LLM-driven Malware Analysis across Code Representations

A Study on the Impact of Code Representation on the Performance of LLM-driven Malware Classification

Sil Folkertsma

Supervisors: Dr. S. Chakraborty, Dr. P. Pawełczak

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Sil Folkertsma
Final project course: CSE3000 Research Project
Thesis committee: Dr. S. Chakraborty, Dr. P. Pawełczak, Prof. dr. A. van Deursen

Abstract

In this study, we determine the extent to which code representation affects the accuracy of LLM-driven Malware analysis. Our results show that LLMs are definitively better at detecting malware in high-level source code than in binary code. For our study we conduct experiments on samples from the SBAN dataset. In the process we evaluate the validity of the SBAN dataset as a malware classification benchmark, directing future efforts towards improving dataset quality.

1 Introduction

Cybercrime and the use of malware have increased in recent years, with the FBI reporting a 33% increase in cybercrime losses in the US. Federal Bureau of Investigation [2025] Meanwhile, SOCs are facing a large number of false positives which are leading to increased employee burnout and turnover. Corelight Resources [2026]
In the light of this, there have been calls to improve malware detection tools such that SOC employees can spend less time investigating repetitive false positives, and more time investigating the true positives.

At the same time, LLMs have shown strong performance in code analysis, which has led to growing interest in their use for cybersecurity purposes. As offensive applications of these technologies keep evolving, the onus is on the defensive cybersecurity community to keep up. A variety of work has explored using LLMs for malware detection in source code, for example in Android applications et al. [2025a]. Other studies have explored using LLMs for decompilation purposes. et al. [2024]et al. [2026] As new models come out, these two subdomains are continuously iterated upon.

These efforts have called for an ample-sized supporting dataset that combines malware and benign samples across code representations. The SBAN dataset et al. [2025b] was proposed to fill this gap. In their accompanying paper, Jelodar et al. mention that their new dataset can be used "to assess the performance of models in tasks such as code summarization, malware classification, and cross-modal translation." et al. [2025b]. However, because massive multi-modal datasets like SBAN rely heavily on automated pipelines and LLMs to augment and accompany their pre-existing data, there remains a question of database quality. If the dataset contains erroneous samples, its use as a benchmark for target models becomes a shifting baseline.

The aim of this study thereby becomes two-fold. First, we address a gap behind the two aforementioned sub-domains: there is still limited evidence on how well LLMs perform when malware is represented at different abstraction levels. In particular, there are few studies that directly compare binary files, assembly code and source code using the same model family and evaluation setup. Because of this, it remains unclear whether LLMs are sufficiently adept at identifying malware in binary files, or whether a decompilation step is strictly necessary, for example. This research addresses this gap by comparing the performance of LLMs with regards to detecting malware in binary files, assembly code and source code. This is an interesting gap, because resources in defensive cybersecurity are relatively scarce, so not having to decompile suspected malware would be a welcome preservation of resources. Secondly, after using the SBAN dataset as a basis for experiments answering the first question, we aim to evaluate the internal validity of the SBAN dataset in its intended malware classification use-case.

The outcome of our research can be used to direct future efforts in the direction of one or more code representations depending on the results of the first question, while further indicating if additional work on malware datasets is required.

2 Problem Description and Methodology

In our experiments, we answer the following questions:

- RQ1: To what extent does code representation affect the accuracy of LLM-driven malware classification?
- RQ2: To what extent does the internal validity of the automated SBAN dataset support its reliability as a benchmark for LLM-based malware classification?

This section outlines the general methodology used to answer these research questions. Section 2.1 describes the preprocessing steps taken to prepare data samples. Section 2.2 describes the methodology used to audit the SBAN dataset. Lastly, section 2.3 describes the evaluation setup for comparing code representations.

2.1 Data Pre-processing

We specifically chose the MalwareBazaar subsection of the SBAN dataset. This part consists of recent samples that have been shared in the information security community, which means that our samples will more closely resemble the variety of files a modern-day SOC encounters. We hereby thank the authors of the SBAN dataset and the corresponding paper for allowing us to use their work in our research.

We first ventured to combine the various columns of the MalwareBazaar subsection into one json file. This required a few cleaning steps:

- We stripped the IDs in the 'evidence' file since they had ".c" appended.
- We fixed the JSON in the 'evidence' file since it was widely missing commas.
- We renamed columns that bore the same name but were different (source_code and binary_code, for example).
- We dropped duplicate columns to facilitate merging.

We then combined binary, assembly, source, and evidence into one .jsonl file. Next, we removed any lines that were labeled "UNKNOWN" or had empty evidence columns. Initial experiments with respect to RQ1 revealed that further cleaning was required. Many samples in the dataset contained source code that only said "Please put the full source code here" or something similar. We used a static keyword filter to remove these samples from our pool. This ensured that our samples were clean, but the multitude of cracks that appeared during the cleaning and initial experimentation stage led us to conduct a deeper investigation into the quality and validity of the SBAN dataset.

2.2 Evaluating the Benchmark Validity of SBAN

Our investigation mainly concerned the labeling of the SBAN dataset. Manual investigations raised suspicions about the contrast between the source code and the final label, as well as the evidence used to corroborate the final label. We decided that a more thorough investigation was warranted. This investigation and subsequent evaluation were performed using an LLM-as-a-judge process. We evaluated some alternative methods:

- Looking up the hash of the samples on publicly available resources such as MalwareBazaar. Although this is a logical option given that MalwareBazaar was the original source of these samples, the processing performed by the authors of the SBAN dataset meant that the hashes are no longer the same as the hashes of the original malware.
- Compiling and running the code in a closed sandbox to investigate its behavior and evaluate its label. Although interesting, there are two reasons why we did not choose to do this. First, the processing done by the authors of the SBAN dataset means that many samples might fail to compile entirely, which means that we would have to compromise the alignment of the source code with the other code representations, damaging the integrity of our research. Second, setting up a sandbox that can safely execute malware is quite complicated and does not fit the scope of this research.

Thus, an LLM-as-a-judge process - given its flexibility - remained the most viable alternative. We set up our judge pipeline as follows:

- We give one sample to the LLM that contains only source_code, evidence and final_label.
- We tell the LLM to act as an independent auditor and to examine the source code and the evidence. After doing so, it should verify whether the features in the evidence text descriptions are actually present in the source code and whether the final label is correct.
- The output is given as a label "CORRECT" or "INCORRECT". As such, it is a zero-shot classification task.

We can then repeat this experiment for the rest of the dataset, after which we can aggregate the results and answer our question concerning benchmark validity. In the meantime, we can use the outputs to filter out incorrectly labeled data such that we can continue our experimentation towards answering RQ1 with correctly labeled data.

2.3 Evaluating the Impact of Code Representation on Classification Accuracy

To determine the effect of code representation on the accuracy of LLM-driven malware analysis, we execute the following pipeline:

- We sample malware and benign code in different code representations from a clean, validated pool of data samples.

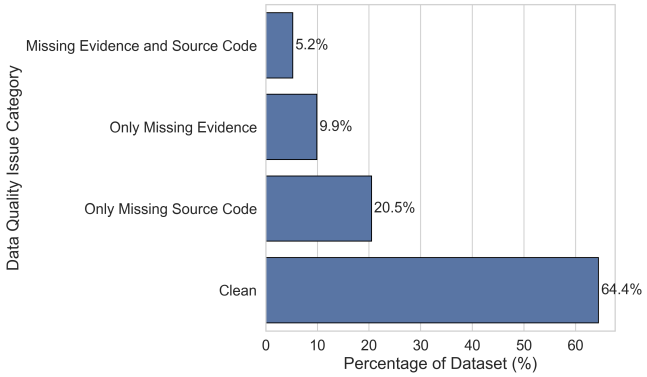

Figure 1: Dataset Cleaning Report

- We prompt an LLM with one sample in one representation at a time and tell it to classify it as 'MALWARE' or 'BENIGN'.
- We compare the output of the LLM to the actual label found in the final_label column.
- We aggregate the results by code representation and compute various accuracy metrics.

We can then answer our research question by comparing the accuracy metrics across code representations.

We chose a zero-shot classification approach for this task, since the underlying motive of this research is to reduce resource usage for malware analysis. Zero-shot prompting is a good starting point. We use a standardized, neutral prompt design, prompting for a final label while minimizing bias towards either of the labels. Performance is measured using three distinct metrics:

- Accuracy, which indicates the percentage of labels assigned that were assigned correctly. This gives a general indication of correctness.
- Precision, which indicates the percentage of samples that our model labeled "MALWARE" which were actually malware. High precision indicates that we are suffering relatively few false positives, which is good for resource conservation.
- Recall, which indicates the percentage of actual malware samples that our model ended up labeling "MALWARE". High recall means that our model mislabels relatively few malware samples, which is crucial. Any mislabeled malware samples could have catastrophic consequences in a real-world setting. As such, our focus will be on recall as a measure of our model's performance.

3 Experimental Setup

3.1 Model Selection

The LLM we used in our study had to fulfill the following requirements:

- It needs to be trained on software, such that it has knowledge of code.
- It needs to be open-source, such that results will be easier to reproduce.

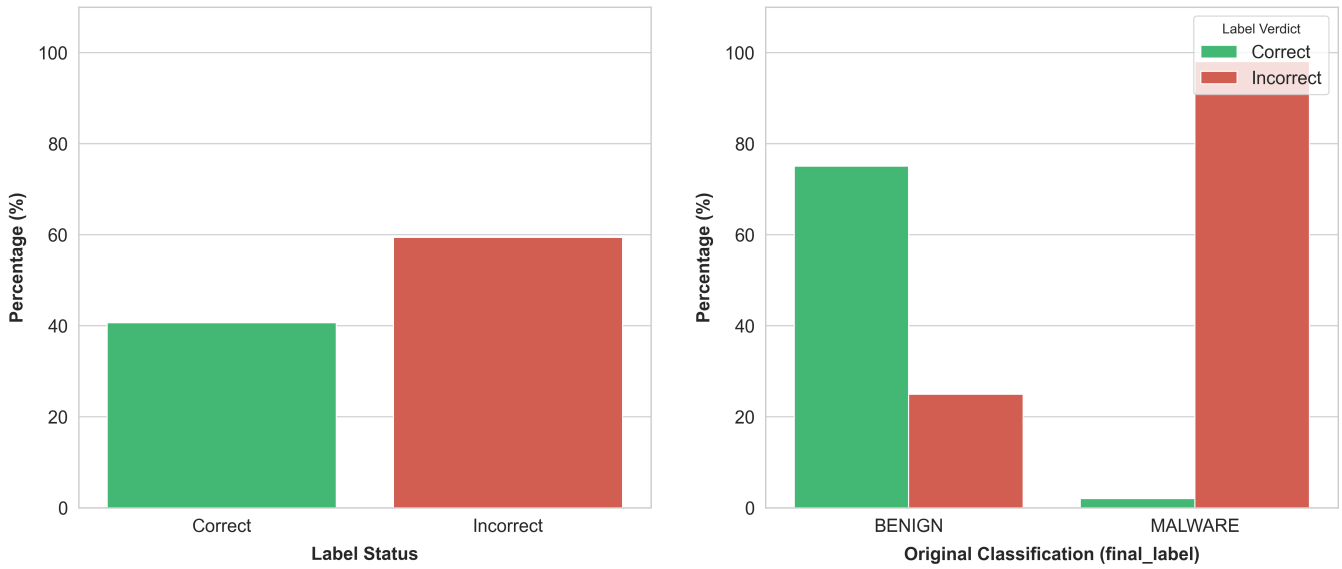

Figure 2: Label Validation Experiment Report. The left shows label correctness over the entire dataset. The right shows label correctness broken down by original classification

- It needs to be small enough to fit on a (relatively small) DelftBlue NVIDIA A100-small partition.

We chose the Qwen model family since it has been used in related papers for similar purposes and fit all of the above requirements. Specifically, we ran our experiments on Qwen3.5 9B, which we quantized to make it fit on the DelftBlue partition. We used 4-bit NormalFloat (NF4) weight compression with double quantization. Temperature was kept at T=0.1, again, to maximize reproducibility.

3.2 Hardware Environment

The LLM was set up on DelftBlue, and all experiments were executed on the DelftBlue A100-small partition. DelftBlue was chosen as an alternative to our own hardware, as it allows us to scale our research to a larger model and data sample. We have detailed the python environment - including module versions - at https://github.com/MrMovey/llm-malware-analysis/blob/main/README.md

4 SBANished: a dataset evaluation

In the paper that accompanies the SBAN dataset, the authors claim their dataset can be used "to assess the performance of models in tasks such as code summarization, malware classification, and cross-modal translation." et al. [2025b]. In our efforts to use it to assess the performance of a model in malware classification, we encountered some hiccups. This led us to conduct a more thorough evaluation of the SBAN dataset, specifically in the context of this malware classification use case. We will first discuss more surface-level dataset quality issues like missing evidence and source code. Then we will dive deeper into the issue of hallucinated evidence.

4.1 Missing fields

The first problem we encountered in the dataset was that of missing fields. We highlighted this briefly in chapter 2.1, but we will now give a more detailed breakdown. Missing fields were found in two important columns:

- evidence: the evidence field is often simply empty, leading to final_label="UNKNOWN". The corresponding samples are thus entirely unusable for a classification experiment.
- source_code: this field often contains phrases like "paste your full source code here" without any further source code. These samples often still have labels and evidence attached that are entirely hallucinated by an LLM. We filter out corresponding samples using a static keyword filter: "paste the full source code", "insert source code", "paste code", "your code", "add main function", "code goes", "implementation goes", "insert code", "no modifications needed", "corrected", "here", "no comments", "additional"

We have visualized the percentage of samples affected by these missing fields in Figure 1. This experiment was executed on the entirety of the MalwareBazaar section of the SBAN dataset.

Figure 1 shows us that 20.5% of samples are only missing the source code. For these samples, the LLM used by the SBAN authors still generated evidence but did not have any source code to base that evidence on. This raised concerns that there might be more hallucinated evidence in the remaining samples that did have source code. Our next experiment will further explore this possibility.

4.2 Hallucinated evidence

To evaluate the remaining labels and evidence in the dataset, we set up an LLM-as-a-judge with the prompt listed in appendix A. We are mainly interested in the accuracy of the 'final_label' field. We executed the experiment on a random sample of size 1000, where 500 were labeled "BENIGN" and

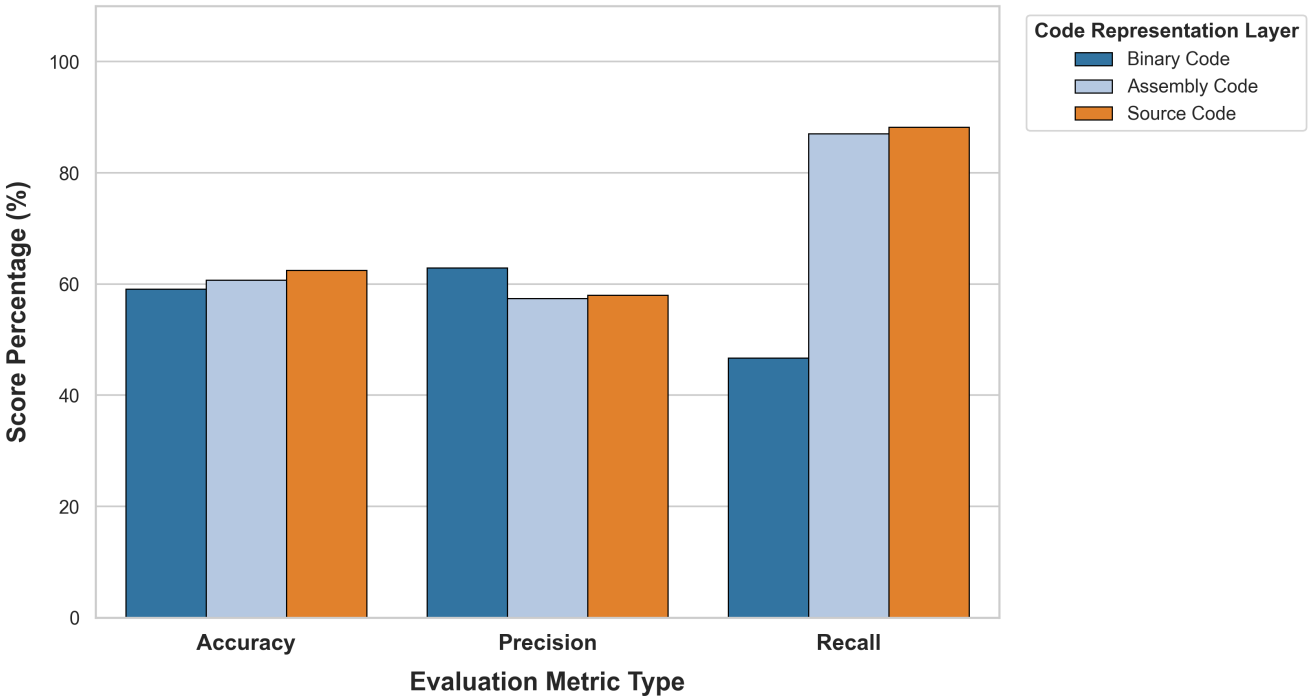

Figure 3: LLM-driven Malware Classification across Code Representations, Averaged over 3 Runs

500 were labeled "MALWARE". The results are displayed in figure 2.

Figure 2 shows that some 59% of all samples have been labeled incorrectly. We can proceed with our malware classification experiments using the 41% correct samples, but there is another issue. The second plot in Figure 2 shows a breakdown of the same experiment by original classification label. We can see that 98% of the samples originally labeled "MALWARE" were correctly labeled as such. This leaves us with very few malware samples to experiment with. In chapter 5 we will discuss how we mitigated this.

We tested the consistency of our LLM-as-a-judge process by running it three times on the same data sample. These runs resulted in the same label in all three runs for around 99% of samples.

5 Cross-Representation Malware Classification

Our dataset cleaning and quality filters left us with very few remaining malware samples for the experiments in this chapter. We mitigated this by repeating the LLM cleaning experiment we detailed in chapter 4.2 multiple times, extracting as many correct malware samples as we could. We also extended our sampling into the BODMAS sub-section of the dataset, which has similar characteristics to the MalwareBazaar part. As the accuracy of the malware labels remained between 1 and 2%, our sample total ended up at 121 after processing more than 10000 samples. We reduced the amount of benign samples accordingly, since over-representing benign samples would lead to bias in measuring accuracy. This means that we conducted the following experiments on 242 samples, with a 50/50 benign/malware split.

We prompted the LLM with one sample of one code representation at a time. The prompt used was the same for all parts of the experiment and can be found in appendix B. The prompt was designed to be as neutral as possible, so that bias towards either label would be minimized.

The result of the experiments is shown in Figure 3.

The graphs shown in Figure 3 have been generated on the basis of three runs for each code representation. Inconsistencies between runs were quite uncommon, which we have visualized in Figure 5. In case of inconsistencies, we determined the label based on the mode of the label assigned by the LLM.

5.1 Impact of Prompt

To challenge our results, we repeated our experiment, changing only the prompt. Our updated prompt aims to leverage a form of Chain-of-Thought prompting. et al. [2022] Our technique makes the LLM output reasoning before delivering a final verdict. This allows the LLM to do a limited amount of 'thinking' before it returns a label (malware or benign, respectively). The prompt used in this experiment is shown in appendix C. All other parameters and data used for this experiment are identical to those used for the other malware classification experiment in this paper.

The result of this experiment is shown in Figure 4. The recall in this experiment is slightly worse than in the experiment with the original prompt, but the discrepancy between binary code and the other code representations is still clearly

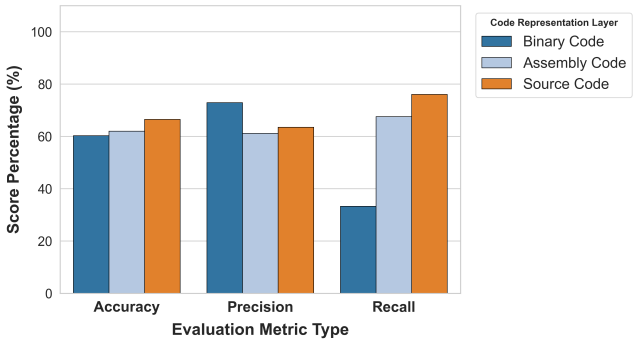

Figure 4: LLM-driven Malware Classification across Code Representations with Chain-of-Thought Prompting. Averaged over 3 Runs. See chapter 5.1.

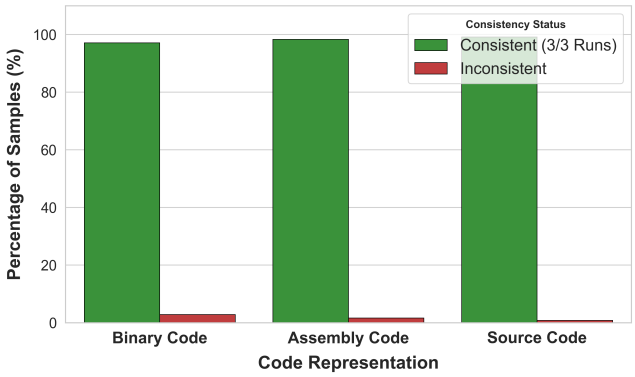

Figure 5: Consistency of LLM-driven Malware Classification across three runs

visible.

6 Responsible Research

In this chapter, we will detail the steps we have taken to ensure the reproducibility of this study. Since the results and conclusions are based on the output of LLMs, there are some concerns about the consistency of these results that should be addressed. In the last section of this chapter, we will discuss some ethical aspects of the outcome of this paper.

6.1 Reproducibility of Results

LLMs, in general, are black boxes; given any input, the output often cannot be predicted with complete accuracy. Furthermore, given any output, we often cannot determine exactly why the LLM generated the output as it did. One may wonder if this means that LLMs are entirely ineligible for use in malware detection, but the answer to that question is outside the scope of this paper.

To mitigate the risk of inconsistent results, we repeated our experiments multiple times:

- The LLM-as-a-judge process used in chapter 4.2 was repeated three times on the same data to evaluate its consistency. In 99% of samples, it returned the same label across all three runs. This result makes us confident that our experiment is reproducible.
- The LLM used as a malware classifier in chapter 5 has also been prompted with the same data samples three times. As we discussed in that chapter, we combined these three runs for our graphs in Figure 3. The results were remarkably consistent across these runs, which is shown in Figure 5, with more than 97% of the labels being exactly the same across all three runs. This increases our confidence that these results are reproducible.
- We repeated the classification experiment with a different prompting technique. This experiment was discussed in chapter 5.1. It resulted in significantly worse recall on binary code than on other code representations, which means the effect we saw in our first experiment also holds for a different prompt and prompting technique. This increases our confidence in the robustness of our conclusion.

Furthermore, we picked an open-source model that we ran locally without persistent memory features across runs. This ensures that the model does not learn from our samples as it goes, nor does our data get used to train the model, impacting future attempts at reproducing our results. Additionally, we have greater control over the circumstances in which it was run, which we have reported in chapter 3.1. Lastly, all the code used for this research is available on GitHub at https://github.com/MrMovey/llm-malware-analysis/

6.2 Ethical Aspects

In the field of malware analysis, new technologies and methods need to be audited with significant rigor before they can be deployed, since failures in malware detection could have catastrophic consequences for the recipient of the missed malware. In the previous chapter, we touched upon the question if LLMs are thereby legible at all to be used in malware detection. Certainly, this paper does not provide a definitive answer to that question. We believe the results to be promising, but our experimental setup - in its current form - is not ready to be deployed in a Security Operations Center. Recall would have to approach, if not hit, 100% for this to be the case. Further research will thus have to improve the performance of LLM-driven malware detection, or dismiss its role in this use case entirely.

6.3 Use of AI

AI was not used at any point for the purpose of writing this paper. All conclusions in this paper are entirely our own, and we assume full responsibility for our work.

AI was used to implement some of the code used in this project. We subsequently edited this code. Examples include the following:

- Code for generating graphs and figures found in this paper.
- Scanning the libraries available on DelftBlue and aligning our LLM scripts with the available versions.
- Code for pre-processing of the SBAN dataset.

7 Discussion

7.1 Broader Context

The results of this study are perhaps somewhat unsurprising; LLMs are generally trained on source code, and not on binary code, and their relative struggles in classifying binary code is therefore to be expected. Still, we consider it important to test such preconceptions, especially considering that LLMs sometimes show emerging behavior. The exact reason for the worse performance of LLMs on binary code remains unclear. In their work on vulnerability detection, Böke and Torka conclude that obfuscation techniques significantly impact LLM performance when identifying vulnerabilities. Böke and Torka [2026] We do not know the degree of obfuscation present in binary samples from the SBAN dataset, but the effects of obfuscation identified by Böke and Torka in vulnerability detection are likely to also occur in malware analysis, especially since attackers can enable these obfuscation techniques to avoid detection.

7.2 Limitations

We encountered two main limitations in our experiments. First, cleaning and validating the samples in the SBAN dataset had drastic consequences for the number of remaining malware samples. As a result of this, our experiment has been run on a reduced number of samples. This also meant that we decided against changing our prompts based on the results. Such changes would have led to overfitting and artificially inflated results. This limits the scope of this work to identifying a gap between binary code and other representations. As such, investigating why this gap exists is left for future research. Second, our access to DelftBlue was practically limited to the A100-small partition. This meant that the size of the model that we could run was limited to around 9B parameters (quantized). Despite these limitations, we are confident in the validity of our conclusions based on these experiments, given the margin by which the assembly code and source code experiments outperformed the binary code experiment. Future research could reach better accuracy using a better model, but improving accuracy is not the main goal of this study. We are similarly confident that these limitations did not significantly affect the validity of our audit of the SBAN dataset, given that we used a model in the same model family (Qwen) as the authors of the SBAN dataset, but our experiments used a more modern version with more parameters.

8 Conclusions and Future Work

This study set out to answer the following questions:

- RQ1: To what extent does code representation affect the accuracy of LLM-driven malware classification?
- RQ2: To what extent does the internal validity of the automated SBAN dataset support its reliability as a benchmark for LLM-driven malware classification?

As discussed, our most important accuracy metric is recall. From our results in Figure 5, we can see that the recall of the LLM was significantly higher for assembly and source code than it was for binary code. Thus, we can conclude that code representation has a significant impact on the accuracy of LLM-driven malware classification, with malware in source code being twice as likely to be recognized as malware in binary code. Future work in this area should therefore further explore LLM-driven malware classification in source code. We recommend using larger, state-of-the-art models that have the potential to push the recall boundaries. To accommodate successful source code malware detection in the industry, the research community should continue its efforts towards effective and efficient decompilation pipelines, where the LLM could play a role. Additionally, future research could dive deeper into why the LLM performs worse when classifying malware in binary code than in other code representations, and if there are possible remedies against this discrepancy. In this study, we decided against significantly adapting our prompt based on the results, as we wanted to avoid overfitting. However, a more supportive dataset could enable such efforts. This leads us to our next conclusion.

After auditing the SBAN dataset, our conclusions are twofold. First, the dataset has significant data quality issues, rendering significant portions unusable for malware classification experiments. 36% of MalwareBazaar samples had missing fields (Figure 1. 59% of remaining samples were labelled incorrectly. Furthermore, malware samples were especially affected, with some 98% of malware samples carrying incorrect labels. This makes the use of this dataset for experiments on malware analysis and classification cumbersome at best. We therefore conclude that the internal validity of the SBAN dataset does not support its reliability as a benchmark for LLM-driven malware classification Given the relevance of datasets that include malware across code representations, for example in decompilation research, we recommend future efforts to concentrate on improving the SBAN dataset or constructing a new alternative. We recommend that any such efforts be focused primarily on the quality of the dataset, given that we were able to run a successful experiment on a smaller sample once the quality was improved.

A Prompt for LLM-as-a-judge

"You are an expert, objective malware analyst and software code auditor. "
"Your task is to evaluate the technical accuracy and internal validity of an automated software dataset."
"You will be provided with three specific fields:"
"[SOURCE CODE] - A snippet of decompiled pseudo-C code or an isolated software function"
"[DATASET EVIDENCE] - A text snippet included in the dataset that asserts specific evidence concerning malicious or benign behaviour in the source code"
"[FINAL LABEL] - The final label assigned in the dataset to this piece of source code. This is either MALWARE or BENIGN"
"INSTRUCTIONS:"
"1. Closely analyze the [SOURCE CODE]."
"2. Audit the [DATASET EVIDENCE]. Does the code show the behaviour claimed by the text?"
"3. Evaluate the [FINAL LABEL]. Is this label justified given the source code and evidence?"

"Your final output must adhere strictly to the following text format. Do not include any markdown blocks, conversational filler, or additional labels:"
"ACCURACY: <an integer from 0 to 100 where 100 means the evidence is fully true to the code, and 0 means it is completely fabricated>"
"CORRECTNESS: <CORRECT/INCORRECT depending on your evaluation of the FINAL LABEL>"
"ELABORATION: <a one-to-two-sentence technical justification supporting your audit>"

B Prompt for Classification Experiments

The system prompt for these experiments was the following: "You are an expert malware researcher. Classify code as MALICIOUS or BENIGN. Your final answer must explicitly follow this format: RESULT: <MALICIOUS/BENIGN> REASONING: <brief summary>"

The user prompt (repeated for each sample): "Analyze this code representation: target_code" Where target_code was replaced with the code representation sample.

C Chain-of-Thought Prompt

The system prompt for these experiments was the following: "You are an expert malware researcher. Classify code as MALICIOUS or BENIGN. Your final answer must explicitly follow this format: REASONING: <brief summary> RESULT: <MALICIOUS/BENIGN>"

The user prompt (repeated for each sample): "Analyze this code representation: target_code" Where target_code was replaced with the code representation sample.

References

Federal Bureau of Investigation. 2024 IC3 annual report, 2025. URL https://www.ic3.gov/AnnualReport/Reports/2024_IC3Report.pdf. Accessed: 2026-04-21.

Corelight Resources. False positives in cybersecurity: Causes, costs, and fixes, 2026. URL https://corelight.com/resources/glossary/false-positives-cybersecurity. Accessed: 2026-06-02.

Tegar Ganang Satrio Priambodo et al. MalQwen: Fine tuned LLM for static android malware analysis report. *IEEE Access*, 13, 2025a. doi: 10.1109/ACCESS.2025.3637047.

Hanzhuo Tan et al. LLM4Decompile: Decompiling binary code with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. URL https://doi.org/10.18653/v1/2024.emnlp-main.203.

Hamed Jelodar et al. Large language model (LLM) for software security: Code analysis, malware analysis, reverse engineering. *Journal of Information Security and Applications*, 2026. URL https://doi.org/10.1016/j.jisa.2026.104390.

Hamed Jelodar et al. SBAN: A framework & multi-dimensional dataset for large language model pre-training and software code mining. In *Proceedings of the 2025 IEEE International Conference on Data Mining (ICDM)*, 2025b. URL https://doi.org/10.1109/icdm65498.2025.00138.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022. URL https://dl.acm.org/doi/10.5555/3600270.3602070.

Ekin Böke and Simon Torka. "digital camouflage": The llvm challenge in llm-based malware detection. *Journal of Systems and Software*, 2026. URL https://doi.org/10.1016/j.jss.2025.112646.