International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

AI: Friend or Foe? An Experimental Evaluation of LLM-Assisted Malware Classification Under Adversarial Conditions

Syed Khalid Tipu Razvi[1], Md. Ateeq Ur Rahman[2], Subramanian K.M[3]

[1]PG Scholar, Department of Computer Science and Engineering,
Shadan College of Engineering and Technology, Hyderabad, Telangana, India - 500086
Email: syedkhalid13055@gmail.com
[2]Professor, Department of Computer Science and Engineering,
Shadan College of Engineering and Technology, Hyderabad, Telangana, India - 500086
Email: mail to ateeq@yahoo.com
[3]Professor, Department of Computer Science and Engineering,
Shadan College of Engineering and Technology, Hyderabad, Telangana, India - 500086
Email: kmsubbu.phd@gmail.com

ABSTRACT

The swift development of artificial intelligence has significantly changed the field of cybersecurity. Artificial Intelligence is now widely used for attacking and defensive purposes, from the generation of malware, automation of the attack development, creating new malware variants which may not match existing signatures, to malware detection, incident response, triaging, and log investigation. This creates a major challenge for the traditional detection techniques, especially when the malware is generated or altered using artificial intelligence.

Previous research papers have shown that Large Language Models (LLMs) can aid in determining the AI-generated malware. However, many of these studies were performed under a controlled environment where the malware samples were easily readable and were slightly modified. Such conditions do not fully represent the real-world scenarios, where the malicious code undergoes transformations such as obfuscation, encoding, and encryption before analysis. Therefore, this research focuses on assessing how Large Language Models respond to AI-generated malware samples under different transformation conditions.

In this study, various Large Language Models (LLMs) were examined from a purple-team perspective, where both the offensive sample preparation and the defensive side classification were evaluated within a safe and controlled environment. Malware-like and dual-use security payloads were tested using structured prompts for analysis. Unlike idealized testing conditions where the source code is directly readable, this study assesses the samples under various transformation states such as plain code, obfuscated code, Base-92 encoded code, and AES-encrypted code. The responses are analyzed using the qualitative and quantitative criteria. The findings reveal that models do provide useful security insights for malware analysis. However, their detection dependability varies greatly when the original code is transformed, hidden, or hard to interpret. The study highlights the need for responsible restrictions on malicious prompts usage while focusing on the importance of evaluating the AI-based malware detection system. This facilitates the practical evaluation of AI-assisted malware analysis in the field of cybersecurity.

Keywords: Artificial Intelligence, Large Language Models, AI-Generated Malware, Malware Detection, Obfuscation, Encoding, Encryption, Purple Teaming, Cybersecurity.

I. INTRODUCTION

Artificial Intelligence (AI) is increasingly used in the field of cybersecurity for purposes such as malware analysis, threat detection, vulnerability assessment, and incident response. Large Language Models (LLMs) help in understanding the code, explaining the behavior of the code and assist security analysts in identifying suspicious logic. However, the same AI capabilities can be misused to generate malicious code or dual-use code, which creates a new challenge for traditional malware detection systems. [1]

Traditional malware detection systems mainly depend on signature-based, heuristic-based, and behavior-based strategies. These strategies are useful for malware which are previously known threats, but they become less reliable when the payloads are new, modified, or hidden. AI-generated malware increases this difficulty because it can create new code variations which may not match the existing research papers. Previous research papers have shown that LLMs can help in detecting and explaining the behavior of AI-generated Malware, but many evaluations are performed under idealized conditions where the source code was readable. [1],[2]

In a practical attack scenario, malware is often obfuscated, encoded, or encrypted or transformed to hide its real malicious intent. Obfuscation can change the visibility of the code, but keeps the main malicious behavior, which makes the detection for traditional and AI-based systems more difficult [5]. Similarly, adversarial malware research shows that malicious samples can be modified to bypass the machine-learning-based detection systems.[6][7]. Therefore, testing LLM-based malware detection with plain source code does not fully represent realistic defensive criteria. The study follows a purple-team perspective, where a controlled offensive sample is used to improve defensive detection. The red-team side of the study represents a malware-like dual-use sample preparation, while the blue-team side emphasizes classification, reasoning, and detection analysis. The

International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

malware samples were prepared only for safe research purposes and not for real-world deployment and harmful use.

This research also evaluates how the various LLMs classify samples under multiple conditions, such as plain code, obfuscated code, base-92 encoded content and AES-encrypted content. The main objective of this research is to observe whether the LLM-based malware detection is able to provide useful security reasoning when the original code is hidden.

The major contributions of this study are as follows:

1. Evaluation of the LLM-based malware detection from a purple-team perspective.
2. Testing malware samples under various conditions, such as plain, obfuscated, encoded and encrypted.
3. Comparison between multiple LLMs using structured prompts.
4. Analyzing both classification outcomes and reasoning quality.
5. Highlights the limitations of relying only on LLM-based models for transformed malware.

II. LITERATURE REVIEW

Existing research shows that the increasing use of Artificial Intelligence (AI) and Large Language Models (LLMs) has become important in cybersecurity tasks such as malware analysis, security code review, triaging, and threat detection. Earlier malware detection systems relied heavily on predefined rules, signatures, and observed the behavior of the program. While these methods remain useful in the modern security processes, prior studies indicate that their effectiveness can decrease when malware is modified through obfuscation, encoding, or encryption [5].

Owen Slubowski's work, "Fixing What You Broke: Can AI Be Used to Thwart AI-Generated Malware?", serves as the main base study for this paper. That research compared AI-assisted malware analysis with legacy detection tools such as VirusTotal, MetaDefender, and Hybrid Analysis. The study revealed that LLMs can provide useful classification and explanation of the AI-generated malware[1]. However, the work was mainly tested on a limited number of readable samples, which leaves the scope for further evaluation under adversarial and transformed conditions.

The offensive side of AI-generated malware is supported by Nethercott, who demonstrated how the generative AI can be misused to bypass security controls and assist in the creation of a malware payload[2]. Researchers from CyberArk's Threat team also demonstrated methods which bypassed the AI security filters to generate the malicious malware payload and showed that AI-assisted code generation can also support a polymorphic malware behavior, which shows that different variants of similar malicious logic can complicate the detection[11]. These works support the red-team side of the study, where malware-like or dual-use samples are prepared only for controlled evaluation.

The LLMs have also been studied for security code review. Yu et al. examined the capability of the LLM to analyze the security-related code[3], while McQuade explored their use for defensive code review [4]. This work supports the use of structured prompts for analyzing questionable code. However, they focus mainly on code review, while the present study applies structured prompts on LLM-based malware detection on a malware-like sample classification under transformed conditions.

Obfuscated and adversarial malware research further supports the need for realistic malware testing. Hasan and Dhakal discussed the obfuscated malware detection in real-world scenarios[5]. Chen reviewed the black-box adversarial attacks against a machine learning-based malware detector [6]. While Li et al. proposed PAD to enhance the robustness against such evasion attacks[7]. These studies show that the detection system should be tested beyond readable code.

A recent study also revealed that LLMs can be used to generate malware variants and controlled malware situations. Akil et al. studied LLM-based malware variant generation[9]. while Saha and Shukla proposed MalGEN for modelling the malicious software in cybersecurity research[10]. These works further support the need for responsible and controlled evaluation of AI-generated malware-like samples.

Recent works in 2026 also support the importance of evaluation of LLMs in malware-related analysis. Chawla and Prasad proposed a decompilation-driven framework for malware detection using Large Language Models, in which executable code is firstly decompiled and then analyzed by LLMs for benign or malicious classification [12]. Their findings indicate that LLMs show promise for malware classification, but standard LLMs are not yet reliable enough to replace traditional antivirus systems. This supports the present study's view that LLMs should be treated as assistive tools rather than standalone malware detectors.

Similarly, Saul et al. proposed Trident, a malware detection approach that combines static features, behavior-based detection rules, and direct LLM analysis of sandbox reports [13]. This study shows that LLMs can contribute to malware detection workflows when combined with traditional security evidence. This is closely related to the future direction of the present research, where LLM-based reasoning can be integrated with sandbox execution, YARA rules, and other malware analysis methods for stronger validation.

These recent works show that LLM-assisted malware analysis is still an active and developing research area. However, while some recent studies focus on decompiled executable code, sandbox behavior, or automated detection frameworks, the present study focuses on comparing public LLM responses under structured prompts and transformation-based conditions such as plain code, identifier-renamed obfuscation, Base92 encoding, AES encryption, and benign encrypted control testing.

Overall, the literature review shows that AI-based malware detection is promising, but certain gaps remain in transformed-sample assessment. Existing studies often focus

International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

on readable source code, final classification results, decompiled executables, or automated detection frameworks. This study addresses the gap by assessing LLM responses under plain, obfuscated, Base92-encoded, and AES-encrypted sample conditions while also inspecting the quality of model reasoning.

The major related works considered in this study are summarized in Table 1.

Table 1: Summary of Related Work

S.no	Research Paper	Year	Relevance	Gap
1	Fixing What You Broke: Can AI Be Used to Thwart AI-Generated Malware? [1]	2025	Main base paper for AI-generated malware detection using LLMs.	Limited readable samples and limited transformation testing.
2	The Evolution of the Digital Predator: Using AI to Evade Security Controls [2]	2023	Supports red-team AI misuse and malware generation background.	Focuses on offensive misuse, not defensive transformed-sample detection.
3	An Insight into Security Code Review with LLMs [3]	2024	Supports structured prompting and LLM-based security code analysis.	Focuses on security code review, not malware classification.
4	Leveraging LLMs for Security-Focused Code Reviews [4]	2025	Supports LLM use for defensive code review.	Does not evaluate transformed AI-generated malware.
5	Obfuscated Malware Detection: Investigating Real-World Scenarios [5]	2024	Supports obfuscation and transformation testing.	Does not deeply evaluate LLM reasoning on encoded/encrypted samples.
6	A Review of Black-Box Adversarial Attacks and Defenses in ML-Based Malware Detection [6]	2024	Supports adversarial evasion testing.	Review-based; does not test LLMs on transformed samples.
7	PAD: Towards Principled Adversarial Malware Detection [7]	2023	Supports robustness against evasion attacks.	Focuses on ML-based detection rather than LLM reasoning.
8	Application of Deep Learning in Malware Detection: A Review [8]	2025	Gives broad malware detection background.	General review, not specific to LLM-based detection.
9	LLMalMorph [9]	2025	Supports LLM-based malware variant generation.	Focuses on generation, not comparative LLM detection.
10	MalGEN [10]	2025	Supports controlled malware simulation for research.	Does not focus on prompt-wise LLM detection evaluation.
11	Chatting Our Way Into Creating a Polymorphic Malware [11]	2023	Supports AI-assisted polymorphic malware and evasion risk.	Industry threat research, not a peer-reviewed academic paper.
12	A Decompilation-Driven Framework for Malware Detection with Large Language Models [12]	2026	Supports LLM-based benign/malicious code classification	Focuses on decompiled executables rather than prompt-wise transformed sample testing
13	Trident: Improving Malware Detection with LLMs and Behavioral Features [13]	2026	Supports combining LLMs with behavioral malware-analysis evidence	Focuses on detection pipeline integration rather than comparing public LLM responses

III. METHODOLOGY

The study follows an experimental purple-team research design to understand how the Large-Language-Models (LLMs) classify and give reasons for malware-like and dual-use samples under various transformation conditions. The objective of the research paper is not to build a new malware detection system, but to examine how existing LLMs respond when suspicious code or transformed code is given in plain, obfuscated, encoded, and encrypted forms.

A. Research Design

The study combines both red and blue team perspectives. The red-team side involves developing a controlled malware-like and dual-use sample which represents the suspicious behavior. The blue-team side involves submitting the samples to the Large Language Models(LLMs) and analyzing the responses for quantitative and qualitative reasoning.

International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

The Purple Team approach was selected because it allows a controlled offensive simulation, which can be used for defensive purposes. The purpose of this study is not to create a new malware detection system but to examine how existing Large Language Models (LLMs) act when given a malware-like or dual-use sample under various transformations.

All malware-like and dual-use samples used in the research paper were prepared only for controlled academic and defensive security research. The samples were not intended for real-world deployment, personal use, or harmful activity. The samples should not be executed against the real systems and should not be reused outside an authorized research environment.

B. Sample Preparation.

The sample set involves two groups. The first group contains payload categories from Owen Slubowski's paper, in which the initial malware samples were generated using the morality bypass technique, as discussed in Foster Nethercott's work [1],[2]. These samples represented common malicious or dual-use tools like keylogger, covert communication, port scanner and reverse shell activity.

The second group consists of 4 additional malware-like and dual-use samples prepared with LLM-assistance from a purple team perspective. The decision to include these additional AI-assisted variant-style samples was motivated by CyberArk's threat research on a polymorphic malware, which revealed that AI-generated samples can make the detection complicated [11]. The samples were included in order to expand the test cases and also to observe how different LLMs respond to a wider set of suspicious behavior.

The purpose of preparing the additional samples was not to create deployable samples, but to support the LLM-based malware classification. As discussed in Section A, all samples were solely used within the research paper to evaluate the model behavior under multiple transformation conditions.

C. Transformation Conditions.

Each sample was tested under multiple transformation conditions in order to understand how the readability affects LLM based malware classification. The conditions used in this study were plain code, obfuscated code, Base-92 encoded content and AES-encrypted content.

In the plain code condition, the sample was provided to the LLM in readable source code form. In the obfuscation condition, the sample was modified using techniques like identifier renaming while retaining the original logic. In the Base-92 encoded condition, the obfuscated sample was encoded to make the source code harder to read directly, and in the AES-encrypted condition, the sample was encrypted so that the original code could not be directly inspected by the model.

These transformation conditions were used to represent different levels of visibility, which range from fully readable source code to hiding the underlying logic. This helped understand whether the LLMs rely on direct code understanding or respond mainly to surface-level indicators, such as unreadable content, encoding patterns, or high-entropy encrypted content.

D. Prompting Strategy

The Large Language Models (LLMs) were tested using the three structured prompts adapted from the previous AI-assisted malware analysis and security code review research [1],[3]. The purpose of using the structured prompts was to keep the testing conditions consistent across different models, samples, and transformation conditions.

Prompt 1 involves a basic analysis prompt that asks the model to describe the function of the given program and classify it as malicious or non-malicious. Prompt 2 adds a security analyst role to the prompt and asks the model to analyze the sample from the defensive security perspective. Prompt 3 added the cybersecurity context by asking the model to analyze the sample and use frameworks such as MITRE ATT&CK and Cyber Kill Chain to classify the malware.

The use of structured prompt styles helped observe whether additional security context improved the model's classification and reasoning quality. Each prompt was applied consistently during the testing stages, and the responses were recorded without a follow-up interaction.

E. LLM-Based Testing Process.

Each sample was submitted to the selected LLMs under the defined transformation conditions. The models which were tested in this study were ChatGPT, Gemini, Perplexity, DeepSeek, and BlackBox. The same prompt structure was used across all the models to ensure consistency during comparisons.

The testing was performed using a one-shot approach, where the response was recorded after the initial prompt without any follow-up questions. This helped reduce variation caused by repeated questioning for clarification. The responses were then grouped according to the testing stage, model, prompt, and transformation condition.

The testing stages included various transformations conditions as stated in Section C. The collected responses were analyzed to understand how each LLM behaved when the same or similar malicious logic was presented in different levels of readability.

F. Evaluation Criteria

The LLM responses were evaluated using the qualitative and quantitative criteria. The quantitative evaluation focused on the final classification given by each model, while the qualitative evaluation focused on the reasoning and explanation provided for the response.

For quantitative evaluation, each response was assigned a color based classification label. Where green represents direct malicious classification, yellow represents suspicious or dual-use recognition without direct malicious classification. Red represents a non-malicious classification, orange/grey represents refusal, unclear output, or inability to analyze the sample. Because the experiment included multiple payloads, models, prompts and transformation conditions, the complete result matrix was too large to include in the main paper. Therefore, the results are

International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

summarized using aggregate tables that show the behavior of the model across different testing stages.

For qualitative evaluation, the reasoning provided by each model was examined. This includes inspecting how the model explained the sample behavior, identifying indicators of compromise (IOC), recognized dual-use functionality, or misclassified transformed content. Special attention was given to the encoded and encrypted samples, where the model may react to unreadable or high-entropy content without actually understanding the original logic.

G.Ethical and Safety Considerations.

Since this study involves malware-like and dual-use samples, ethical and safety controls were followed throughout the research. The samples used are only for academic and defensive analysis purposes. The samples were not deployed, executed against real systems, or shared for harmful purposes.

No instructions, source code, or operational steps intended to enable real-world misuse are included in the paper. The study focuses only on classification behavior, reasoning quality, and the effect of transformation techniques on LLM-based malware analysis.

IV. EXPERIMENTAL SETUP

The experiment was arranged as a structured matrix to compare LLM responses across models, prompts, and transformation conditions. Five LLM-based tools were evaluated, where each model was tested with the same payload groups and the same prompt conditions. This was done to ensure consistency across every model.

A. Models and Test Environment

The selected models were tested through the available web interfaces. The responses were collected manually and recorded in a structured table. No follow-up questions were asked after the first response, so that each result represented the model's first classification and explanation.

The Following models, along with the versions, are evaluated in this study:

- ChatGPT-5.3 (Free)
- Google Gemini - Gemini Pro (Paid)
- Perplexity - Web version (Free)
- DeepSeek - V3 (Free)
- BlackBox 1.0 (Free)

B. Experimental Matrix

The test cases were grouped according to the payload condition and prompt type. Plain payloads and AES-encrypted payloads were tested using the three prompts, where the obfuscated and encoded payloads were tested only with prompt 2.

After the plain payload testing stage, four payloads were selected for the obfuscation and Base-92 encoding stages. These payloads were selected because they produced stronger malicious classification responses during the plain payload assessment.

The purpose of using this subset was to observe whether the samples that were more clearly detected in a readable form would still be detected after transformation. Prompt 2 was selected for the obfuscation and Base-92 encoded testing stages because it showed stronger overall performance during the plain payload testing stage and provided balance security analyst context.

Table 2 shows the overall experimental matrix used in the study.

Table 2: Experimental Testing Matrix

Prompt Condition	Cases	Prompt Type
Plain Payload – Prompt 1	16	Basic analysis prompt
Plain Payload – Prompt 2	16	Security analyst prompt
Plain Payload – Prompt 3	16	MITRE-informed prompt
Obfuscated Payload	4	Transformation-based testing
Base92 Encoded Payload	4	Encoded transformation testing
AES Encrypted – Prompt 1	16	Basic analysis prompt
AES Encrypted – Prompt 2	16	Security analyst prompt
AES Encrypted – Prompt 3	16	MITRE-informed prompt
Benign AES Control Test	1	MITRE-informed prompt

C.Transformation Tools

The transformation stages were used to test how LLMs respond when the original payload becomes less readable. Obfuscation was performed by modifying identifiers while preserving the original behavior of the sample. This allowed the experiment to compare the responses from the model between the readable and the modified version of the same logic.

CyberChef was used for Base-92 encoding and AES encryption logic. In the Base92-encoded condition, the previously obfuscated sample was encoded further, making the source code more difficult to the models to analyze. Base92 was selected instead of Base64 because preliminary testing it showed that the LLMs could often recognize and decode the Base64 content more easily. Base92 provided a less recognizable encoded condition, making it more suitable for observing how the models provide a response when the source code is not easily readable or recoverable.

In the AES-encrypted condition, the sample content was encrypted, preventing the model from directly inspecting the source code. This condition was included to observe how models classify or reason about the content when the underlying code behavior is hidden.

International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

Additionally, a benign AES-encrypted English story was also included as a control test to observe whether the models would classify an encrypted non-malicious file as malicious.

D.Result Recording

For every test case, the model's final classification and reasoning were recorded. The responses were grouped according to the model, prompt, and payload condition and the transformation stage. The final classification was later converted into color-based notation, which was defined in the methodology section.

The quantitative results were summarized by aggregating the table rather than presenting every individual test case. This approach was selected because the full result matrix was too large for the main paper in the two column journal format. The qualitative observations were then used to explain why certain models classified transformed samples as malicious, suspicious, benign or unclear.

V. RESULTS AND DISCUSSION

This section presents the quantitative and qualitative findings that were obtained from testing different LLMs across plain, obfuscated, Base92-encoded, and AES-encrypted payload conditions. The quantitative results are based on "Green" classification, where the model directly classified the samples as malicious. Since a classification label does not explain why the model came to the conclusion, qualitative reasoning was also analyzed to understand whether the response was based on actual behavior, suspicious indicators, dual-use functionality, or assumptions about transformed content.

The results are discussed across the four main testing stages: plain payload testing, obfuscated payload testing, Base92-encoded testing, and AES-encrypted testing. This helps compare how model behavior changes as the original payload's semantic visibility reduces or is completely hidden.

Table 3: Quantitative Summary of LLM Malware Classification Results

Testing stage	Cases Per Model	ChatGPT	Gemini	Perplexity	DeepSeek	BlackBox
Plain Payload – Prompt 1	16	7/16	14/16	9/16	10/16	5/16
Plain Payload – Prompt 2	16	7/16	16/16	10/16	11/16	16/16
Plain Payload – Prompt 3	16	8/16	16/16	11/16	9/16	10/16
Obfuscated Payload	4	4/4	4/4	4/4	4/4	4/4
Base92 Encoded Payload	4	0/4	4/4	2/4	4/4	4/4
AES Encrypted – Prompt 1	16	5/16	0/16	16/16	0/16	0/16
AES Encrypted – Prompt 2	16	8/16	16/16	0/16	16/16	15/16
AES Encrypted – Prompt 3	16	16/16	16/16	0/16	13/16	16/16

ISSN 2250-3676 ijesatj@gmail.com https://www.ijesat.com/

International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

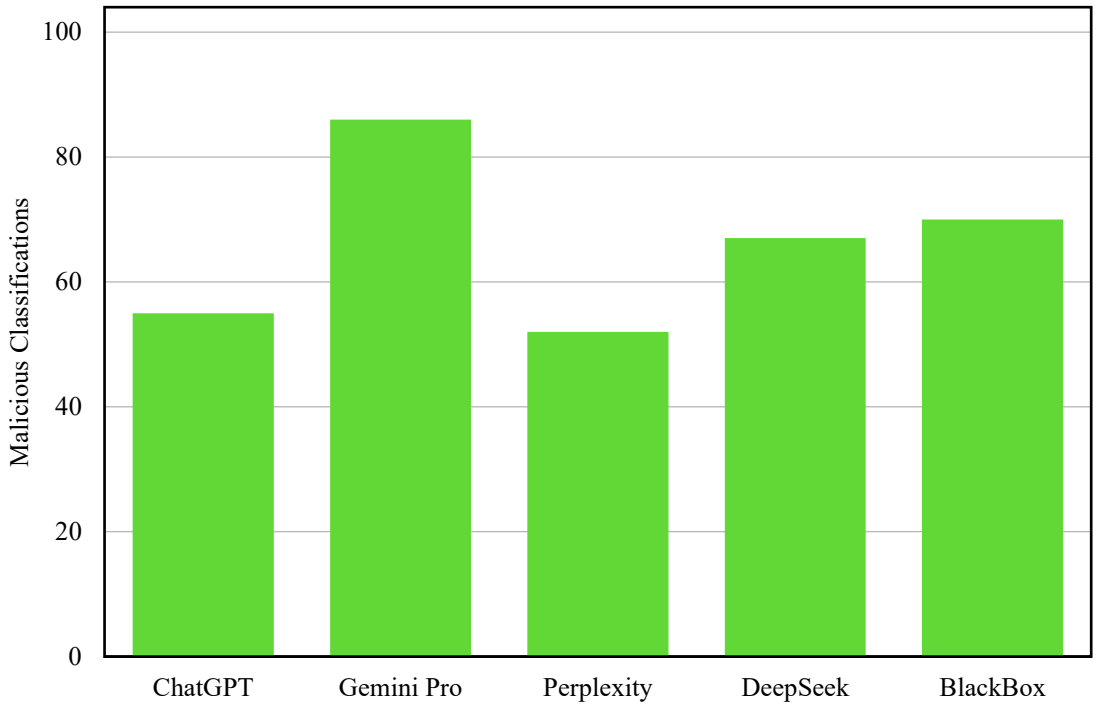

Figure 1: Total Direct Malicious Classifications by Each LLM Across All Testing Stages

A.Quantitative Results

Table 3 presents the aggregated quantitative results across all testing stages. The values represent the number of "Green" classifications, where the model directly classified the payload as malicious, out of the total cases tested in that stage.

As shown in Table 3, the models produced different classification patterns across the testing stages. Gemini showed consistently high malicious classification results in plain payload and AES-encrypted testing stage under Prompt 2 and Prompt 3. ChatGPT showed a cautious approach in plain payload and in Base92-encoded testing, but its malicious classification increased strongly in AES-encrypted Prompt 3. Perplexity showed more of a moderate output in the plain payload testing, but shows inconsistency during the encrypted testing. DeepSeek and BlackBox show strong classification results for obfuscated and Base92-encoded payloads, while their AES-encrypted results varied depending on prompt context.

The overall model-wise classification summary is shown in Figure 1. The chart represents the total number of malicious classifications across all testing stages.

B. Plain Payload Results.

In the plain payload testing results, the models were tested on a readable source code sample using three different prompts. From Table 3, Gemini produced the strongest direct malicious classification results, scoring 14/16 in Prompt 1 and 16/16 in Prompt 2 and Prompt 3. BlackBox showed a fewer malicious classifications in Prompt 1, but improved significantly with Prompt 2 to reach 16/16.

During the plain payload testing stage, ChatGPT exhibited a mindful approach behavior. Scores were 7/16 for Prompt 1 and Prompt 2 and 8/16 for Prompt 3. The lower number of malicious classifications does not necessarily mean that the model failed to analyze the samples. In several cases, the model recognized suspicious or dual-use sample behavior but did not always directly classify it as malicious. Perplexity and DeepSeek showed a moderate approach behavior, with results varying slightly depending on the prompt. In the plain payload testing out of all the prompts, Prompt 2 produced strong results for multiple models because it framed the task from a security analyst's perspective. This suggests that the role-based prompting can influence LLMs' classification behavior. However, Prompt 3 did not improve all the models equally, showing that additional cybersecurity framework context does not always guarantee higher malicious classification rates.

C. Obfuscated and Base92-Encoded Results

In the obfuscation payload stage, all five models classified all four samples as malicious, producing 4/4 results across ChatGPT,Gemini, Perplexity, DeepSeek, and BlackBox. This shows that with obfuscation techniques(Identifier Renaming) did not significantly reduce the LLM-based malicious classification when the general code logic remained visible.

The Base92 encoding stage produced more varied results. Models such as Gemini, DeepSeek, and BlackBox

International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

classified four encoded samples as malicious, while Perplexity classified 2/4. ChatGPT wasn't able to classify directly any Base92 encoded sample as malicious. One possible reason for this variation is that most of the models were not able to decode easily, like other encoding methods, such as Base64, which is widely used; Base92 is less recognized than Base64, which may have made direct decoding less likely. As a result, some models appeared to treat the encoded content itself as suspicious, while others avoided making a direct malicious classification because the underlying behavior could not be verified.

The results suggest that LLMs may still identify suspicious behavior when obfuscated code remains readable at the logic level. However, when the obfuscated samples are encoded, and the original source code is no longer directly readable, model behavior becomes less consistent. In such cases, some models may rely on suspicious patterns or an unreadable nature of the content rather than actual code behavior.

D.AES-Encrypted Results

The AES-Encrypted stage produced the most inconsistent model behavior. Since the encrypted content prevents direct inspection of the original source code, malicious classification in this stage does not mean that the model understood the actual payload behavior. Instead, the model may have been relying on indirect signals such as unreadable content, high-entropy structure, or the security context in the prompt.

Perplexity classified all 16 AES-encrypted samples in Prompt 1 as malicious. Gemini, DeepSeek and BlackBox did not provide a direct malicious classification in Prompt 1. ChatGPT classified 5 out of 16 samples as malicious. Under Prompt 2, Gemini, DeepSeek, and BlackBox showed a major increase in malicious classification, while ChatGPT improved to 8/16. Under Prompt 3, ChatGPT,Gemini, and BlackBox classified all samples as malicious, while DeepSeek classified 13/16, whereas Perplexity showed 0/16.These results suggest that prompt framing strongly affects LLM behavior when the underlying code is encrypted. However, AES-encrypted results must be interpreted carefully because the models could not perform direct inspection of the original source code. In some cases, the models appeared to interpret the encrypted content as hexadecimal-like data and attempted to reason from its structure. Such responses were often based on approximate assumptions rather than verified code behavior. Therefore, high malicious-classification rates in this stage may indicate suspicion towards the encrypted content rather than a confirmed understanding of the actual sample behavior.

E. Benign Encrypted Control Test

To understand further about the model's behavior on encrypted content. A benign control test was performed. In this test, a normal English story was AES-encrypted and submitted to the models using Prompt 3. The purpose of this test was to observe whether the models would classify encrypted content as malicious even if the original plaintext did not contain malware-like logic.

Table 4: Benign AES-Encrypted Control Test

LLM	Sample Type	Transformation Applied	Prompt Used	Result
ChatGPT	Benign English story	AES encryption only	Prompt 3	Classified as malicious
Gemini	Benign English story	AES encryption only	Prompt 3	Inconclusive / unable to determine
Perplexity	Benign English story	AES encryption only	Prompt 3	Classified as malicious
DeepSeek	Benign English story	AES encryption only	Prompt 3	Not malicious
BlackBox	Benign English story	AES encryption only	Prompt 3	Classified as malicious

From Table 4, three models classified the AES-encrypted benign story as malicious, while Gemini gave an inconclusive response, and DeepSeek classified it as non-malicious. The results suggest that some models may associate encrypted or high-entropy content with malicious behavior even when the original plaintext is harmless. Therefore, encrypted-content classification should be interpreted carefully, since a malicious label may reflect suspicion towards the transformation rather than a confirmed understanding of the hidden content.

F. Qualitative Observations

The qualitative analysis showed that the models differed not only in their final classification, but also in the quality of the explanations. During the plain payload and the obfuscated payload stages, the models generally provided proper reasoning since the source code logic was still visible. In these cases, the responses often identified the suspicious behavior such as keylogger, covert communication, scanning activity, or remote access functionality or any other malware samples in the research study.

International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 26 Issue 6 (June) 2026
Received: 10-5-2026 Accepted: 21-5-2026 Published: 07-6-2026

For Base92-encoded and AES-encrypted samples, the quality of reasoning becomes less consistent. Since the original code is unreadable, some models relied on indirect indicators, such as unreadable text and encoded patterns, high-entropy structure, or security context provided in the prompt. As a result, some classifications appeared to be based on suspicion rather than verifying the actual payload behavior.

The benign AES-encrypted control test further supported this observation. Several models classified an encrypted story as malicious. Even though the plain text was harmless. This suggests that encrypted or unreadable content may cause some of the LLMs to produce suspicious or malicious classifications without confirming its underlying behavior.

Overall, the qualitative findings indicate that LLMs can be useful for explaining suspicious code when the code logic is visible or partially visible. However, when the samples are encoded or encrypted, the models provide results based on assumptions. Therefore, LLM-based malware analysis should be treated as an assistive technique rather than a standalone detection method.

VI. LIMITATIONS

This study has several limitations which should be considered while interpreting the results. First, this study

was designed as a focused experimental evaluation rather than a large-scale benchmark. The selected malware-like and dual-use samples were sufficient to compare LLM behavior across plain, obfuscated, encoded, and encrypted conditions. However, the result should be interpreted within the scope of the selected sample, prompts, models, and transformation methods used in this study.

Second, automation was explored, but it was not used to complete the final testing process. A Python-based automation prototype was created to read payload files, apply selected prompts, send requests to supported LLM APIs, collect responses, extract verdicts, assign classification labels, and store results in a structured manner. However, the final testing was performed manually through LLM web interfaces due to token costs, API rate limits, and model availability differences. This limitation is also consistent with Owen Slubowski's work, where an n8n-based workflow was proposed for AI-assisted malware analysis automation, but it still depended on the API rate limits and token costs[1].

Third, the study evaluated a limited number of LLM platforms. The tested models include ChatGPT, Gemini Pro, Perplexity, DeepSeek, and BlackBox. LLM platforms are frequently updated over time, and as a result, their responses may change over time due to changes to backend models, safety filters, reasoning capabilities, and platform policies. Therefore, the results represent the behavior of the tested models during the testing period, not a permanent evaluation of these platforms.

Fourth, this study used a limited set of programming languages and payload categories. Although the selected samples covered different suspicious behaviors, the evaluation can be expanded further by adding more programming languages and payload types.

Fifth, the study focused mainly on how the LLMs classify and explain malware-like samples. The LLMs did not perform independent malware-analysis operations such as sandbox execution, YARA rule matching, multi-engine scanning.

Finally, encoded and AES-encrypted samples limited the visibility of the original source code. Although the original behavior of the samples was known during the experiment, the LLMs could not directly inspect the behavior after encoding or encryption. As a result, responses in these stages may reflect how models react to hidden, unreadable, or encrypted content rather than how they interpret visible logic programs.

In general, such limitations indicate that the results should be interpreted in view of the specific models, prompts, payload types, programming languages, and transformation conditions chosen for this research.

VII. CONCLUSION AND FUTURE WORK

A. Conclusion

This research investigated the effectiveness of Large Language Models in analyzing and classifying malware-like or dual-use code samples across multiple transformation condition stages, which include obfuscation, Base92-encoding and AES encryption. The main finding is that LLM based malware classification is strongly affected by how much the original code logic remains visible to the model.

When the code was readable or partially readable, the models were generally able to identify suspicious behavior and provide useful explanations. However, when the same content was encoded or encrypted, the responses became less consistent. In such cases, some models appeared to rely on indirect indicators such as unreadable text, encrypted structure, or prompt context rather than a verified understanding of the underlying behavior. The benign AES-encrypted test also demonstrated that encrypted content alone could trigger malicious classifications, even though the content was harmless.

From the result, AI cannot be classified as a friend or enemy alone. Used correctly for defensive malware analysis, security reasoning, and classification support, it can be a friend. Used in the creation of malware-like code, hiding malicious intent, or trusting the outputs without validation, it can be an enemy. This study finds that AI is a friend when used in a controlled, ethical manner, and with human supervision for defensive purposes, but an enemy if misused or over-trusted, without the traditional security validation.

These findings suggest that LLMs can support malware analysis, but it should be treated as an assistive reasoning tool rather than a complete standalone malware detection system. The model's strength lies in explaining visible code behavior, identifying suspicious indicators, and recognizing malicious logic even after simple identifier renaming. Their weakness appears when the original logic is hidden, where the response may become assumption-based. Therefore, LLM outputs should be

reviewed carefully and, where possible, supported by the traditional security analysis methods.

B. Future Work

Future work can expand the study by testing larger and more diverse datasets, which can contain more malicious, benign, dual-use, obfuscated, encoded and encrypted samples. Additional programming languages such as PowerShell, JavaScript, C#, Go, Rust, BatchScripting, VBScript, can also be included to observe whether LLM behavior changes across different languages and code structures.

Future research can also evaluate more LLM platforms, including Claude, Mistral, Llama-based models, and other open-source or commercial systems. Comparing free and paid or pro versions of the same platforms may help determine whether an advanced model may provide better reasoning quality, fewer false positives, and stronger handling of transformed samples.

Another important direction is automation. Although a Python-based automation prototype was explored in this study, the final testing was performed manually due to token costs and API rate limits. Future work can develop a scalable automation framework using Python and APIs, or workflow tools to submit prompts, collect responses, classify outputs, and generate result tables automatically.

Future studies can also integrate LLM-based reasoning with traditional malware analysis tools, such as sandbox environments and implement YARA rules. Combining LLM explanations with evidence, static and dynamic analysis may produce a more reliable and practical framework for malware classification.

Finally, future LLM updates should improve prompt-level safety checks to identify adversarial prompt abuse, such as prompt injection, token smuggling, jailbreak attempts and disguised requests for harmful code generation. These restrictions should be balanced with support for legitimate defensive use cases such as malware analysis, threat explanation, and cybersecurity education.

REFERENCES

[1] O. Slubowski, “Fixing What You Broke: Can AI Be Used to Thwart AI-Generated Malware?” SANS Institute, 2025.

[2] F. Nethercott, “The Evolution of the Digital Predator: Using AI to Evade Security Controls,” SANS Institute, 2023.

[3] J. Yu, P. Liang, Y. Fu, A. Tahir, M. Shahin, C. Wang, and Y. Cai, “An Insight into Security Code Review with LLMs: Capabilities, Obstacles and Influential Factors,” arXiv preprint arXiv:2401.16310, 2024.

[4] D. McQuade, “Leveraging Large Language Models for Security-Focused Code Reviews,” SANS Institute, 2025.

[5] S. M. R. Hasan and A. Dhakal, “Obfuscated Malware Detection: Investigating Real-World Scenarios through Memory Analysis,” arXiv preprint arXiv:2404.02372, 2024.

[6] J. Chen, “A Review of Black-Box Adversarial Attacks and Defenses in Machine Learning-Based Malware Detection,” Applied and Computational Engineering, 2024.

[7] D. Li, S. Cui, Y. Li, J. Xu, F. Xiao, and S. Xu, “PAD: Towards Principled Adversarial Malware Detection Against Evasion Attacks,” arXiv preprint arXiv:2302.11328, 2023.

[8] Y. Song, D. Zhang, J. Wang, Y. Wang, Y. Wang, and P. Ding, “Application of Deep Learning in Malware Detection: A Review,” Journal of Big Data, vol. 12, article 99, 2025, doi: 10.1186/s40537-025-01157-y.

[9] M. A. Akil, A. S. Li, I. Karim, A. Iyengar, A. Kundu, V. Parla, and E. Bertino, “LLMalMorph: On the Feasibility of Generating Variant Malware Using Large-Language-Models,” arXiv preprint arXiv:2507.09411, 2025.

[10] B. Saha and S. K. Shukla, “MalGEN: A Generative Agent Framework for Modeling Malicious Software in Cybersecurity,” arXiv preprint arXiv:2506.07586, 2025.

[11] E. Shimony and O. Tsarfati, “Chatting Our Way Into Creating a Polymorphic Malware,” CyberArk Threat Research Blog, 2023.

[12] A. Chawla and U. Prasad, “A Decompilation-Driven Framework for Malware Detection with Large Language Models,” arXiv preprint arXiv:2601.09035, 2026.

[13] R. Saul, J. Jiang, E. Chia, and D. Wagner, “Trident: Improving Malware Detection with LLMs and Behavioral Features,” arXiv preprint arXiv:2605.00297, 2026.