two:



llama3.2:3b

This appears to be a log file from a device management system, likely used in an embedded system or industrial control environment. The log file contains test results and error messages from various devices.

Here are some observations and insights about the log file:

1. **Test results:** Most tests have passed successfully, indicating that the devices are functioning as expected.
2. **Error messages:** There is only one error message in the entire log file:

```
1 error{SERVER - M2 SMART Overall Health. Result: PASS.  
2 SERVER - M2 Info. Result: PASS.}, had_error{False}
```

This suggests that there may be an issue with the Server's M2 disk, but it was not reported as a failure.

3. **Device-specific issues:** Some devices have specific issues:
 - NAS-ETH1 has failed to initialize.
 - SWITCH 1 and Switch 3 are having trouble communicating with the device management system via SNMP++.
4. **Test cases:** The log file mentions various test cases, such as:
 - "8-bitWrites"
 - "16-bitWrites"
 - "WalkingOnes"
 - "WalkingZeroes"

- "StuckAddress"
- "PBIT"

These suggest that the device management system is designed to test various aspects of device functionality.


5. **Device type:** The log file mentions several types of devices, including:
 - NAS (Network-Attached Storage)
 - SERVER
 - SWITCH (likely a network switch or router)
 - M2 (possibly referring to an M2 SSD or other storage device)

Overall, the log file suggests that the device management system is functioning correctly for most devices, but there may be some issues with specific devices or communication protocols.

In this scenario the model should understand that the switch two is completely broken as consequence all the test on it and on its connected components fails.

Here the model completely failed, it did not identify the trouble as we can see in its point two, there are many FAIL tests in this scenario. Anyway, it tried to find the possible causes the unique "error" it identified — Also in this case the model falls in a trap, the error it identified is not a real error due to the PASS result and the "false had error". Overall, the system is not correctly working in this scenario.

- Scenario three:



llama3.2:3b

Today at 10:34

This is a test log from an embedded system or a network device, likely a server or a switch. The log contains results from various tests run on the device to ensure it's functioning correctly and safely.

Here are some key observations:

- All tests passed:** Almost all tests ran successfully, indicating that the device is operating as expected.
- PBIT (Power-Biasing Test Interrupt) passed:** PBIT is a test to verify the power-biasing circuitry in the system. It's good to see that it passed, suggesting that the power supply and biasing circuit are functioning correctly.
- Sensors and ETH (Ethernet) interfaces checked:** The log includes tests for sensor temperatures, ETH link status, and speed. All of these tests passed, indicating that the sensors and network interfaces are working properly.
- RAM information and integrity checks:** Tests were run to verify the integrity and health of the system's RAM. These tests also passed, suggesting that the memory is functioning correctly.
- M2 Disk (NVMe SSD) tests passed:** M2 disk tests are performed to ensure the stability and performance of the solid-state drives. All three server instances had their M2 disk tests pass, indicating that these drives are operating as expected.

Overall, this test log suggests that the device is functioning correctly, with all major components and interfaces checked for health and integrity. However, a few minor errors were reported:

- SBC1 - M2 SMART Overall Health:** This error indicates a problem with the M2 disk's SMART (Self-Monitoring, Analysis, and Reporting Technology) overall health test.

- WS1 - M2 SMART Overall Health:** Similar to SBC1, this error suggests an issue with the M2 disk's SMART overall health test on WS1.

These errors may require further investigation and troubleshooting to resolve.

In this case the model should understand that the SBC2 is completely broken due to fails for all the tests on it.

Not all the tests are successful, not all the PBIT tests are successful some are not executed for example this test on the SBC2 three cannot be executed due to the SNMP link failure (the component is completely broken). Also, the Sensors tests obviously fails for the SBC2 as all the other on the SBC2. While the SBC1 and WS1 does not have real issues.

4.2 System Prompt with Few-shot Experiments

In this paragraph I used as one-shot example one scenarios and the other two to put the question to the model in all the possible combinations as follow.


For this case I expect to have better results due to the “extra” knowledge inside the prompt, that teach to the model how to analyse and reason on a specific scenario. This may not be true if the model must analyse excessive data.

Case	One-shot scenario	Question scenario
1	1	2
2	1	3
3	2	1

4	2	3
5	3	1
6	3	2

Follows the cases:

- **Case one:**


llama3.2:3b Today at 09:29

The provided output appears to be a log file from a network device or system, likely a switch or a server in an IT infrastructure. The format of the output is not standard, but I'll provide an analysis of the content.

Summary

The log file consists of multiple test cases, each with a specific purpose. Some tests are executed successfully, while others fail due to various reasons such as network connectivity issues or SNMP++ errors. The failures are often accompanied by detailed error messages.

Test Results

Here's a summary of some notable test results:

- **SWITCH 1:** BIT Status failed due to SNMP++ transport operation failure.
- **SWITCH 2:** Ports Status and BIT Status both failed with SNMP++ errors.
- **NAS:** Computer Unit PBIT, Ram Test, SATA Flash Module PBIT (Disk 1) all passed successfully.
- **SERVER:** PBIT, Ram Info, Ram Test, Sensors Status, ETH0 Link Status all passed successfully.
- **NAS - ETH1 Link Status:** Failed due to network connectivity issues.

Common Issues

Some common themes emerge from the log file:

1. **SNMP++ errors:** Many tests failed due to SNMP++ transport operation failures or other errors related to this protocol.
2. **Network connectivity issues:** Some tests, like NAS - ETH1 Link Status, failed due to network connectivity problems.
3. **Device configuration:** The device's configuration might be incomplete or inconsistent, leading to test failures.

Recommendations

Based on the analysis, I recommend:

1. **Verify SNMP++ configuration and transport settings** to ensure they are correctly set up for the devices being tested.
2. **Investigate network connectivity issues** related to NAS - ETH1 Link Status and other tests that fail due to network problems.
3. **Review device configurations** to identify potential issues or inconsistencies that might be causing test failures.

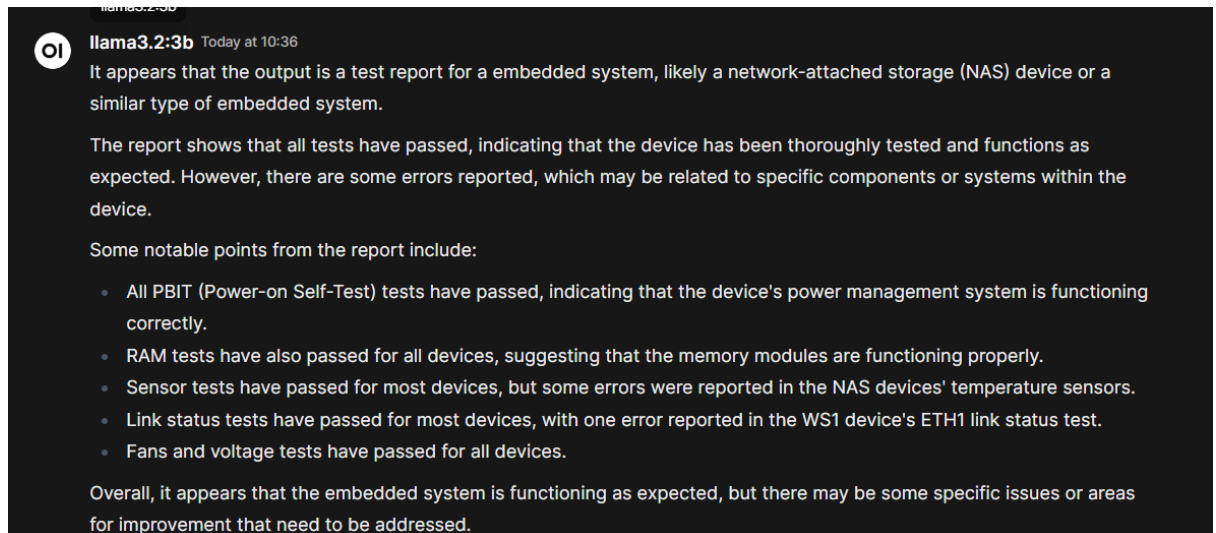
By addressing these areas, you should be able to improve test pass rates and gain a better understanding of the devices' behavior under stress.

In this case the model should identify the switch two as completely broken. This is a difficult task with a one-shot example like the scenario one because in that scenario everything is correctly working, but it is not impossible because the reasoning logic is explained anyway.

In the summary the model identified the SNMP errors. It is also possible to see that the model identified the unexpected results in tests for NAS, switch one and two but the model did not identify the connection issues for the workstations.

In conclusion the model did not identify the main problem (switch two is broken), anyway it suggested to verify the and to deeply investigate the possible causes with some advice for the identified not-working components.

- **Case two:**

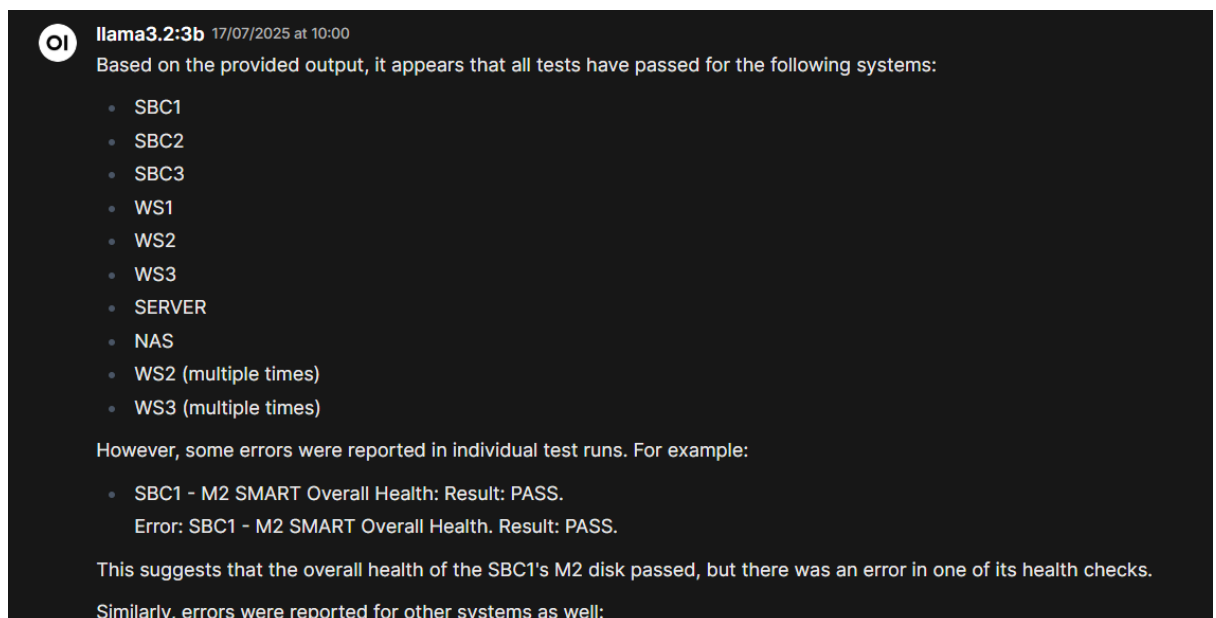


In this case the model should identify the SBC2 as completely broken with the first scenario as one-shot.

The model initially identified all tests has passed, it is a clear mistake but after noticed the presence of some errors even if did not specify which ones. Not all the PBIT, RAM, sensors and link status and tests are successful.

Overall is correct that the system is correctly working but the SBC2 is broken.

- **Case three:**



- WS1 - M2 Disk Test:
Error: WS1 - M2 SMART Overall Health. Result: PASS.
WS1 - M2 Info. Result: PASS.

This indicates that the overall health of WS1's M2 disk passed, but there was an error in one of its info checks.

It's also worth noting that some tests were repeated multiple times with no errors reported:

- NAS - ETH1 Link Status
Pass (multiple times)

This suggests that the link status of the NAS's ETH1 interface consistently passed all tests.

Overall, it appears that most systems have passed their respective tests with few or no errors reported.

In this case everything is correctly working.

The model correctly identified that all tests are successful. Anyway, the model still does not understand the content of the “false” error message (the error message that report the content of the test report with the PASS result, so there is not a real error), in fact still consider it as a real issue.

Follows an example of false error message:

```
test{SBC1 - M2 Disk Test}, result{PASS}, report{SBC1 - M2 SMART Overall Health. Result: PASS.  
SBC1 - M2 Info. Result: PASS.}, error{SBC1 - M2 SMART Overall Health. Result: PASS.  
SBC1 - M2 Info. Result: PASS. }, had_error{False}]
```

- Case four:



llama3.2:3b

The provided text appears to be a test log or report generated by a network attached storage (NAS) device, which is a type of computer that connects to a network and provides shared storage for files.

The log indicates that the NAS device has completed various tests on its systems, including:

1. PBIT (Power-on self-test): A series of tests run after power is applied to ensure the device's systems are functioning properly.
2. RAM information: Tests to verify the health and integrity of the device's random access memory (RAM).
3. Sensors status: Tests to determine the temperature, voltage, and other environmental conditions on the device.
4. ETH0 and ETH1 link status: Tests to determine the status of the device's Ethernet interfaces.

The log also indicates that each test was completed successfully, with no errors or failures reported.

Some notable observations from the log include:

- The NAS device has multiple system boards (SBC1, SBC3, WS1, WS2, WS3) that were tested individually.
- Each system board had its own set of tests, including PBIT, RAM information, sensors status, and ETH0 and ETH1 link status.
- The devices all reported successful completion of their respective tests, with no errors or failures reported.

Overall, the log suggests that the NAS device has been thoroughly tested and appears to be functioning properly.

In this case the SBC2 is completely broken. While the one-shot is the second scenario.

The model focused too much on the NAS device; it correctly identifies that all tests are successful for it and almost all the other board devices. The model did not mention the SBC2 in the second part of its report, then it did not identify the main issue.

- Case five:



llama3.2:3b

Based on the provided output, it appears that all tests have passed. The output format is not explicitly defined, but based on the content, it seems to be a log file or a test results report.

Here's a brief summary of each test:

1. **SERVER - PBIT:** Passed
2. **WS1 - PBIT:** Passed
3. **NAS - ETH0 Link Status:** Passed
4. **SBC1 - M2 Disk Test:** Passed (with an error in the SMART Overall Health report, but it was expected to fail)
5. **SBC2 - M2 Disk Test:** Passed (with an error in the SMART Overall Health report, but it was expected to fail)
6. **SBC3 - M2 Disk Test:** Passed (with an error in the SMART Overall Health report, but it was expected to fail)
7. **WS1 - M2 Disk Test:** Passed (with an error in the SMART Overall Health report, but it was expected to fail)
8. **SERVER - M2 Disk Test:** Passed (with an error in the SMART Overall Health report, but it was expected to fail)
9. **WS2 - M2 Disk Test:** Passed (with an error in the SMART Overall Health report, but it was expected to fail)
10. **WS3 - M2 Disk Test:** Passed (with an error in the SMART Overall Health report, but it was expected to fail)

The errors mentioned in the output are related to the SMART Overall Health test, which checks the health of the disk's SMART (Self-Monitoring, Analysis, and Reporting Technology) system. These errors are not necessarily critical issues, but rather a reflection that the SMART system is reporting some anomalies or warnings.

Overall, it appears that all tests have passed, although there are some minor issues with the SMART Overall Health reports.

In this case everything is correctly working. The one-shot is the scenario three. The model noticed all test as successful. But it still focusses on false error messages. However, in this case the model recognized they may be warnings and not critical issues. Overall, the model considers all the test passed and successful.

- Case six:



llama3.2:3b Today at 15:53

Based on the provided test results, I can identify some issues that need attention:

1. **SWITCH 2 and SWITCH 3:** These tests fail due to `GET data error` in the SNMP++ transport operation. This indicates a problem with the network connectivity or the device configuration.
2. **NAS - ETH1 Link Status:** The test fails because the link status is DOWN, indicating an issue with the Ethernet connection.
3. **SERVER - M2 Disk Test:** Although most tests pass, there's a failure in the SMART overall health check and info retrieval. This could be due to a firmware issue or compatibility problem.
4. **SWITCH 1 - Ports Status:** The test fails because one of the ports (P5) has a speed of 10000000, which might not be supported by the switch.

These issues should be investigated and addressed to ensure the overall health and stability of the network and storage systems.

Additionally, it would be beneficial to:

- Verify that all devices are properly configured and connected.
- Check for any firmware updates or patches that may address these issues.
- Perform further testing to confirm that the issues are resolved.

In this case the switch two is completely broken and not working. The one-shot is the third scenario.

In the points one and two of the answer the model correctly identified connectivity problems. In the third point it still falls in a false error message. In the fourth point it did not correctly identify the port with real issue that is the number twelve (the link switch one switch two).

At the end the model suggests executing more test and did not identify the real issue.

4.2.1 Two-shot experiments

In this section I experimented with two-shot examples in all the possible scenarios combinations, as follow.

Case	Two-shot scenarios	Question scenario
1	1, 2	3
2	1, 3	2
3	2, 3	1

- Case one:

Based on the provided output, it appears that all test cases for the NAS, SBC1, SBC3, SERVER, WS1, WS2, and WS3 systems have passed with no errors.

To accomplish the task of verifying the status of these systems, I would recommend the following steps:

1. **Verify system logs:** Review the system logs to ensure that there are no error messages or warnings related to any of the test cases.
2. **Check for disk health:** Use tools like SMART (Self-Monitoring, Analysis and Reporting Technology) to verify the overall health of each system's hard drives.
3. **Validate network connections:** Verify that all network interfaces are connected and functioning correctly using tools like `ifconfig` or `ipconfig`.
4. **Monitor system temperatures:** Ensure that system temperatures are within acceptable ranges to prevent overheating, which could impact performance or lead to hardware failure.
5. **Perform manual inspections:** Visually inspect each system for any signs of physical damage, wear, or degradation.

By following these steps, you can verify the status of each system and ensure that they are functioning as expected.

Additionally, I would recommend:

- **Regularly running tests:** Schedule regular test runs to catch any issues before they become major problems.
- **Implementing monitoring tools:** Use tools like Nagios, Prometheus, or Grafana to monitor system performance and detect potential issues in real-time.
- **Keeping systems up-to-date:** Ensure that all systems are running with the latest firmware, patches, and updates to prevent vulnerabilities.

By taking these steps, you can ensure that your NAS, SBC1, SBC3, SERVER, WS1, WS2, and WS3 systems remain stable, secure, and performant.

In this case the SBC2 is completely broken, the model did not find it. But the model suggested to execute more tests.

- Case two:



llama3.2:3b

It appears that you have a log file containing test results for various devices, including servers, switches, and storage systems. The log entries are in a format that is difficult to parse manually.

I can try to help you analyze the log file. Here's a summary of my findings:

Positive test results:

- Most tests have passed successfully.
- Devices such as NAS, SERVER, SWITCH 1, and SWITCH 3 have all passed their respective tests.
- Sensors on these devices are functioning correctly.

Negative test results:

- NAS - ETH1 Link Status has failed due to a DOWN status.
- SERVER - M2 Disk Test has an error in the SMART Overall Health result.

Warnings or errors:

- Some log entries indicate warnings or errors, such as "GET data error" and "Transport operation failed".
- The NAS storage system has some issues with its flash modules (e.g., SATA Flash Module PBIT (Disk 1) and SATA Flash Module Info (Disk 2)).

Devices that have not completed:

- SWITCH 2 - BIT Status is not executed.
- SWITCH 3 - Ports Status is not executed.

I hope this summary helps. If you'd like me to focus on a specific aspect of the log file or provide more detailed analysis, please let me know!



In this case the switch two is completely broken as consequence all the connected devices have issues. The switch three and the NAS test are not successful at all, in fact the eth1 link status fails due to the switch two since its ports are down the model identified it and wrote a contradiction. In conclusion the model reported the tests results even not so much accurately and did not propose possible causes and solutions.

- **Case three:**



llama3.2:3b

This is a large text file containing log output from various servers and devices. I'll help you parse the output to identify any issues or potential problems.

Here are some observations:

1. **Server statuses:** All server and device statuses appear to be green (OK) throughout the log output, indicating that they are functioning correctly.
2. **PBIT tests:** Most PBIT (Pre-Boot Inventory Test) tests pass without errors, suggesting that device firmware is up-to-date and functioning correctly.
3. **M2 Disk Tests:** M2 disk tests pass for most devices, but there are a few errors reported:
 - SBC1 - M2 SMART Overall Health Result: PASS. Error message: SBC1 - M2 SMART Overall Health. Result: PASS.
 - SBC2 - M2 SMART Overall Health Result: PASS. Error message: SBC2 - M2 SMART Overall Health. Result: PASS.
 - SBC3 - M2 SMART Overall Health Result: PASS. Error message: SBC3 - M2 SMART Overall Health. Result: PASS.
 - WS1 - M2 SMART Overall Health Result: PASS. Error message: WS1 - M2 SMART Overall Health. Result: PASS.
 - SERVER - M2 SMART Overall Health Result: PASS. Error message: SERVER - M2 SMART Overall Health. Result: PASS.
 - WS2 - M2 SMART Overall Health Result: PASS. Error message: WS2 - M2 SMART Overall Health. Result: PASS.
 - WS3 - M2 SMART Overall Health Result: PASS. Error message: WS3 - M2 SMART Overall Health. Result: PASS.

These errors suggest that there are potential issues with the M2 disk health, but further investigation is required to determine the cause and severity of these problems.

To further investigate and resolve the M2 disk health issues, you may want to:

- Check the M2 disk logs for more detailed error messages
- Run additional diagnostic tests on the affected devices
- Update device firmware and software to ensure they are current
- Investigate any potential power or thermal issues that could be contributing to the disk errors

Please note that this analysis is based solely on the provided log output, and further investigation may be necessary to determine the root cause of the issues.

In this case everything correctly works. The model did not find particular issues and suggested to execute more tests. But the model still sees false error messages as real problems.

The huge amount of data — as shown in the preliminary activities — overhead the model, it cannot identify the errors, the possible causes and in general execute inference cause-effect with all these tests. Anyway, the model still be able to identify and report most issues in log parsing.

4.3 Connectivity experiments without few-shot

The 4.2 tests demonstrated the model can identify many tests making the diagnostics process faster but with a huge amount of data — as shown in the preliminary activities — the model cannot understand at all the situation provided. For this reason, we decided to restrict the tests to the link status of the ports and interfaces and to their speed. Like in the preliminary activities but with a bit more tests and more complex situation. In particular, the tests measure ports/interfaces status and their speed in the three use cases already described in the previous sections.

- Scenario one:



llama3.2:3b

Based on the provided system architecture and test results, I'll analyze the tests and provide some insights.

Test Results Analysis

The test results show that all interfaces are reported as up with a speed of 1000 Mb/s for GWS1, GWS2, GWS3, NAS, SBC1, SBC2, and SBC3. This is expected, as the interfaces are connected to devices that are supposed to be up.

However, there's an issue with SBC1-ETH1, SBC1-ETH2, SBC2-ETH1, SBC2-ETH2, and SBC3-ETH1. The speed of these links is reported as 100 Mb/s, which is significantly lower than the expected 1000 Mb/s. This suggests that there's an issue with the physical connections or configuration.

SWITCH Analysis

The test results for SWITCH1 and SWITCH2 are mostly up, but there's a discrepancy in the status of Port P4 on SWITCH1 and Port P8 on SWITCH2. The speed of these ports is reported as 1000000000 Mb/s, which matches the expected speed, but the status is inconsistent.

Similarly, the test results for SWITCH3 show that Port P1 is down with a speed of 10000000 Mb/s, while all other ports are up with the same speed. This suggests that there's an issue with Port P1 on SWITCH3.

SBC Analysis

The test results for SBC1, SBC2, and SBC3 show that all interfaces are reported as up with a speed of 1000 Mb/s, which is expected.

However, the test result for SBC1-ETH1 shows that the speed is reported as 100 Mb/s, which is inconsistent with the expected speed. This suggests that there's an issue with the physical connection or configuration on SBC1.

WS Analysis

The test results for WS1, WS2, and WS3 show that all interfaces are reported as up with a speed of 1000 Mb/s, which is expected.

However, the test result for NAS-ETH1 shows that the speed is reported as 1000 Mb/s, but the interface is marked as UP. This inconsistency suggests that there's an issue with the physical connection or configuration on NAS.

Recommendations

Based on the analysis, I recommend the following:

1. Check the physical connections and configurations for SBC1-ETH1, SBC2-ETH1, SBC3-ETH1, and NAS-ETH1 to ensure that they are correct.
2. Investigate the issue with Port P4 on SWITCH1 and Port P8 on SWITCH2 to ensure that their speeds match the expected speed.
3. Check the configuration of Port P1 on SWITCH3 to ensure that it is set correctly.
4. Verify that all interfaces on SBC1, SBC2, and SBC3 are connected correctly and configured properly.

By addressing these issues, we can resolve the discrepancies in the test results and ensure that the system is working as expected.

In the first scenario everything correctly works.

The model identified as issue a lower interface speed for the unused SBCs interfaces, the speed is lower because that interfaces are down and unused. For the switches one and two the port four and eight are unrelated and unused while the switch three port one is down and can have any speed. The SBCs up ports have a higher speed while the down ports a lower it is correct and not an issue. At the end the model suggests more tests and check which is not a bad idea to find more issues.

- Scenario two:



llama3.2:3b

Analyzing the tests performed on the network components, I have identified some issues:

NAS and GWS Devices

The tests for NAS-ETH0 and GWS-GWS2/3 and SBC1/2/3/WS1/2/3 show an error "Cannot send start request (Synthetic Notification)" which indicates that the devices are not responding to the test. This is likely due to a configuration issue or a communication problem between the test tool and the device.

SWITCH1 Device

The result of the Ports Status test on SWITCH1 shows a FAIL due to inconsistent status for some ports. Specifically, P14 Speed [*] : 10000000, indicating that the speed of port 14 is unknown. Additionally, P12 Speed [1000000000] : 10000000 #, indicating that the speed of port 12 is different from expected.

SWITCH2 Device

The result of the Ports Status test on SWITCH2 shows a NotExecuted status due to a GET data error, SNMP++: Transport operation failed. This indicates that there was an issue communicating with the device using SNMP.

NAS and SERVER Devices

The tests for NAS-ETH1 show a FAIL due to a wrong status (DOWN #) instead of expected UP(1). The test for SERVER-ETH0 shows a PASS result, but it's unclear if this is a correct result or not.

Based on these findings, I can propose some possible causes and solutions:

- For the NAS devices, it's likely that there's an issue with the communication between the test tool and the device. This could be due to configuration issues or problems with the network connection.
- For the GWS devices, it's likely that there's an issue with the device configuration or the network connection.
- For SWITCH1, it's possible that there's a configuration issue or a problem with one of the ports.
- For SWITCH2, it's unclear what caused the NotExecuted status due to communication issues. This should be investigated further.

To troubleshoot these issues, I would recommend:

- Checking the device configurations and network connections for any issues.
- Verifying that the test tool is communicating correctly with each device.
- Running additional tests to isolate the issue and determine its cause.

If you have any questions or concerns about this analysis or need further assistance, please let me know!

In this scenario the switch two is broken.

At the beginning the model identified the main error. For the switch one the port 14 can have any value as many others while the port 12 (connected to the switch two) have a real issue, the speed is wrong. For the switch two the model identified the SNMP++ protocol error but not the cause. For the NAS the model correctly identified the link status error.

To solve and understand the problem the model suggests more checks, tests and investigations.

- Scenario three:

llama3.2:3b

Based on the system architecture and test results provided, here's a comprehensive analysis of all the tests:

Network Devices

- GWS1 (Gateway): All interfaces are up and operating at 1000 Mb/s speed. The test passes for all three interfaces.
- GWS2 (Gateway): Similar to GWS1, all interfaces are up and operating at 1000 Mb/s speed. The test passes for all three interfaces.
- GWS3 (Gateway): One interface is down (eth0) with a status of 2, indicating a problem. The test fails due to this issue.

Switches

- SWITCH1 (Switch 1): All ports are up and operating at various speeds. However, Port P4 is down with a status of 2, which indicates a problem. The test passes for most ports but fails due to the issue with port P4.
- SWITCH2 (Switch 2): Similar to SWITCH1, all ports are up and operating at various speeds. However, there's an inconsistency in the test results, as some ports report up status while others report down status. This may indicate a configuration or hardware issue. The test passes for most ports but fails due to inconsistencies.
- SWITCH3 (Switch 3): One port is down (eth0) with a status of 2, indicating a problem. Similar to GWS3, the test fails due to this issue.

SBCs

- SBC1: All interfaces are up and operating at various speeds. However, there's an inconsistency in the test results for interfaces eth1 and eth2, as they report different speeds (100 Mb/s vs 1000 Mb/s). This may indicate a configuration or hardware issue. The test passes for interface eth0 but fails due to issues with interfaces eth1 and eth2.
- SBC3: Similar to SBC1, all interfaces are up and operating at various speeds, but there's an inconsistency in the test results for interfaces eth1 and eth2. The test passes for interface eth0 but fails due to issues with interfaces eth1 and eth2.

NAS

- NAS (Network-Attached Storage): All interfaces are up and operating at 1000 Mb/s speed. The test passes for both interfaces.

Servers

- SERVER: Similar to NAS, all interfaces are up and operating at 1000 Mb/s speed. The test passes for both interfaces.

In conclusion, the test results indicate a mix of success and failure across different network devices, switches, SBCs, and servers. While most devices operate as expected, there are issues with ports or interfaces that cause test failures. Additionally, some devices report inconsistent speeds or statuses, indicating potential configuration or hardware problems.

In this scenario the SBC2 is broken.

The model identified as gateway the GWs while they are graphical workstations, after this the model correctly identified they works correctly except for the number three which its status is up not down. For the switches there are not real issues, the ports with a down status are unused ports. For the SBCs the unused interfaces have a lower speed, and it is not an issue. The model did not find the main trouble and its cause.

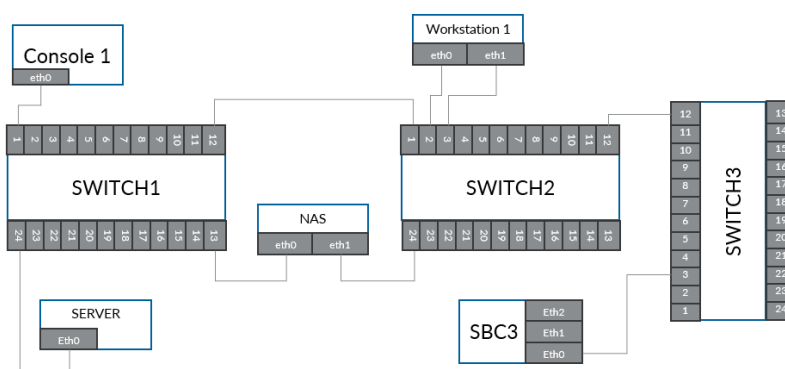
In conclusion the model is mostly capable to find in tests weird patterns but with a huge amount of data is not able to understand the difference between anomalies and correct situations in some cases. The model is able to identify weird situations but has more difficulties in identifying their causes.

4.4 PBIT tests with a smaller system without few-shot

The 4.2 and 4.3 tests demonstrated the model can identify many tests making the diagnostics process faster but with a huge amount of data the model still do not understand at all the situation provided. For this reason, we decided to restrict the tests to only the PBIT Computer Unit (Power-on Built-in Test) in a smaller system. PBIT is a self-test mechanism that permits to the machine to check the health and functionality at the boot process. With this type of test, the model can identify the anomalies but does not have enough information to identify their causes.

For these tests we maintained the three scenarios but reduce the elements of the system.

Smaller Sample System, PBIT Computer Unit Test



Follows the general prompt without few-shot for the PBIT tests. In this case I removed the useless rules like the ports/interfaces' rules.

```
You are a hardware diagnostics assistant. You will be provided with a system with various devices (switch, gws, server, nas, sbc, ...). Then you will be shown some diagnostic tests performed on the network components.

Here are some rules about the tests:
- The hash '#' next to the result indicates that the value is different from the expected one.
- The asterisk '*' indicates that the result can be anything.
- Each test can have subtets.
- Each test has a final result that can be 'Default', 'NotExecuted', 'PASS' or 'FAIL'. If all the tests are PASS then there are no problems. If a test is composed of subtets then to be PASS all the subtets must be correct. If a test is FAIL, NotExecuted or Default there are issues then is necessary to evaluate any possible error.

Each test is represented with the following format:
test{test name},result{final test result},report{tests performed and/or information about the results},error{any errors/info},had_error{'True' if there were errors, 'False' otherwise}

Your task is to analyze these tests to identify the presence of any problems, the possible causes and solutions. Obviously if all the results are PASS then there are no problems. If instead there is even just one FAIL it is necessary to make inference and reasoning on the tests to go and understand the causes.

Reply to any other sentence, phrase or question not related to this system with the following sentence:
Sorry, I can support you only in tasks related to this system.
```

In this situation there are three cases, one for each scenario.

- Scenario one:

```
OI llama3.2:3b Today at 10:19 AM
Based on the provided system architecture description and test results, I'll analyze the tests to identify any potential problems.

NAS Component:
The NAS component has two SATA Flash Module PBIT tests. Both tests are reported as PASS with all subtets also passing (PASS). There are no errors or had_error indications for these tests.

However, upon closer inspection, it's possible that there might be an issue with the disk configuration or firmware of the SATA Flash Module PBIT (Disk 1) and SATA Flash Module PBIT (Disk 2), as they have a report indicating PASS. This suggests that the test results are correct, but further investigation is needed to confirm the underlying cause.

SBC3 Component:
The SBC3 component has a single PBIT test with a result of PASS and a report indicating "PBIT successfully completed." There are no errors or had_error indications for this test.

Server and WS1 Components:
Both the SERVER and WS1 components have PBIT tests with results of PASS, reports of "PBIT successfully completed," and no errors or had_error indications. This suggests that these components are functioning correctly in terms of their PBIT functionality.
```

GWS1 Component:

The GWS1 component has two tests:

- test{GWS1 - Computer Unit PBIT} with a result of PASS.
- test{GWS1 - SATA Flash Module PBIT} with a result of PASS.

However, there are no reports or error indications provided for these tests. Given that both tests have the same result as the NAS and SBC3 components (PASS), it's possible that there is an issue with the reporting or configuration of this component.

Conclusion:

Based on the provided test results, there are no apparent issues with the SATA Flash Module PBIT functionality across all components (NAS, SBC3, SERVER, WS1). However:

- The NAS component has two tests reported as PASS without any errors or had_error indications. Further investigation is needed to confirm the underlying cause.
- There seems to be an issue with reporting or configuration of the GWS1 component, as no reports are provided for its PBIT tests.

Recommendations:

- Perform further testing on the NAS SATA Flash Module PBIT (Disk 1) and SATA Flash Module PBIT (Disk 2) to confirm their functionality.
- Investigate the issue with the GWS1 component's reporting or configuration.
- Consider performing a reboot or other reset operation on the GWS1 component to verify its PBIT functionality.

In this scenario the model correctly identified there are no issues, and all tests are successfully passed.

- Scenario two:



llama3.2:3b Today at 10:21 AM

Based on the system architecture description and the test results, here is my analysis:

Summary:

The system has issues with Power over Ethernet (PoE) Injection based Traffic Control (PBIT) for SBC3, WS1, and the SATA Flash Module of GWS1. The tests show that the start and stop requests for PBIT are failing with a "Cannot send request" error.

Possible Causes:

1. **PBIT configuration issues:** It's possible that there are configuration errors in the PBIT settings on SBC3 and WS1, which are preventing them from receiving the start and stop requests.
2. **Network connectivity problems:** There might be network connectivity issues between SBC3/WS1 and the GWS1 or NAS, which is preventing the start and stop requests from being sent successfully.
3. **Hardware or firmware issues:** It's possible that there are hardware or firmware issues with the SATA Flash Module of GWS1, which is causing the PBIT tests to fail.

Potential Solutions:

1. **Verify and correct PBIT configuration:** Check the PBIT settings on SBC3 and WS1 for any errors or inconsistencies. Ensure that the configuration matches the expected values.
2. **Check network connectivity:** Verify that there are no issues with network connectivity between SBC3/WS1 and GWS1/NAS. Use tools like ping or traceroute to check the connectivity.
3. **Inspect SATA Flash Module hardware/firmware:** Check the SATA Flash Module for any signs of physical damage or firmware corruption. Perform a factory reset or upgrade the firmware if necessary.

Additional Recommendations:

- Review the system logs for any errors or warnings related to PBIT or network connectivity issues.
- Check the documentation for the SBC3 and WS1 devices to ensure that they are configured correctly for PBIT.
- Consider performing a reboot of GWS1 and NAS after making any changes to their configuration or firmware.

System Components Status:

- GWS1: SATA Flash Module (PBIT) - PASS
- GWS1: Computer Unit (PBIT) - PASS
- SBC3: PBIT - FAIL
 - Error: Cannot send start request (Synthetic Notification)
- WS1: PBIT - FAIL
 - Error: Cannot send stop request (Synthetic Notification)
- NAS:
 - Computer Unit (PBIT) - PASS
 - SATA Flash Module PBIT (Disk 1) - PASS
 - SATA Flash Module PBIT (Disk 2) - PASS
- SERVER: PBIT - PASS

In this scenario the switch two is broken, so all the connected components cannot be reached and tests results for that are “Default” with the error “Cannot send start/stop request (Synthetic Notification)”.

The model correctly identified the test and so the components with problems. But it did not mention the cause in the possible causes, this result could be reached with a fine-tune or more examples with a larger model in few-shot.

- Scenario three:



llama3.2:3b Today at 10:23 AM

Based on the provided system components and tests, here's an analysis of the results:

1. **GWS1 - Computer Unit PBIT:** Result is PASS, which indicates that there are no issues with the computer unit of GWS1.
2. **GWS1 - SATA Flash Module PBIT:** Result is PASS, indicating no problems with the SATA flash module of GWS1. Since both components of GWS1 have passed, it's likely that the overall system is functioning correctly.
3. **NAS - Computer Unit PBIT:** Result is PASS, which means there are no issues with the computer unit of NAS.
4. **NAS - SATA Flash Module PBIT (Disk 1)** and **NAS - SATA Flash Module PBIT (Disk 2):** Both results are PASS, indicating that both SATA flash modules in NAS are functioning correctly.
5. **SBC3 - PBIT:** Result is PASS, which means the Power Business Continuity Interface Test for SBC3 was successful.
6. **SERVER - PBIT:** Result is PASS, which means the Power Business Continuity Interface Test for SERVER was successful.
7. **WS1 - PBIT:** Result is PASS, which means the Power Business Continuity Interface Test for WS1 was successful.

Based on the results of all tests passing (except none), it appears that:

- The computer units and SATA flash modules of GWS1, NAS, SBC3, SERVER, and WS1 are functioning correctly.
- There are no apparent issues with the overall system.

However, without further information or additional testing, I couldn't determine if there are any potential problems with other components (e.g., ports, interfaces) that are not being tested here.

In this scenario the SBC2 is broken. The SBC2 has been removed in this smaller sample system as consequence all the remaining tests are successful. The model correctly identified it.

I decided also to experiment with a different textual representation for the tests, the Json. This representation could be clearer and more understandable for the model.

- **Scenario one:**

```
o1 llama3.2:3b
Analysis of the diagnostic tests:

GWS1 - Computer Unit PBIT
  • Test result: PASS (OK)
  • Report:
    • BIOS Vendor: American Megatrends International, LLC.
    • BIOS Version: 5.19
    • BIOS Date: 03/14/2023
  • Error: -
  • Had error: False

GWS1 - SATA Flash Module PBIT
  • Test result: PASS (OK)
  • Report: PASS
  • Error: -
  • Had error: False

It appears that the GWS1 components have correct BIOS information and passed all tests.

NAS - Computer Unit PBIT
  • Test result: PASS (OK)
  • Report:
    • BIOS Vendor: American Megatrends Inc.
    • BIOS Version: 3.8b
    • BIOS Date: 01/05/2023
  • Error: -
  • Had error: False
```

NAS - SATA Flash Module PBIT (Disk 1) and NAS - SATA Flash Module PBIT (Disk 2)

- Test result: PASS (OK) for both tests
- Report:
 - PASS for both tests
- Error: -
- Had error: False

The NAS components also have correct BIOS information and passed all tests.

SBC3 - PBIT

- Test result: PASS (OK)
- Report:
 - PBIT successfully completed
 - UEFI Boot Mode ENABLED
 - BIOS Vendor: American Megatrends Inc.
 - BIOS Version: 4.03.01
 - BIOS Date: 11/24/2022
- Error: -
- Had error: False

SERVER - PBIT

- Test result: PASS (OK)
- Report:
 - PBIT successfully completed
 - UEFI Boot Mode ENABLED
 - BIOS Vendor: American Megatrends Inc.
 - BIOS Version: 4.03.01
 - BIOS Date: 11/24/2022
- Error: -
- Had error: False



WS1 - PBIT

- Test result: PASS (OK)
- Report:
 - PBIT successfully completed
 - UEFI Boot Mode ENABLED
 - BIOS Vendor: American Megatrends Inc.
 - BIOS Version: 4.03.01
 - BIOS Date: 11/24/2022
- Error: -
- Had error: False

The SBC3, SERVER and WS1 components have also passed the PBIT tests.

No problems found

All test results are PASS (OK), indicating that there are no critical issues with these system components.

In this scenario everything correctly works.

The model reported some tests and did not identified problems, correctly.

- Scenario two:



llama3.2:3b

Based on the provided tests and results, I will analyze the findings and identify potential problems.

Test Analysis

1. **SBC3 - PBIT:** The test is set to "Default" with an error message indicating "Cannot send start request (Synthetic Notification)" and another error message for "Cannot send stop request (Synthetic Notification)". Both errors are marked as having occurred ("had_error": True). This suggests a failure in sending notifications, likely due to a configuration issue or a problem with the SBC device itself.
2. **WS1 - PBIT:** Similar to SBC3, this test is also set to "Default" with error messages related to sending start and stop requests. Both errors are marked as having occurred ("had_error": True). This indicates another potential failure in notification processing.
3. **GWS1 - Computer Unit PBIT and NAS - Computer Unit PBIT:** These tests show a "PASS" result, indicating no issues found with the BIOS version or date for both devices.

Subtest Analysis

For SBC3 and WS1, without more detailed information on what exactly is causing the errors, it's difficult to pinpoint an exact cause. However:

- The fact that both devices have similar error messages suggests a common issue related to notification processing.
- It's likely that these issues are not due to hardware failures but rather configuration or software-related problems.

Potential Solutions

1. Review and verify the configuration for PBIT (Power and Bandwidth Management) on SBC3, WS1, and potentially other devices.
2. Check the device logs for any errors related to notification processing.
3. Verify that the UEFI Boot Mode is enabled on all affected devices (SBC3, WS1).
4. Consider updating or reconfiguring the BIOS version/date for GWS1 and NAS if necessary.

Server Analysis

The only server-related test is **SERVER - PBIT**, which shows a "PASS" result. This suggests that the server's notification processing is working correctly.

Switches Analysis

No switch tests are provided, but given the context, it's possible that switches might also be affected by similar issues related to PBIT or notification processing.

Recommendations

To resolve these issues:

1. Investigate and resolve any configuration errors related to PBIT on SBC3, WS1, and potentially other devices.
2. Review device logs for any errors related to notification processing.
3. Verify the UEFI Boot Mode is enabled on all affected devices.
4. Consider updating or reconfiguring BIOS versions/date if necessary.

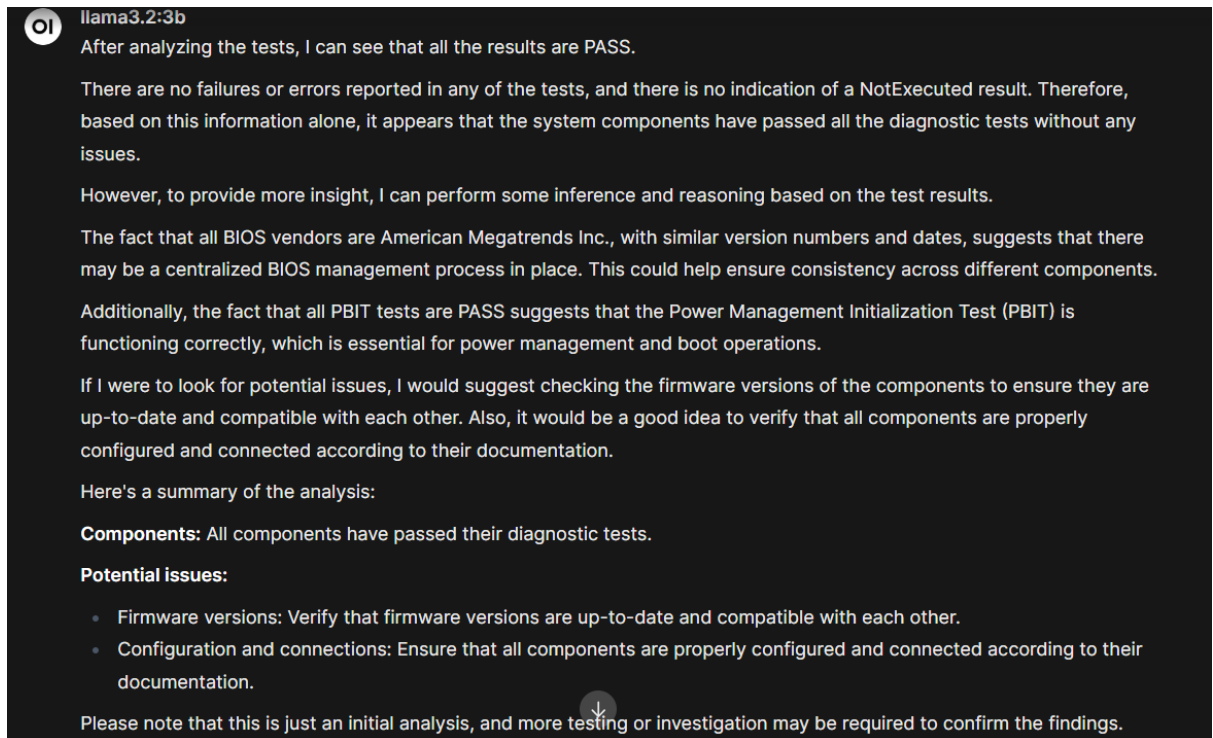
If these steps do not resolve the issues, further investigation may be required to identify a root cause and implement additional fixes or workarounds.

In summary, while there are some tests that show failures, it's likely due to configuration or software-related problems rather than hardware failures. By investigating and resolving these issues, we can ensure proper notification processing and mitigate any potential system disruptions.

In this scenario the switch two is broken with consequences on all its connected devices. I remind you that these types of tests do not regard the switches but only the computer units.

The model correctly identified the not working components and proposed possible solutions and recommendations for those.

- Scenario three:



In this scenario the SBC2 is broken. The SBC2 has been removed in this smaller sample system as consequence all tests are successful. The model correctly identified it.

The Json representation helped the model in understanding the tests and their results, combined with fewer data it led the model to better results compared to the previous experiments.

5. Conclusion

5.1 Summary and Conclusion

The few-shot examples should help the model in understanding its tasks and lead to better results. This is because the model has more knowledge on how it should analyse the diagnostics tests. However, if there is a great amount of data, in this case tests and their results, creates a large context input for a small model, making difficult for it a “complete understanding” of the situation and to focus on the real issue. In fact, I experimented with bigger commercial models like ChatGPT the model task, these enormous models can understand the situation and complete the task more accurately. In this case the size of the model leads the few-shot technique to worse performance than the cases with a general prompt with the inference logics only due to the huge amount of data for a small model.

Also, the number of tests has a great impact on the performance of this approach. In fact, with less data the model was able to better understand the situation and identify anomalies.

The applied solution demonstrated also the importance of the representation, in fact with Json the model was able to “understand” more. As it demonstrated that the performance widely depends on the model/input size.

In conclusion, this approach demonstrated that the LLMs could identify most of the errors making the diagnostics faster, but sometimes small LLMs have difficulties in reasoning and identifying the possible causes of these errors with many data due to their limited memorization and reasoning capacities.

5.2 Future Developments

As future development this work could be integrated into the HMI diagnostics software. For example, with a dedicated button could be possible to take the tests data for a use case and feed them to the model to receive a recap/analysis answer.

Furthermore, with more data could be possible to perform a fine-tune rather than in-context learning. A fine-tune could drastically improve the performance of this approach.

Bibliography

- [1] Fault isolation: https://en.wikipedia.org/wiki/Fault_detection_and_isolation
- [2] LLM4SecHW: Leveraging Domain Specific Large Language Model for Hardware Debugging: <https://arxiv.org/abs/2401.16448>
- [3] Structuring unstructured data with LLMs Article: <https://medium.com/@vivekvjnk/structuring-unstructured-data-with-llms-13a19810174d>
- [4] FD-LLM: Large Language Model for Fault Diagnosis of Machines: <https://arxiv.org/abs/2412.01218>
- [5] LogGPT: Exploring ChatGPT for Log-Based Anomaly Detection: <https://arxiv.org/abs/2309.01189>
- [6] LogLLM: Log-based Anomaly Detection Using Large Language Models: <https://arxiv.org/pdf/2411.08561v1>
- [7] An Automated Machine Learning Approach for Real-Time Fault Detection and Diagnosis: https://www.researchgate.net/publication/362748807_An_Automated_Machine_Learning_Approach_for_Real-Time_Fault_Detection_and_Diagnosis
- [8] Kitchenham and Charters methodology: <https://dl.acm.org/doi/10.1145/1134285.1134500>
- [9] Machine Learning for Anomaly Detection: A Systematic Review: https://www.researchgate.net/publication/351830421_Machine_Learning_for_Anomaly_Detection_A_Systematic_Review
- [10] LLMs: https://en.wikipedia.org/wiki/Large_language_model and <https://web.stanford.edu/~jurafsky/slp3/>
- [11] GPT: https://en.wikipedia.org/wiki/Generative_pre-trained_transformer
- [12] Prompt engineering: https://en.wikipedia.org/wiki/Prompt_engineering
- [13] Llama: [https://en.wikipedia.org/wiki/Llama_\(language_model\)](https://en.wikipedia.org/wiki/Llama_(language_model))
- [14] Phi-4: <https://www.microsoft.com/en-us/research/wp-content/uploads/2024/12/P4TechReport.pdf>
- [15] Quantization: <https://huggingface.co/blog/merge/quantization>