LLM-Based Autonomous Agents for Dynamic Malware Analysis

Research paper

Thomas Crull

Supervisor(s): Przemyslaw Pawelczak, Soham Chakraborty

[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: Thomas Crull
Final project course: CSE3000 Research Project
Thesis committee: Prof. Przemyslaw Pawelczak, Prof. Soham Chakraborty, Prof. Arie van Deursen

Abstract

Dynamic malware analysis produces large amounts of behavioural evidence, which can be difficult to interpret manually and too large to process directly with small Large Language Models (LLMs). This paper evaluates to what extent Qwen3-4B can distinguish between benign and malicious Windows executables using reduced CAPEv2 dynamic-analysis reports. To test this, we built a sandbox pipeline which executes samples in a Windows 10 Pro detonation VM, collects CAPEv2 reports, filters them down to the most relevant dynamic-analysis information, and feeds them to Qwen for classification. The reduced reports retain the process tree, domains and DNS activity, behavioural signatures, and their ATT&CK TTP and Malware Behavior Catalog mappings. The dataset consisted of 1082 malware samples from MalwareBazaar and 762 benign samples collected from PortableApps, PortableApps installers, the Sysinternals Suite, and Benign-NET. Two prompts were tested: one with only benign and malware as possible verdicts, and one which also allowed an inconclusive verdict. The first prompt achieved 73.01% recall and 60.91% precision, showing that Qwen could detect many malware samples but also misclassified many benign samples as malicious. The second prompt did not solve this issue, since more correct classifications became inconclusive than incorrect ones. Overall, Qwen3-4B shows some potential for dynamic malware analysis, but its high false positive rate makes it unsuitable as a standalone classifier without further improvements or fine-tuning.

1 Introduction

Dynamic malware analysis is one of the main methods for detecting and examining malware, which consists of observing the actions a program performs on the machine upon execution [1]. This method is commonly used to address the limitations of static analysis, particularly when analysing malware that uses obfuscation techniques [2].

To perform dynamic analysis, a specialist typically sets up a sandbox environment, often in the form of a virtual machine (VM), which allows malware to be executed without risking the host system. The malware is then executed while monitoring tools are used to observe its interactions with the operating system and network to determine if it is malicious [1] [2]. However, monitoring tools often generate large amounts of data, such as system calls, file modifications, and network activity logs, which take significant time and effort to interpret, even for an expert in the field.

Large Language Models (LLMs) have recently been explored as tools for automating parts of malware analysis. However, existing approaches often rely on large models, fine-tuning, or substantial computational resources. In this paper, we instead evaluate whether a small, locally deployable LLM can support malware detection in a constrained setting. Specifically, we use Qwen3-4B [3] [4] to classify samples based on reduced CAPEv2 dynamic-analysis reports. These reports are generated by executing samples in a sandbox and reducing the resulting output to the parts most relevant for classification, such as the process tree, behavioural alerts, network activity, and mappings to known malware behaviour categories. This reduction is necessary because full sandbox reports can be too large to process under the available computational constraints.

This leads to the following research question:

> **RQ:** To what extent can Qwen3-4B distinguish between benign and malicious Windows executables using reduced dynamic-analysis reports?

The four main contributions of this paper are a dataset for benign and malware Windows executables, a reproducible experiment which answers our research question, the findings resulting from the experiment, and a malware analysis setup for analysing Windows executables, which can be easily expanded for general Windows malware and other operating systems to facilitate future research within the field of static and dynamic malware analysis.

2 Related Work

There exists a small but growing body of research on the integration of LLMs in malware analysis systems. Nevertheless, there are no research papers on our specific topic. The two most closely related studies are the proposal made by Feng et al. for LLM-MalDetect [5] and the analysis made by Lang and Schreck on the use of LLMs for malware detection in memory forensics [6]. Both help us understand the potential and the limitations of LLMs within the field of malware analysis.

Feng et al. introduced LLM-MalDetect [5], an LLM-based Android malware detection framework that extracts APK features such as permissions, API calls, and strings, converts them into a textual representation, and uses the Mistral-7B model [7], fine-tuned with LoRA [8], for classification. Their results show that LLM-MalDetect can outperform several traditional machine learning and deep learning baselines, demonstrating the potential of LLMs for Android malware detection. On the Drebin dataset [9], the framework achieved its best results, with an accuracy of 98.97%, precision of 97.26%, recall of 98.33%, and F1 score of 97.79%, outperforming random forest, image-based methods, the AMDDLmodel [10], and DroidMDetection [11]. Regardless, their LLM was fine-tuned and their setup exceeds ours in computational power, although it is arguably still constrained when compared to industry standards.

Lang and Schreck investigated the use of LLMs for malware detection in memory forensics, which is related to dynamic analysis because it analyses runtime artefacts [6]. Their study integrated LLMs into a workflow based on Volatility3 [12], where memory artefacts such as process listings, network connections, loaded modules, suspicious memory regions, and command-line data were analysed by several

models, including GPT-4o, OpenAI o1, Gemini 2.0 Flash, Gemini 2.0 Flash-Thinking, Grok 3, and Grok 3-Thinking. All models used had significant computational power available since they were hosted by their providers, but were not fine-tuned. Their results showed that models with reasoning disabled generally performed worse than with it enabled. They also showed that all models, with or without reasoning enabled, had an accuracy above 96%, indicating their potential.

However, all models achieved precision scores under 22%, indicating that they flagged many benign processes as malicious. The authors attribute this to the tendency of LLMs to err on the side of caution and hallucinate threats whenever faced with ambiguous evidence to protect the user from incorrect classifications of malware samples. Furthermore, they argued that the limited context windows of current LLMs inhibit proper classification, since the evidence produced by Volatility3 tends to be too large and complex for them to handle [6].

3 Methodology

To answer our research question, we ran an experiment which required four separate parts to work together:

- A sandbox environment in which malicious executables could be safely executed without permanently tainting the host system. Also, the sandbox had to include an array of malware analysis tools to capture the behaviours displayed by the executable, producing reports which can be fed to an LLM for classification.
- A preprocessing step for filtering out the noise and static analysis information from the reports, as some monitoring tools are both for static and dynamic analysis and there may be noise present from the operating system's native apps. Moreover, the reports had to be shrunken down at this step as the logs produced by most tools were too large for our limited LLM environment.
- A computationally constrained LLM environment for hosting the model.
- A dataset containing both benign and malicious Windows executables.

For these reasons, this section is divided into four components. Section 3.1 describes the sandbox used for detonating the executables and why CAPEv2 was chosen as its orchestrator. Section 3.2 describes the filtering script used to reduce the analysis reports. Section 3.3 describes why Qwen was the chosen LLM and the setup of its environment. Section 3.4 describes the dataset construction process. The general overview of the experiment can be seen in Figure 1.

3.1 Sandbox

The first part of the experiment was the dynamic malware analysis sandbox. To run the experiment, each sample had to be executed inside a controlled Windows environment, monitored while it was running, and then discarded so that the next sample could start from a clean state. Doing this manually hundreds of times would not be realistic, therefore an orchestrator was necessary.

For this purpose, we used CAPEv2 [13]. CAPEv2 is a malware analysis sandbox based on Cuckoo, but with active development and more modern support. Earlier research papers in this area often mention Cuckoo [14] as a common sandbox choice [15] [16] [17]. However, Cuckoo itself has not been actively maintained for several years [14], possibly making it vulnerable to obfuscation or escape. Since the setup would execute real malware samples and had to remain stable throughout hundreds of analyses, using a maintained derivative was the safer option. CAPEv2 was therefore chosen as the main orchestrator.

CAPEv2 starts the analysis virtual machine, injects the submitted sample, runs the built-in and configured monitoring tools whilst hiding them from the malware to deter obfuscation, collects the produced logs, and restores the machine to its previous snapshot after the analysis is finished. To introduce this orchestrator we had to run Linux [13] which we could have installed natively on our laptop, but we opted for nested virtualisation to add another layer of security, see Figure 2. The physical host was a Lenovo laptop running Microsoft Windows 11 Pro, with an Intel Core Ultra 7 265H processor and 64 GB of RAM. On this host, VMware Workstation Pro 25H2 was used to run the CAPEv2 recommended Linux distribution Ubuntu 24.04.4 virtual machine [13]. It was assigned 16 virtual CPUs, 32 GB of RAM, and a 600 GB virtual disk. Inside this Ubuntu VM, the actual malware detonation machine was run using KVM/QEMU and libvirt, as per the recommendations of the CAPEv2 team [13]. The detonation machine was a Microsoft Windows 10 Pro VM with 4 virtual CPUs, 8 GB of RAM, and a 100 GB virtual disk.

The Windows 10 detonation VM was the only environment in which samples were executed. We disabled most security features to allow the malware samples to run unhindered. Unnecessary Windows features and background activity were disabled where possible, because normal operating-system noise pollutes the analysis reports. Similar to the discoveries of Lang and Schreck [6], if the report is full of unrelated Windows activity, the LLM would potentially base its classification on the noise instead of the actual behaviour displayed by the sample.

The detonation VM was not entirely debloated. Whilst debloating the system, we broke Windows Update, making it impossible for the automatic update to be stopped. This constrained us from further debloating the system and adding two more analysis tools, Procmon [18] and Sysmon [19], as they required a restart which would trigger an update, undoing most work done on disabling security features. As such, Windows Update, Windows Defender, SmartScreen and Msftconnecttest still produced noise in the final reports. To fix this issue, the entire detonation VM would have to be redone which was unfeasible due to the time constraint.

The network configuration was another important part of the setup. Many malware samples attempt to communicate with external services after execution. Fully blocking the network would hide this behaviour, while allowing real internet access would be unsafe and ethically questionable. To solve this, the setup used a fake-internet approach with INetSim

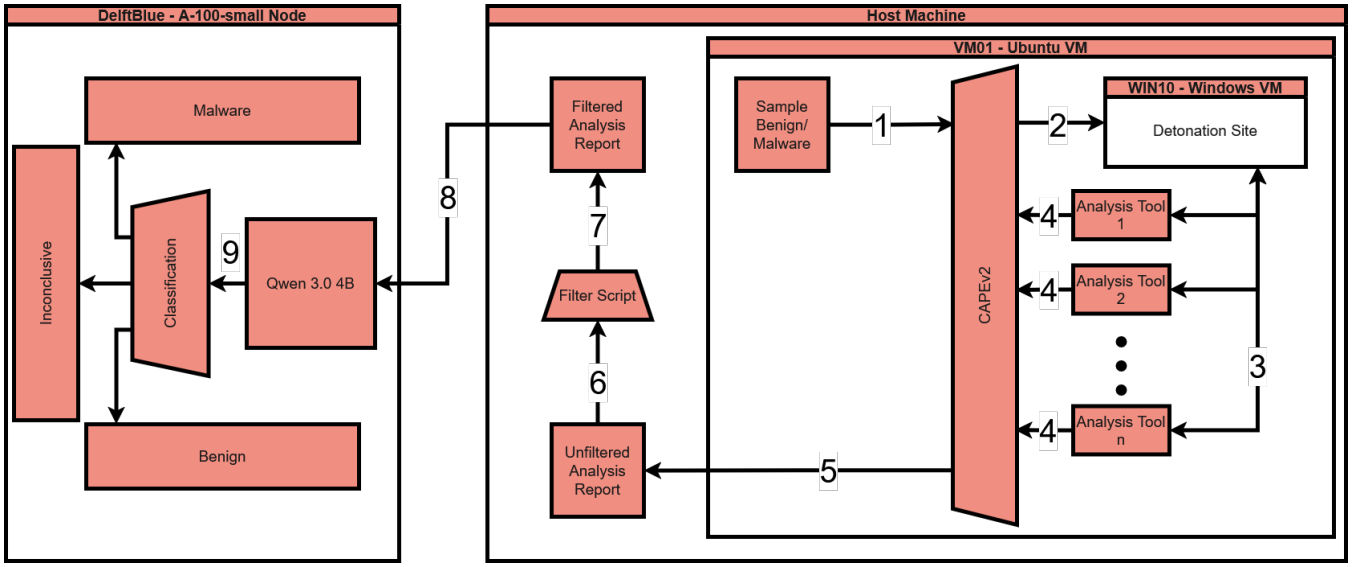

Figure 1: Experiment procedure. The benign/malware sample is first submitted to CAPEv2 for analysis. CAPEv2 then injects both the executable and its analysis tools inside the Windows VM. After the executable finished running, the logs produced by the analysis tools are extracted to the host machine in the form of a report. The report is then filtered, reduced down, and fed to Qwen for classification.

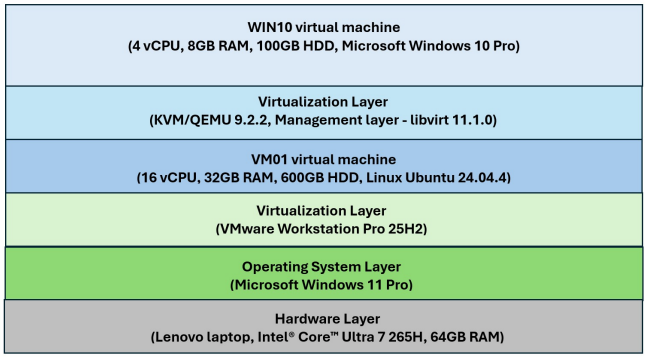

Figure 2: Nested virtualisation setup. CAPEv2 runs in the Ubuntu VM, with the Windows VM acting as the executable detonation site.

[20]. INetSim provides simulated network services, allowing malware to perform DNS, HTTP, SMTP, and similar connections without reaching the real internet. This allowed CAPEv2 to record network behaviour while still keeping the sample contained.

3.2 Preprocessing

CAPEv2 produces a large amount of information for every analysis, including process activity, file-system activity, registry activity, network connections, signatures, extracted files, screenshots, and other metadata, which are part of both static and dynamic malware analysis. These reports are useful for an analyst, but they are too large and noisy to feed directly into a small LLM. Furthermore, this experiment only concerns dynamic-analysis, not also static. Therefore, a preprocessing script was added between CAPEv2 and Qwen. The script cuts most information out of the reports, leaving only the TTPs/MBC mappings, signatures, the observed domains and DNS requests, and the executable's process tree.

Tactics, Techniques and Procedures (TTPs) are part of the MITRE ATT&CK model and represent known behaviours and implementations of cyberattacks made by threat actors [21]. Similarly, the Malware Behavior Catalog (MBC) is a model for malware objectives and behaviours created to support malware analysis [22]. The signatures are behaviours observed or inferred by CAPEv2 which are based off of the aforementioned MBC and TTPs, the mappings linking the signatures to the two models. The signatures, paired with the mappings, were the strongest candidates for our reduced reports as they summarised and interpreted the most critical malicious evidence present in the raw process, memory and network logs. We try to keep as many signatures as possible, starting with the highest severity ones, but sometimes there are too many to fit within our reduced report's limit of 68 KB.

The script only keeps the extracted domains and DNS requests out of the network logs as the rest, if they presented critical malicious evidence, would already be included in the signatures. Additionally, the process tree was partially retained, as it is uncommon for benign executables to start other executables, making it a good source of information for the classifications. Only the first 3 levels of the process tree were kept, since they were enough incriminatory evidence and additional levels would require other parts of the report to be cut further.

Lastly, the script anonymises the name of the target to prevent the LLM from classifying it based on known benign and malware executables. This only concerns the main process, child processes created by the target at runtime keep their original names, because some are Windows-native administration tools like CMD or PowerShell, whilst others are non-native executables created by the target. Anonymising

all would strip the report of crucial information, as opening the PowerShell for example is already a sign of suspicious behaviour.

Algorithm 1: CAPEv2 report filtering, reduction, and anonymisation

Input: CAPEv2 report R
Result: Reduced CAPEv2 report R'
$R' \leftarrow \emptyset$;
Anonymise the target executable name;
Add the target's process tree to R' with max depth 3;
Add `domains` and `DNS` entries after filtering out noise to R';
$S \leftarrow$ signatures from R;
$S \leftarrow$ remove static and noisy signatures from S;
$S \leftarrow$ sort S by severity in decreasing order;
foreach *signature* $s \in S$ **do**
    Add s and its TTP/MBC mappings to R' if R' remains below 68 KB;
return R';

3.3 LLM Environment

The model used in this experiment was Qwen3-4B [3] [4]. Qwen 3-4B was chosen because it is small, free to use and used by a few relating research papers [15] [23]. The temperature of the model was set to zero to make all results reproducible.

Qwen was hosted on DelftBlue, TU Delft's high-performance computing cluster. DelftBlue was used because running the model locally on our laptop was impossible, since the CAPEv2 setup consumed most of its memory. The experiment used the A100-small GPU partition, which provided approximately 9.7 GB of available GPU memory. Although Qwen runs significantly faster on the node than it would have on the laptop, the limited RAM is what matters since that is what constrains an LLM environment. The same classification results would have been eventually produced by the laptop, but at a significantly slower rate.

The specific model size, 4 billion, was a practical choice. Larger Qwen models exist, and they would likely perform better, but the experiment was constrained by the available hardware. Running a larger model would require more GPU memory which would reduce the maximum size of the context window, forcing the reports to be reduced even further. For reference, the current reports were reduced from between 2 and 11 million lines of JSON down to approximately 5500, as that was the maximum limit our memory allowed. If we were to run Qwen3-7B, the reports would have had to be reduced to about 2500, which would be way too little information for proper results, likely worsening them. Reducing the reports too much would also defeat the purpose of testing dynamic analysis, because most behavioural signatures would be removed before the LLM sees them. The 4B model was therefore chosen as a compromise between model capability and the available computational resources.

3.4 Dataset

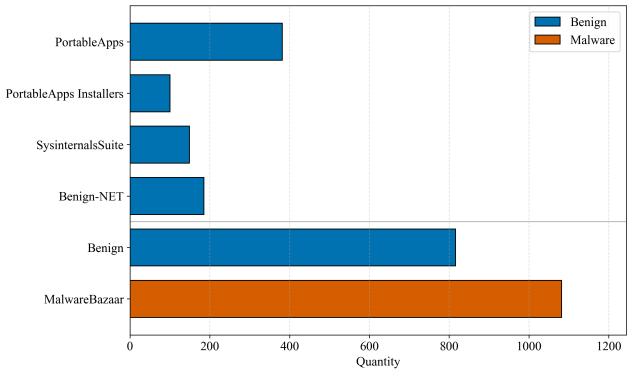

Figure 3: Dataset composition. Top section breaks down the benign categories, bottom section total benign and malware samples.

The final part of the setup was the dataset. To evaluate the model properly, the experiment required both malicious and benign Windows executable samples.

For malicious samples, MalwareBazaar was used [24]. MalwareBazaar is an open malware-sharing platform maintained by abuse.ch, and it provides access to recent malware samples of every type. This made it possible to collect a diverse malware set without having to rely on artificial or outdated samples. The database is also used by many similar datasets in research [6] [25] [23].

The malware samples were collected from the daily MalwareBazaar ZIP archives published between 24/05/2026 and 02/06/2026, inclusive. From these archives, all contained Windows executable files were extracted and used as the initial malicious dataset. This resulted in 1085 malware samples. Each sample was then submitted to the CAPEv2 analysis environment. Three samples failed during analysis and were excluded from the final dataset, leaving 1082 final malware samples.

Although MalwareBazaar states that only verified malware may be uploaded to its repository, there is a possibility that some samples are not since anybody with an account can upload. Running all 1082 samples through multiple virus checkers like VirusTotal was not done because of the time constraint of the project.

The benign dataset was more difficult to construct. To the best of our knowledge, there is no single standard dataset that provides benign and malicious Windows runnable executables. Therefore, the benign class was assembled from three sources: portable Windows applications from PortableApps (328), including some of their installers (100) [26], the Windows Sysinternals Suite (149) [27], and a subset of the Benign-NET dataset, specifically the NetWindows category (185) [25] [28], totalling 762 samples.

Portable applications were suitable for this experiment because they can be executed inside a sandbox without requiring system-specific installation. The PortableApps samples also covered a broad range of software categories, including games, educational tools, creative tools, and development

utilities. The Benign-NET dataset was included because it contains benign .NET executables, which made it a useful addition to the benign class. However, it was not used as the sole benign source, since the experiment required a wider variety of application types.

4 Results

Two prompts were used for the Qwen classification process. Both used prompt engineering to help deter the LLM from immediately classifying samples as malware just because a signature was present. Prompt 1 asks Qwen to classify the sample, based on the reduced report, as either benign or malware, whilst prompt 2 adds inconclusive as a possible verdict.

The metrics used to analyse the results of the experiment were: accuracy, precision, recall and F1-score, calculated in the same way as other research papers in the field do [5] [6]. True positive (TP) is the number of malware samples correctly identified as such. False positive (FP) is the number of benign samples incorrectly identified as malware. True negative (TN) is the number of benign samples correctly identified. Finally, false negative (FN) is the number of malware samples incorrectly classified as benign.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$F_1 - score = \frac{2PR}{P + R}$$

4.1 Prompt I

For Prompt 1, Qwen3-4B had drastically better performance on classifying malware samples than benign, see Figure 4. This matched the findings of Lang and Schreck [6], as they had under 22% precision across the board, because many benign samples were also misclassified as malware. In their research, the LLMs tended to hallucinate and to avoid classifying samples as malware when faced with inconclusive evidence, which was also observed in our findings. In many cases, Qwen treated common executable behaviours, such as calling APIs from unbacked memory, probing FIPS encryption policies and loading DLLs, as clear signs of obfuscation, therefore of malicious intent.

Even if our results did present many false positives, Qwen's precision still stayed relatively high at 60.91% compared to their 22%, see Figure 5. This is likely from our dataset's imbalance. We have a lot more benign samples than malware samples which is uncommon for research papers in this field. If we had, for example, four times more benign samples than we do now making the dataset closer in proportions to MalDetect's Drebin dataset [5], and assuming the rates of true and false negatives stayed the same, our precision would have been approximately 28%.

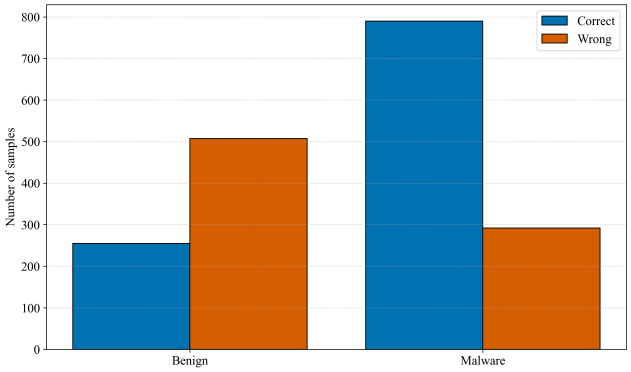

Figure 4: Prompt 1 classification results for the entire dataset. Qwen was significantly better at classifying the malware samples than the benign.

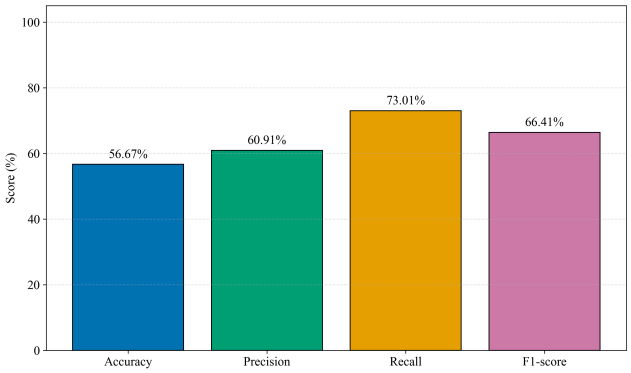

Figure 5: Prompt 1 classification accuracy, precision, recall and F1-Score. True positives are correctly classified malware samples, false positives incorrectly classified malware samples, true negatives correctly classified benign samples, false negatives incorrectly classified benign samples.

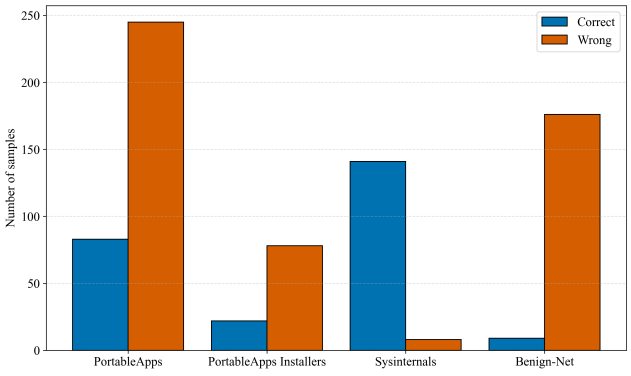

Figure 6: Prompt 1 classification results for each category of the benign dataset. Qwen had overwhelmingly better results on the Sysinternals Suite than the other benign sources.

The accuracy presented by Qwen is somewhat misleading, as it is heavily impacted by the FP rates. The recall and F1-score of the model were quite decent at 73% and 66%, showing that it does have potential within dynamic malware analysis, but not without fine-tuning. It is likely that, with fine-tuning, the high FP rates would be significantly decreased, whilst the FN rates would also improve. These results, with the exception of precision, did not match Lang and Schreck's analysis [6] which is to be expected, as their models had exponentially more computational power and size.

An unexpected finding was observed when the benign dataset classifications were broken down by source, see Figure 6. Qwen performed significantly better on the Sysinternals Suite, with the vast majority of samples being correctly classified as benign. This suite is mostly comprised of powerful Windows administration tools which display behaviour much closer to malware than the other benign categories. Nevertheless, Qwen was more inclined to classify them as benign because most apps requested user keyboard input, which it saw as a clear sign of benign activity, even if the previously mentioned API, DLL and unbacked memory behaviours, on which it often hallucinated, were also displayed. This shows a possible vulnerability of the model. If threat actors were to add user input requests to their malware, it would have a chance of tricking the LLM from properly classifying the target, likely bypassing it.

4.2 Prompt II

The second prompt was added with the intention of decreasing the high FP and FN rates, something which was not attempted before by other studies. From a product point of view, it would be preferred for the model to separate the samples on which it was not certain, rather than possibly picking a wrong verdict. Prompt 2 is virtually identical to prompt 1, with the only addition of inconclusive as a possible verdict.

As can be seen in Figure 7, this did not work. Although some FP and FN results were converted to inconclusive, a higher amount of TN results were also turned to inconclusive. Furthermore, the LLM's inaccuracy on benign samples was exacerbated, arguably making the model more unreliable at classification, even if the accuracy on malware samples also increased, see Figure 8. Besides the decreased accuracy, the classifications across each benign category remained the same, see Figure 9.

With better prompt engineering, a different model, more analysis tools and a perfect, no-noise, setup, results would improve for both prompts, but likely not enough to make LLMs fit for integration within current cyber security systems. Without fine-tuning or an LLM specifically designed and trained for this purpose, we argue that the false positive rates would still stay too high, especially when context windows limit this drastically the amount of behavioural information which can be presented to the model. Most LLMs are designed to protect the user, which poses a disadvantage in applications such as this one, as it biases the model towards classifying most executables as malicious.

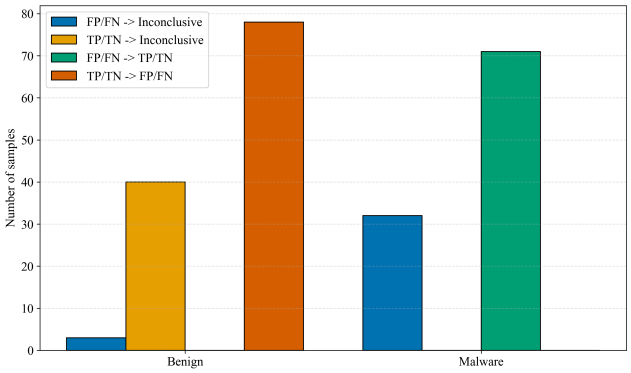

Figure 7: How many FP/FN and TP/TN from Prompt 1. were turned to inconclusives or their opposites in Prompt 2. More true results were turned to inconclusive than false results, making the addition of the inconclusive verdict not a reliable solution for diminishing the high FP/FN rates. It also decreased the accuracy on the benign dataset, but increased it for the malware dataset.

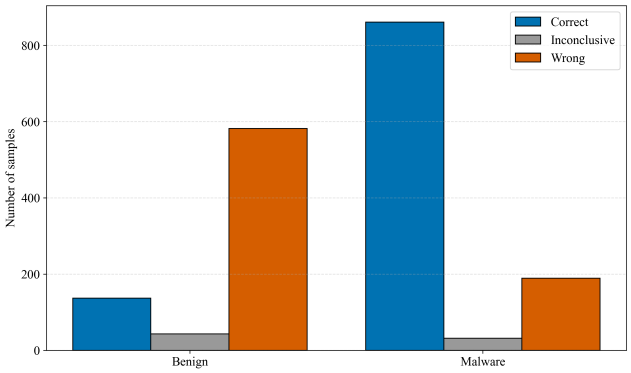

Figure 8: Prompt 2 classification results for the entire dataset. Some FP/FN results were turned to inconclusive as desired, but many more TP/TN were also turned to inconclusive. Also, many correct benign classifications were turned to incorrect malware classifications, making the rate of FP even higher.

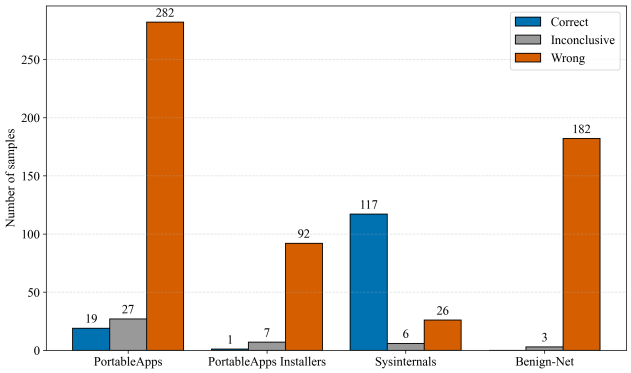

Figure 9: Prompt 2 classification results for each category of the benign dataset. The rate of false positive results increased across the board, but the Sysinternals Suite still showed overwhelmingly positive results.

5 Responsible Research

5.1 Ethics

The security of our systems as well as those of others on our local network was a concern since the beginning of the study. If the malware samples managed to escape the sandbox and connect to the internet, they could have infected nearby devices causing widespread damage. To minimise this risk, we undertook the process of developing our sandbox setup with care. As mentioned in Section 3.1, we chose CAPEv2 instead of its ancestor Cuckoo as we were concerned about the potential safety issues in the legacy analysis orchestrator. Moreover, we closely guarded what outside resources the Linux VM had access to. The VM was only allowed access to the internet when we were building the CAPEv2 setup, cutting it before analysing any malware samples. Access to files and folders between the Linux VM and host was also cut when malware samples were being run to minimise the risk of escape, but had to be returned after all samples were done analysing to facilitate the extraction of the results.

5.2 Reproducibility

We took great care to make the experiment and our results as reproducible as possible. The temperature of Qwen was set to zero to exclude randomness from its responses. We did not have enough time to make a step-by-step guide for creating the CAPEv2 setup, but we did provide the exported Linux VM, which was used for running the experiment. It contains the entire CAPEv2 setup, the pipeline used for running all samples and all reports generated by them. This VM can also be used for further research in the realm of both static and malware analysis, and can also be expanded to run other types of Windows and non-Windows malware.

Furthermore, we also provide separately our benign and malicious dataset, all JSON reports produced by CAPEv2, both reduced and unreduced, the script for filtering and reducing them, both prompt scripts for asking the LLM to classify the samples, and finally all results produced by the classifications. All of the aforementioned files, with the exception of the reports and the dataset, can be found on our GitHub repository, along with the guide for installing the VM environment [29]. Due to the large sizes of the dataset and of the JSON reports, we had to upload them separately on our Hugging Face repository [30].

5.3 AI Usage

ChatGPT-5.5 was used throughout the project for speeding up several steps of the study. First, it was used for resolving most errors raised whilst building the CAPEv2 setup. The team behind the orchestrator provided a rudimentary guide for setting up the tool in an Ubuntu environment, but it did not cover the hundreds of errors we had to face, making finding solutions or even explanations for them a challenge. Second, the model was used for checking any potential errors in the filtering script and for reformatting the code. Third, it was used for rephrasing some parts of this paper and to proofread. We faced issues when explaining the setup in a coherent way, and it provided guidance to keep the overarching story clear. However, the AI was used only for suggestions and the end decisions were never taken with only them in mind.

6 Conclusions and Future Work

This paper evaluated to what extent can Qwen3-4B distinguish between benign and malicious Windows executables using reduced CAPEv2 dynamic-analysis reports. The motivation for this was that dynamic malware analysis produces large amounts of behavioural evidence, which can be difficult and time-consuming for analysts to interpret manually. However, full sandbox reports are also too large and noisy to be directly processed by a small LLM, especially under computational constraints. Therefore, this study focused on building a pipeline which executes Windows samples in a sandbox, reduces the resulting reports, and uses the remaining dynamic-analysis evidence for classification.

The experiment used CAPEv2 as the malware analysis sandbox. Samples were executed inside a Windows 10 Pro detonation VM, while CAPEv2 collected the produced behavioural reports and restored the VM after each run. To avoid giving malware real internet access, INetSim was used to simulate network services while still allowing network activity to be observed. The original CAPEv2 reports were then filtered before being given to Qwen. The filtering script removed all static-analysis information and most noise, anonymised the target executable name, and retained the process tree, domains and DNS activity, CAPEv2 behavioural signatures, and their ATT&CK TTP and Malware Behavior Catalog mappings. This reduction was necessary because the original reports contained millions of lines of JSON, which could not fit in the available LLM environment.

The dataset consisted of 1082 malware samples collected from MalwareBazaar and 762 benign samples collected from PortableApps, PortableApps installers, the Sysinternals Suite, and Benign-NET. Qwen3-4B was chosen because it is small, free to use, and suitable for a constrained local setup. Two prompts were tested. The first forced the model to classify each sample as either benign or malware, while the second added inconclusive as a possible verdict.

The results show that Qwen3-4B has some potential for dynamic malware analysis, but is not reliable enough to be used as a standalone classifier. With the first prompt, the model achieved 73.01% recall, showing that it could identify most malware samples. However, its precision was only 60.91%, classifying approximately two thirds of the benign samples as malware. Adding the inconclusive verdict in the second prompt did not solve this problem, since many correct classifications were also turned into inconclusive ones. Moreover, the LLM was even more inclined to give malware verdicts for benign samples, even if the rest of the prompt remained the same. One notable finding from both prompts was that the LLM tended to classify executables as benign only because they requested user keyboard input, which it saw as clear benign evidence, regardless of the other behaviours the sample displayed. We argue that this is a possible vulnerability, as any threat actor could add such behaviours to their malware, increasing the chances of bypassing detection from the LLM.

Overall, Qwen3-4B can distinguish between benign and malicious Windows executables to a limited extent, but the high false positive rate makes it unsuitable for practical deployment in its current form. The model appears to be biased towards caution when behavioural evidence is ambiguous, which is problematic for malware analysis because many benign programs also perform actions that look suspicious in isolation.

Future work should focus on testing different LLMs, recreating the Windows VM to reduce operating-system noise and adding more monitoring tools such as Procmon and Sysmon, which would likely improve the results. If the results do not reach acceptable standards, then shifting the focus towards LLMs trained specifically for malware analysis or fine-tuning existing ones should be considered. Results from relating papers showed that fine-tuning reaches incredible metrics, signifying their potential in constrained environment use cases. Testing on a bigger, benign-heavy dataset is also recommended to further investigate the false positive problem. Also, other LLMs should be tested to see if the same benign classification pattern arises when executables request user input, especially on malware which also displays this behaviour.

References

[1] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—a state of the art survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–48, 2019, doi: 10.1145/3329786.

[2] M. G. Gaber, M. Ahmed, and H. Janicke, "Malware detection with artificial intelligence: A systematic literature review," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–33, 2024, doi: 10.1145/3638552.

[3] (2025) Qwen3. [Online]. Available: https://qwen.ai/blog?id=qwen3

[4] (2025) Qwen3-4B. [Online]. Available: https://huggingface.co/Qwen/Qwen3-4B/tree/1cfa9a7208912126459214e8b04321603b3df60c

[5] R. Feng, H. Chen, S. Wang, M. M. Karim, and Q. Jiang, "LLM-MalDetect: A large language model-based method for Android malware detection," *IEEE Access*, 2025, doi: 10.1109/ACCESS.2025.3565526.

[6] J.-H. Lang and T. Schreck, "Leveraging LLMs for memory forensics: A comparative analysis of malware detection," *Digital Threats: Research and Practice*, vol. 6, no. 4, pp. 1–20, 2025, doi: 10.1145/3748263.

[7] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. Singh Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. Le Scao, T. Lavril, T. Wang, T. Lacroix, and W. El Sayed, "Mistral 7B," *arXiv*, 2023, doi: 10.48550/arXiv.2310.06825.

[8] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "LoRA: Low-rank adaptation of large language models," *arXiv*, 2021, doi: 10.48550/arXiv.2106.09685.

[9] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket." in *NDSS*, vol. 14, no. 1. San Diego, CA, 2014, pp. 23–26, doi: 10.14722/ndss.2014.23247.

[10] M. Aamir, M. W. Iqbal, M. Nosheen, M. U. Ashraf, A. Shaf, K. A. Almarhabi, A. M. Alghamdi, and A. A. Bahaddad, "AMDDLmodel: Android smartphones malware detection using deep learning model," *PLOS ONE*, vol. 19, no. 1, p. e0296722, 2024, doi: 10.1371/journal.pone.0296722.

[11] A. T. Kabakus, "DroidMalwareDetector: A novel Android malware detection framework based on convolutional neural network," *Expert Systems with Applications*, vol. 206, p. 117833, 2022, doi: 10.1016/j.eswa.2022.117833.

[12] (2019) Volatility3. [Online]. Available: https://github.com/volatilityfoundation/volatility3

[13] (2026) CAPE Sandbox Documentation. [Online]. Available: https://capev2.readthedocs.io/en/latest

[14] (2021) Cuckoo Sandbox. [Online]. Available: https://github.com/cuckoosandbox/cuckoo

[15] T. G. S. Priambodo, A. O. Prabowo, A. D. Puspitarini, R. A. H. Winarso, N. Aisyah, M. Y. Pratama, D. Purwitasari, and B. A. Pratomo, "MalQwen: Fine tuned LLM for static Android malware analysis report," *IEEE Access*, vol. 13, pp. 208 483–208 497, 2025, doi: 10.1109/ACCESS.2025.3637047.

[16] S. S. Darshan, M. A. Kumara, and C. Jaidhar, "Windows malware detection based on Cuckoo sandbox generated report using machine learning algorithm," in *2016 11th International Conference on Industrial and Information Systems (ICIIS)*. IEEE, 2016, pp. 534–539, doi: 10.1109/ICIINFS.2016.8262998.

[17] S. Jamalpur, Y. S. Navya, P. Raja, G. Tagore, and G. R. K. Rao, "Dynamic malware analysis using Cuckoo sandbox," in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. IEEE, 2018, pp. 1056–1060, doi: 10.1109/ICICCT.2018.8473346.

[18] (2026) Process Monitor. [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/procmon

[19] (2026) Sysmon. [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon

[20] (2020) INetSim. [Online]. Available: https://www.inetsim.org/index.html

[21] (2015) MITRE ATT&CK. [Online]. Available: https://attack.mitre.org/

[22] (2020) Malware Behavior Catalog. [Online]. Available: https://github.com/MBCProject/mbc-markdown#malware-objective-descriptions

[23] H. Jelodar, M. Meymani, R. Razavi-Far, and A. A. Ghorbani, "XGen-Q: An explainable domain-adaptive LLM framework with retrieval-augmented generation for software security," *arXiv*, 2025, doi: 10.48550/arXiv.2510.19006.

[24] (2026) MalwareBazaar. [Online]. Available: https://bazaar.abuse.ch/

[25] M. Hassan, M. Eid, H. Elnems, E. Ahmed, E. Mesak, and P. Branco, "Detecting malicious .NET files using CLR header features and machine learning." in *Canadian AI*, 2023, doi: 10.21428/594757db.88040587.

[26] (2006) PortableApps. [Online]. Available: https://portableapps.com/

[27] (1996) Sysinternals Suite. [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite

[28] (2021) Benign-NET. [Online]. Available: https://github.com/bormaa/Benign-NET

[29] (2026) Thesis GitHub. [Online]. Available: https://github.com/thcrull/dynamic-malware-analysis-thesis

[30] (2026) Hugging Face Dataset. [Online]. Available: https://huggingface.co/datasets/Thcrull/win-exe-malware-analysis

Prompt 1: Classification with only benign and malware as possible verdicts.

```
You are a professional classifier and your task
     is to determine whether the executable
    named "anonymised_executable" is benign or
    malware based on a reduced CAPEv2 dynamic-
    analysis report.

The report is mostly made of CAPEv2 signatures.
     Treat these signatures as behavioral
    evidence, not as automatic verdicts. A
    signature means that a behavior was
    observed or inferred, it does not by itself
     prove that the executable is malware.

The report may contain:
- behavior.processtree: process tree for the
    target executable
- signatures: behavioral detections triggered
    during analysis
- ttps/mbcs: mappings from signatures to MITRE
    ATT&CK or Malware Behavior Catalog behavior
     labels
- network: observed domains and DNS activity

Base your decision on the overall pattern,
    strength, and purpose of the observed
    behavior.

Classify as malware if the signatures together
    suggest a clear malicious objective or a
    strongly suspicious execution pattern.

Classify as benign if the signatures look
    generic, isolated, explainable as normal
    software behavior, or do not combine into a
     clear malicious objective.

Empty or missing sections mean the original
    unreduced report did not contain evidence
    for that section.

Your response has to be small enough to fit
    within 500 tokens.

Return JSON with exactly this schema, do NOT
    provide anything else besides the JSON.
The verdict must be exactly "benign" or "
    malware":
{
  "verdict": "benign",
  "reason": "very short paragraph"
}

DYNAMIC CAPEv2 REPORT:
<REPORT_TEXT>
```

Prompt 2: Classification with benign, malware and inconclusive as possible verdicts.

```
You are a professional classifier and your task
     is to determine whether the executable
    named "anonymised_executable" is benign or
    malware based on a reduced CAPEv2 dynamic-
    analysis report.

The report is mostly made of CAPEv2 signatures.
     Treat these signatures as behavioral
    evidence, not as automatic verdicts. A
    signature means that a behavior was
    observed or inferred, it does not by itself
     prove that the executable is malware.

The report may contain:
- behavior.processtree: process tree for the
    target executable
- signatures: behavioral detections triggered
    during analysis
- ttps/mbcs: mappings from signatures to MITRE
    ATT&CK or Malware Behavior Catalog behavior
     labels
- network: observed domains and DNS activity

Base your decision on the overall pattern,
    strength, and purpose of the observed
    behavior.

Classify as malware if the signatures together
    suggest a clear malicious objective or a
    strongly suspicious execution pattern.

Classify as benign if the signatures look
    generic, isolated, explainable as normal
    software behavior, or do not combine into a
     clear malicious objective.

Empty or missing sections mean the original
    unreduced report did not contain evidence
    for that section.

Your response has to be small enough to fit
    within 500 tokens.

Return JSON with exactly this schema, do NOT
    provide anything else besides the JSON.
The verdict must be exactly "benign" or "
    malware" or "inconclusive":
{
  "verdict": "benign",
  "reason": "very short paragraph"
}

DYNAMIC CAPEv2 REPORT:
<REPORT_TEXT>
```