Computer Modeling in Engineering & Sciences

Tech Science Press

Doi:10.32604/cmes.2025.064104

ARTICLE

Malware of Dynamic Behavior and Attack Patterns Using ATT&CK Framework

Jong-Yih Kuo[1], Ping-Feng Wang[2,*], Ti-Feng Hsieh[1,*] and Cheng-Hsuan Kuo[1]

[1]Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei, 106344, Taiwan
[2]Department of Artificial Intelligence and Computer Engineering, National Chin-Yi University of Technology, Taichung, 411030, Taiwan

*Corresponding Authors: Ping-Feng Wang. Email: pfwang@ncut.edu.tw; Ti-Feng Hsieh. Email: t112599002@ntut.edu.tw

Received: 05 February 2025; Accepted: 19 May 2025; Published: 30 June 2025

ABSTRACT: In recent years, cyber threats have escalated across diverse sectors, with cybercrime syndicates increasingly exploiting system vulnerabilities. Traditional passive defense mechanisms have proven insufficient, particularly as Linux platforms—historically overlooked in favor of Windows—have emerged as frequent targets. According to Trend Micro, there has been a substantial increase in Linux-targeted malware, with ransomware attacks on Linux surpassing those on macOS. This alarming trend underscores the need for detection strategies specifically designed for Linux environments. To address this challenge, this study proposes a comprehensive malware detection framework tailored for Linux systems, integrating dynamic behavioral analysis with the semantic reasoning capabilities of large language models (LLMs). Malware samples are executed within sandbox environments to extract behavioral features such as system calls and command-line executions. These features are then systematically mapped to the MITRE ATT&CK framework, incorporating its defined data sources, data components, and Tactics, Techniques, and Procedures (TTPs). Two mapping constructs—Conceptual Definition Mapping and TTP Technical Keyword Mapping—are developed from official MITRE documentation. These resources are utilized to fine-tune an LLM, enabling it to semantically interpret complex behavioral patterns and infer associated attack techniques, including those employed by previously unknown malware variants. The resulting detection pipeline effectively bridges raw behavioral data with structured threat intelligence. Experimental evaluations confirm the efficacy of the proposed system, with the fine-tuned Gemma 2B model demonstrating significantly enhanced accuracy in associating behavioral features with ATT&CK-defined techniques. This study contributes a fully integrated Linux-specific detection framework, a novel approach for transforming unstructured behavioral data into actionable intelligence, improved interpretability of malicious behavior, and a scalable training process for future applications of LLMs in cybersecurity.

KEYWORDS: Linux malware; dynamic analysis; behavior analysis; behavioral feature; ATT&CK; sandbox; large language model; fine-tuning

1 Introduction

1.1 Motivation and Objectives

With the rapid advancement of information technology, cyberattacks have become increasingly prevalent across various domains, making information security a critical global concern. According to FortiGuard Labs' 1H 2023 Global Threat Landscape Report [1], the volume of malware activities detected in Taiwan during the first half of 2023 nearly doubled (96.85%) compared to the same period in 2022. This dramatic increase highlights the growing scale and sophistication of cybercriminal operations, as threat actors continue to expand both the scope and diversity of their malware campaigns.

Furthermore, Trend Micro's 2022 Annual Cybersecurity Report [2] revealed a shift in ransomware targeting strategies, indicating that attacks are no longer limited to traditional platforms like Windows and macOS. Instead, Linux systems are increasingly becoming a target. The number of ransomware attacks against Linux systems was reported to be nearly double that of attacks on macOS in 2022. Trend Micro's 2023 Mid-Year Cybersecurity Report [3] further confirms this trend, noting that incidents involving Linux-based systems increased by over 62% from the first half of 2022 to the first half of 2023. These statistics suggest that threat actors have begun to exploit vulnerabilities in Linux environments, which are often under-researched and insufficiently protected compared to other operating systems.

Despite the growing threat landscape, most malware detection research in recent years has focused primarily on the Windows platform, with limited attention given to Linux-based malware. In light of this imbalance, our study proposes a dedicated technical framework for detecting malware on the Linux platform, with a focus on dynamic behavior analysis and semantic threat interpretation.

In this study, we collect malware samples from public repositories and perform dynamic analysis using Cuckoo Sandbox [4] to observe and record their behavior in a simulated execution environment. The resulting behavioral data, such as system calls and execution traces, are essential indicators of malicious intent. However, due to the complexity and variability of these behavioral sequences, manual interpretation is difficult. To address this, we analyze the frequency and patterns of behavioral features to extract common attack traits across malware families. These behaviors are then semantically mapped to the MITRE ATT&CK framework [5,6], which standardizes adversary tactics, techniques, and procedures (TTPs).

TTPs refer to the tactical goals adversaries aim to achieve (Tactics), the methods used (Techniques), and the detailed execution patterns (Procedures). By aligning behavioral features with MITRE ATT&CK techniques, we provide a structured and intuitive interpretation of malware activity. To support this mapping, we construct a TTP Keyword Mapping Table that correlates observed system behaviors with their respective ATT&CK techniques based on documented definitions.

To further enhance behavioral interpretation, we employ a large language model (LLM) capable of understanding the semantics of human language and contextual associations between technical elements. The model is fine-tuned using prompt-engineered datasets derived from behavioral logs and MITRE ATT&CK mappings. This allows the LLM to accurately infer the likely data sources, data components, and technique combinations used by malware during execution.

1.2 Contributions

This study presents a novel malware detection framework targeting the Linux platform, integrating dynamic behavioral analysis with semantic modeling based on the MITRE ATT&CK framework. The key contributions are outlined as follows:

First, we propose a behavior-based detection framework for Linux malware by collecting dynamic execution traces and mapping them to standardized techniques in the MITRE ATT&CK framework. Through the construction of TTP Keyword and Conceptual Definition Mapping Tables, the system bridges low-level behavioral data with high-level semantic representations, enabling structured and interpretable threat analysis.

Second, we introduce a prompt engineering strategy that transforms observed malware behavior into structured instruction-response pairs. These are used to fine-tune a large language model (Gemma 2B) to classify and interpret malware behaviors in terms of ATT&CK techniques.

Third, our approach enhances the explainability of malware detection. By mapping complex behaviors to standardized attack techniques, the system provides interpretable output that security analysts can use for threat investigation and response.

Finally, the proposed framework addresses a significant research gap by focusing specifically on Linux-based threats, which remain underrepresented in malware detection literature despite their growing prevalence in cloud and enterprise environments.

2 Background and Previous Work

2.1 Malware Detection

Malware detection is the process of identifying or detecting programs that may be suspicious or potentially threatening by analyzing the data, code, or behavior in a computer system to identify and isolate malicious programs. Malware detection is mainly divided into signature detection and behavior detection.

2.2 MITRE ATT&CK

The MITRE ATT&CK Framework, a research project initiated by the MITRE organization in May 2015, makes public a total of nine tactic types and 96 techniques contained within each type, with the tactic types corresponding to the various stages of the malware attack lifecycle, and the techniques describing the means required to achieve the goals of a particular tactic type. The framework provides a way to characterize the different attack techniques used by malicious attackers and their tactical phases in a matrix categorization. The ATT&CK matrix uses the concepts of Tactic, Technique, and Procedure as the core of analyzing malicious attack behaviors, and the descriptions of these three concepts are shown in Table 1.

Table 1: The meaning of TTP in the MITRE ATT&CK framework

Name	Meaning
Tactic	The tactical goal of an attack describes the ultimate goal that the attacker hopes to achieve by using a specific technique.
Technique	Techniques Used to Accomplish Tactical Objectives describe the methods used by an attacker to accomplish a specific tactical objective, which may be a system vulnerability or a specific tool.
Procedure	Procedures for implementing an attack technique describe the behaviors or actions taken by an attacker in order to use a specific technique in order to accomplish a tactical objective.

2.3 Large Language Model

Large Language Model (LLM) is a deep learning model based on Natural Language Processing (NLP) technology, which is mainly used for understanding, generating and manipulating human language, but also for recognizing and generating relevant textual content.

Large languages are often modeled with large amounts of textual information, often derived from articles and websites on the Internet. Repeated training allows the model to learn the complex structure and context of human language. Large language models can be used in a wide variety of applications, such as intelligent assistants, chatbots, sentiment analysis, speech recognition, and machine translation, which dramatically improve productivity and user experience.

2.3.1 Gemma

Gemma [7] is an open-source large language model developed by Google DeepMind. It is specifically optimized for responsible AI deployment and research usage, supporting both instruction-following and general-purpose language understanding. Gemma models are designed with a focus on efficiency and performance, making them suitable for a wide range of academic and industrial applications. Based on the Transformer decoder-only architecture, Gemma is trained on curated datasets with reinforcement learning and safety-aligned techniques, enabling it to perform well on natural language generation, reasoning, and comprehension tasks.

2.4 Prompt Engineering

Prompt engineering refers to the practice of designing and refining input prompts to elicit optimal responses from large language models. For models like Gemma, prompt engineering can significantly impact the quality of outputs by guiding the model toward context-relevant and precise answers. This process involves creating prompts that are specific, concise, and tailored to the task at hand, followed by iterative testing and refinement to enhance response accuracy and relevance. Effective prompt engineering is especially valuable in zero-shot or few-shot learning scenarios.

2.5 Fine-Tuning

Fine-tuning is the process of adapting a pre-trained language model like Gemma to a specific domain or task by further training it on a task-specific dataset. This allows the model to learn domain-specific vocabulary, semantics, and task requirements, thereby improving its performance on specialized applications. Fine-tuning leverages the generalized language understanding already embedded in Gemma, which reduces the need for full retraining, lowers computational costs, and improves efficiency.

2.6 LoRA

LoRA (Low-Rank Adaptation) is a parameter-efficient fine-tuning technique proposed by Hu et al. [8] and is widely applied to large language models such as Gemma. Traditional fine-tuning involves updating all model parameters, which can be computationally prohibitive due to the billions of parameters in modern large models. To address this challenge, LoRA introduces trainable low-rank matrices into specific projection layers, such as the query, key, and value projections in the attention mechanism.

This approach dramatically reduces the number of parameters that need to be trained, while still maintaining the original model's performance. LoRA is particularly beneficial for deploying Gemma models in resource-constrained environments, as it lowers GPU memory usage, accelerates training, and enables efficient domain adaptation without full model retraining.

2.7 Previous Work

In the field of malware analysis, Catak et al. [9] used the Cuckoo Sandbox to dynamically analyze malware targeting the Windows platform. Their approach involved recording Windows API calls during sandbox execution and providing a benchmark dataset for Windows malware. Chierzi et al. [10] proposed a methodology to track the evolution of malware capabilities using the MITRE ATT&CK framework. Their method combined static and dynamic analysis techniques, mapping the results to ATT&CK-defined techniques and comparing the methods used by the same type of malware over a two-year interval. Ahn et al. [11] introduced a behavioral visualization approach for malware detection on the Windows platform. Their process applied static analysis to classify software as benign, malicious, or suspicious, followed by

dynamic analysis on suspicious samples to extract system calls. These behavioral features were then mapped to the MITRE ATT&CK framework to provide a more intuitive representation of tactics and techniques, thereby aiding in the determination of whether the software was malicious. To enhance detection accuracy, their model utilized ensemble learning techniques, including Random Forest, AdaBoost, and Gradient Boosting, through iterative training and testing.

Sartoli et al. [12] introduced the concept of recurrence plots to visualize the binary structure of malware files, graphically representing their intrinsic characteristics. Malware samples belonging to the same family exhibit similar recurrence plots, enabling the classification of malware families using convolutional neural networks (CNNs). Shiva Darshan and Jaidhar [13] proposed a CNN-based malware detection model for the Windows platform, utilizing system calls and their operational categories as dynamic features. These features are refined through feature selection techniques to generate black-and-white images, which are then input into a CNN to extract higher-order features for classification. Xu et al. [14] developed a malware behavior detection model for Windows that combines the strengths of CNN and long short-term memory (LSTM) networks. Their hybrid deep learning model analyzes file names and system call information to distinguish between benign software and seven different malware types. Similarly, Zhang [15] employed a CNN-LSTM model to detect dynamic malware behavior by capturing both spatial correlations and temporal sequences within system call patterns, thereby producing a high-level representation of behavioral features. Anandhi et al. [16] proposed a detection method that combines bidirectional LSTM (BiLSTM) with CNN for identifying malware on Windows systems. Their model classifies software as benign or as one of five specific malware families, demonstrating superior accuracy compared to models using BiLSTM or CNN alone. He et al. [17] integrated self-attention mechanisms with LSTM for enhanced malware detection, leveraging attention to model interdependence among system calls and LSTM to capture sequence order. Or-Meir et al. [18] applied Transformer models to classify malware families on the Windows platform and studied the impact of system call sequence length on detection performance. Their findings show that Transformer-based models not only outperform LSTM-based models in accuracy but also reduce training and detection time by up to 75%. Walker et al. [19] developed a malware family classification system for Windows using dynamic analysis to extract system call combinations. Their model, based on Single Hidden Layer Feedforward Neural Networks (SLFN), including ELM and OS-ELM variants, accurately detects and classifies malware families.

In addition to the aforementioned approaches, An et al. [20] proposed a dual Siamese network-based malware detection framework that leverages few-shot learning techniques to enhance detection of novel and variant malware. Their model integrates two independent Siamese networks—one trained on bytecode grayscale images and the other on opcode 2-g frequency images. By measuring similarity between image pairs using L1 distance, the system achieves high accuracy with limited data, showcasing the effectiveness of similarity-based models in identifying malware variants. This approach, while powerful, is primarily visual and static in nature and lacks semantic interpretability of behavioral patterns and attack techniques, which is the focus of our study.

In the domain of applying large language models (LLMs) to information security, several studies have demonstrated their potential for enhancing threat detection capabilities. Sekharan et al. [21] utilized an LLM for clickbait title detection, employing OpenAI's Ada model for fine-tuning. Their approach enabled the model to effectively identify malicious intent behind deceptive titles, achieving a detection accuracy of up to 99.5%. Li et al. [22] proposed a network security vulnerability detection framework based on a fine-tuned GPT-3.5 model provided by OpenAI [23]. The framework demonstrated the ability to deeply

analyze common cybersecurity vulnerabilities and generate corresponding mitigation strategies. Similarly, Li et al. [24] introduced a method for detecting malicious web pages using an LLM. By analyzing webpage URLs and textual content, their method leveraged zero-shot and few-shot prompting strategies to adapt the LLM for detection tasks. The results showed that this approach outperformed traditional deep learning methods designed for the same purpose, highlighting the adaptability and effectiveness of LLMs in web threat detection.

From the reviewed literature, it is evident that the majority of recent research efforts have focused on the Windows platform, with relatively limited attention given to malware targeting Linux environments. However, across all platforms, system calls and their sequential patterns serve as critical indicators for analyzing malware attack behaviors. By further examining these behavioral characteristics, it is possible to abstract and generalize the underlying malicious actions. This study aims to analyze the behavioral features of malware during execution in a real environment and interpret them through the lens of the MITRE ATT&CK framework. To this end, we propose an analysis process tailored for detecting Linux malware attack techniques, employing dynamic analysis as the primary method. Software samples are executed in a Cuckoo Sandbox to monitor actual runtime behavior. The system calls invoked by malware during execution are extracted and used as key behavioral features. These features are then analyzed using a large language model (LLM) that has been fine-tuned through prompt engineering. The objective is to semantically interpret the observed behaviors and map them to the corresponding Tactics, Techniques, and Procedures (TTPs) defined in the MITRE ATT&CK framework. This approach enables a more precise and automated understanding of Linux malware behavior within a structured threat intelligence framework.

3 Proposed Approach

3.1 System Architecture

This study proposes a malware technology detection process, which combines the MITRE ATT&CK framework as the basis of the study, as illustrated in Fig. 1. This study uses a malware database to download the required malware as the malware samples for this study. Then, the sample format is verified by the behavior analysis module to ensure that the sample can be executed properly in the sandbox environment, and the sample is uploaded to the Cuckoo sandbox for dynamic analysis. After obtaining the behavioral track report of the sample, the behavioral track of the sample is extracted as the behavioral characteristics of the malicious attack are extracted through the technical detection module, and then the frequency of the behavioral characteristics and the analysis of the attack pattern are further conducted. Then, we obtained the relevant information recorded in the MITRE ATT&CK framework, including information on data sources, data components, and concepts such as TTP, and organized and summarized them into a concept mapping table and a TTP technical keyword mapping table as defined in this study. The Concept Mapping Table and TTP Technical Keyword Mapping Table are used for model training on large language models, and the behavioral characteristics of attack patterns are used as model inputs for the technical detection of the MITRE ATT&CK framework.

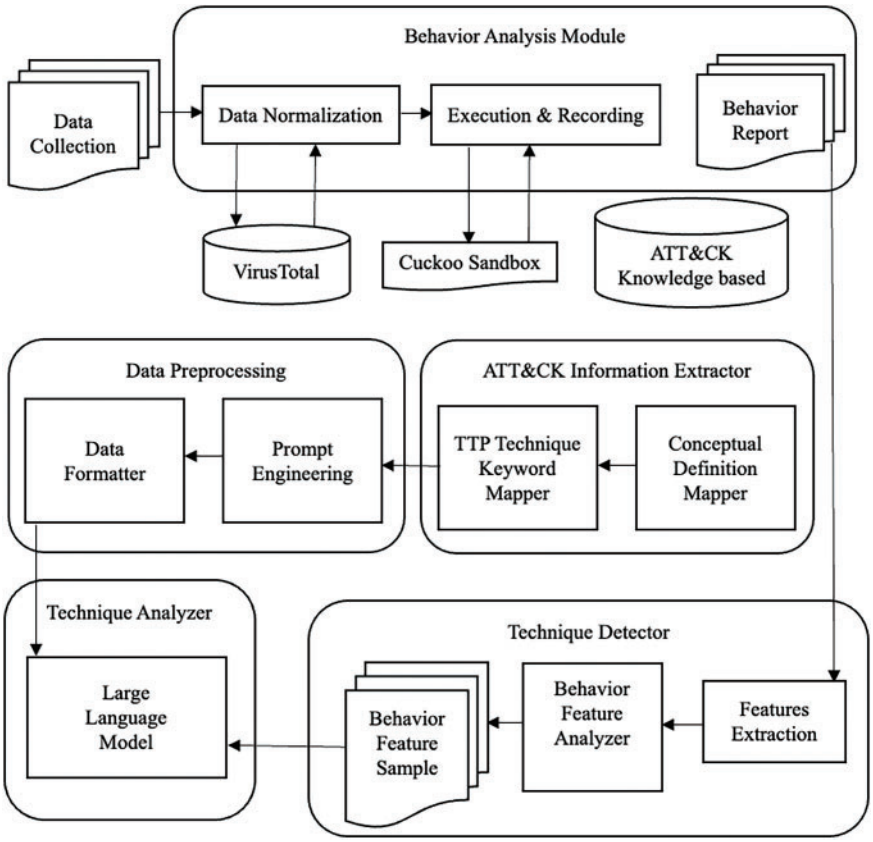

Figure 1: System architecture

3.2 Data Collection

3.2.1 Malware Sample Download

This study focuses on analyzing various types and families of malware that can be executed on the Linux platform. Accordingly, malware samples are selected based on specific criteria to ensure a diverse and representative dataset. Samples are collected from several well-known malware repositories, including VirusShare [25], MalShare [26], MalwareBazaar [27], and vx-underground [28]. Among these, vx-underground is a comprehensive and widely used malware archive that aggregates samples from multiple sources and supports advanced search capabilities for retrieving specific types of malware. These repositories provide a broad spectrum of malicious samples, enabling this study to perform a robust evaluation of Linux-targeted malware behaviors.

3.3 Behavioral Analysis Module

The behavioral analysis module of this study can be divided into two steps: the first step is the data normalization, and the second step is the behavioral trajectory report of the sample implementation record and compilation.

3.3.1 Data Normalization

In the data regularization phase, particular attention is given to the compatibility of malware formats with the Linux platform, as the file format significantly impacts the success of subsequent dynamic analysis.

Therefore, it is essential to verify the format of each collected malware sample before execution in a sandbox environment. To achieve this, static analysis is performed using the VirusTotal malware analysis system, as illustrated in Fig. 2.

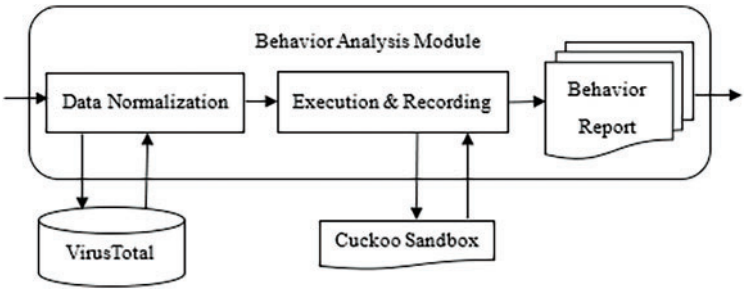

Figure 2: Data normalization in this study

VirusTotal is a widely used, freely accessible malware analysis service that enables analysts to evaluate suspicious files or URLs via both a web interface and an API. It provides comprehensive static analysis results, including detailed information on file structure, format, and potential threat indicators. Analysts can input a file's hash value (e.g., MD5, SHA-256), IP address, domain, or URL to initiate a static analysis.

The static analysis results include detailed metadata about the malware sample, such as the executable file format (e.g., ELF for Linux), the applicable instruction set architecture (e.g., x86-64), the linking type (e.g., statically linked), and the file size (e.g., 156.27 KB), among other technical attributes.

In this study, VirusTotal is used to verify the structural properties of each malware sample, including the file type, instruction set architecture, linking method (static or dynamic), and required dynamic libraries. This verification step is crucial to ensure that all samples are compatible with the sandbox environment used for dynamic behavioral analysis and can be executed reliably without format-related errors.

3.3.2 Sample Execution Record and Export Behavioral Track Report

After confirming that the malware samples are executable within the sandbox environment, dynamic analysis is conducted to observe and record their runtime behavior. In this study, the Cuckoo Sandbox is employed as the primary tool for dynamic analysis.

Upon uploading a malware sample to the Cuckoo Sandbox, the system executes the sample in an isolated virtual environment and monitors its behavior throughout the execution process. This includes recording system calls, command-line instructions, and any application or API calls invoked by the malware. All observed actions are systematically documented and exported as a comprehensive behavioral malware trajectory report, which serves as the foundation for further behavioral feature extraction and analysis.

3.4 Technique Detector

The technique detection process in this study consists of three main steps: (1) behavioral feature extraction, (2) behavioral feature analysis, and (3) the construction of a behavioral feature sample corresponding to each malware instance.

3.4.1 Feature Extraction

Cuckoo Sandbox is employed to capture all system calls invoked by the malware sample during execution. These system calls constitute the core behavioral features and are compiled into the behavioral trajectory report. As illustrated in Fig. 3, the report includes a top-level "behavior" field, which contains a "processes" subfield that records the activities of each process spawned during malware execution.


```
"behavior": {
    "generic": [···
    ],
    "processes": [
        {
            "calls": [···
            ],
            "pid": 818,
            "process_name": "2d77592c3c46ba05a18655430e5a191a16e68c308ed1406a73a4dae156adca32.elf",
            "command_line": "/tmp/2d77592c3c46ba05a18655430e5a191a16e68c308ed1406a73a4dae156adca32.elf",
            "first_seen": 1698510824.703906,
            "ppid": 817,
            "type": "process"
        },
        {
            "calls": [···
            ],
            "pid": 819,
            "process_name": "2d77592c3c46ba0",
            "command_line": "",
            "first_seen": 1698510824.728718,
            "ppid": 818,
            "type": "process"
        }
    ],
```


Figure 3: Illustration of the behavior field of the behavior track report

Each process may invoke multiple system calls, each accompanied by specific parameters and return values. As shown in Fig. 4, within the "processes" subfield, the "calls" list contains individual entries, where:

- "api" specifies the name of the system call,
- "arguments" provides the parameter list passed to the call,
- "return_value" records the result returned after execution.

These fields collectively represent the malware's dynamic behavior during runtime. In this study, all "api" calls captured within the "processes" structure are extracted as primary behavioral features for downstream analysis, including frequency modeling and semantic mapping to ATT&CK techniques.

3.4.2 Behavior Feature Analyzer

Following feature extraction, a set of behavioral features is compiled for each malware sample. Since malware may invoke the same system, call multiple times, or follow repeated behavioral patterns during execution, the resulting dataset may contain many redundant or repeated features and sequences. These repetitions often reflect characteristic attack behaviors, known as attack patterns.

```
"calls": [
    {
        "status": "",
        "raw": "Sat Oct 28 16:33:44 2023.703906 2d77592c3c46ba0@7f93c24110a9[818] execve(\"/tmp/2d77
        "api": "execve",
        "return_value": "",
        "instruction_pointer": "7f93c24110a9",
        "time": 1698510824.703906,
        "process_name": "2d77592c3c46ba0",
        "pid": 818,
        "arguments": {
            "p2": [···
            ],
            "p0": "/tmp/2d77592c3c46ba05a18655430e5a191a16e68c308ed1406a73a4dae156adca32.elf",
            "p1": [
                "/tmp/2d77592c3c46ba05a18655430e5a191a16e68c308ed1406a73a4dae156adca32.elf"
            ]
        }
    },
```


Figure 4: Illustration of the system calls used by the program

This study emphasizes both the frequency analysis and pattern recognition of these behavioral features. Repetitive features and sequences are normalized and generalized to reduce their noise effect and highlight the core malicious behavior. The focus is placed on identifying dominant behavioral patterns that represent the malware's functional intent.

After analysis, each behavioral trajectory report is transformed into a structured and summarized set of behavioral features. These processed samples are then prepared for use as input to the detection model, ensuring consistency and relevance in the modeling phase.

3.5 ATT&CK Information Extractor

The ATT&CK information extraction process in this study is divided into two main steps: (1) Conceptual Definition Mapping, and (2) TTP Technical Keyword Mapping.

This study adopts a two-step approach for ATT&CK information extraction. In the first step, data sources defined within the MITRE ATT&CK framework are collected, along with their corresponding data components and detailed descriptions. These data sources represent types of system or application-level information that can be captured, while data components define specific attributes relevant to identifying and detecting attack techniques. In the second step, technical keywords annotated on each technique's official page are extracted. These keywords are further categorized and generalized to enable more accurate attribution of observed behavioral features to specific Tactics, Techniques, and Procedures (TTPs). By systematically mapping behavioral characteristics to corresponding ATT&CK techniques, based on both keyword associations and relevant data sources and components, this study establishes a robust foundation for detecting and interpreting malicious behaviors within a structured threat intelligence framework.

3.5.1 Conceptual Definition Mapper

This study extracts descriptions of both data sources and data components as defined by the MITRE ATT&CK framework [29]. The framework currently defines 37 distinct data sources and 106 data components. Each data source represents a category of information typically collected in a system's logs, while each data component specifies attributes used to identify and detect ATT&CK techniques or sub-techniques.

Fig. 5 illustrates an example of a data source: Command, which represents an instruction given to a system to perform a specific operation. This study extracts the name and description of such data sources for mapping purposes.

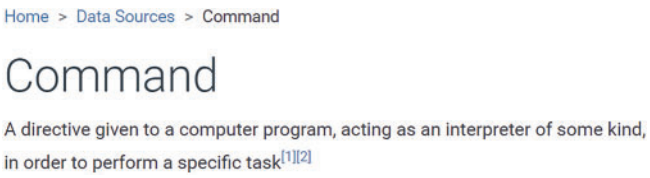

Figure 5: Illustration of MITRE ATT&CK data source description

Fig. 6 presents an example of a data component: Command Execution, which falls under the Command data source. This component refers to a line of executable code derived from a program and executed within the system. The names and descriptions of data components such as this are also extracted for use in this study.

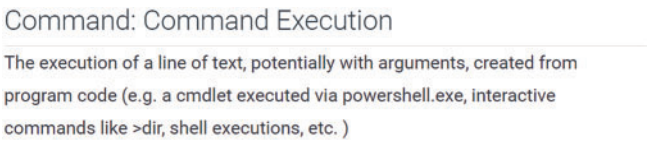

Figure 6: Illustration of MITRE ATT&CK data component descriptions

Furthermore, this study collects detailed information on all techniques and sub-techniques defined in the MITRE ATT&CK framework, including their identifiers, names, and descriptions. These TTPs represent the specific methods or actions used by adversaries to accomplish tactical objectives. At the time of this study, the framework includes 201 techniques and 424 sub-techniques.

Fig. 7 illustrates a TTP example: *Abuse Elevation Control Mechanism: Setuid and Setgid* (T1548.001). The figure shows the TTP name, its unique identifier, and a description detailing the implementation or potential impact of the technique. To support the model training and semantic mapping process, this study creates the following structured mapping tables:

- Mapping Table of Data Sources: Table 2 is a mapping of the data sources defined in this study, with different columns for different data sources and their descriptions. The description of each column of the table is shown in the table.
- Mapping Table of Data Component: Table 3 is a mapping of the data components defined in this study, each of which belongs to a specific data source. Each data element belongs to a specific data source. This table is divided into columns based on different data elements and their descriptions. Descriptions of the individual columns of the table are shown in the table.
- Mapping Table of TTP Technology: Table 4 is a mapping of the TTP technologies defined in this study, with each TTP technology serving a different purpose. Each TTP technology has its own function. The

table is divided into different columns according to the different technologies and their descriptions. The description of each column of the table is shown in the table.

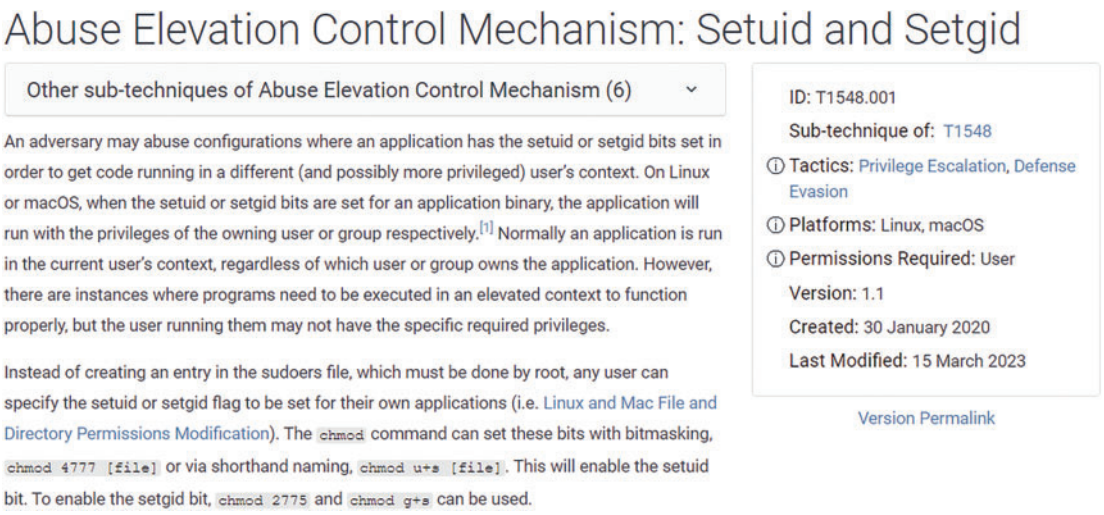

Figure 7: Illustration of MITRE ATT&CK TTP technical description

Table 2: Mapping table of data sources column description

Columns	Descriptions
data_source	Sources of information defined in the MITRE ATT&CK framework
data_source_description	Specific description of data sources in the MITRE ATT&CK framework definition

Table 3: Mapping table of data component column description

Columns	Descriptions
data_source	Data sources belonging to the data components defined in the MITRE ATT&CK framework
data_component	Data components defined in the MITRE ATT&CK framework
data_component_description	Specific description of data elements in the MITRE ATT&CK framework definition

Table 4: Mapping table of TTP technology column description

Columns	Descriptions
ttp_id	TTP technology number defined in the MITRE ATT&CK framework
ttp_name	TTP technology name as defined in the MITRE ATT&CK framework
ttp_description	Technical description of the TTP as defined in the MITRE ATT&CK framework

3.5.2 TTP Technique Keyword Mapper

To enhance the precision of malware behavior interpretation, this study establishes a systematic method for extracting, categorizing, and mapping technical keywords from the MITRE ATT&CK framework. The goal is to semantically align low-level behavioral features, such as system calls, commands, and file interactions, captured through dynamic analysis, with high-level adversary tactics and techniques.

The MITRE ATT&CK framework currently defines 202 techniques and 435 sub-techniques. Each entry contains detailed textual descriptions outlining how adversaries can achieve specific tactical goals. Within these descriptions, MITRE highlights critical operational elements—such as commands, behaviors, or low-level system interactions—using visual indicators (e.g., gray background text). These keywords are central to understanding the execution logic of each technique and are used in this study as semantic anchors for behavior classification.

As shown in Fig. 8, the technique T1548.001: Abuse Elevation Control Mechanism: Setuid and Setgid includes operational details such as the use of chmod 4777 or chmod u + s to escalate privileges. These command patterns are extracted as technical keywords, which serve as direct behavioral indicators of specific attack strategies. By generalizing such indicators across all techniques, this study builds a structured mapping between system-level observations and ATT&CK-defined TTPs (Tactics, Techniques, and Procedures).

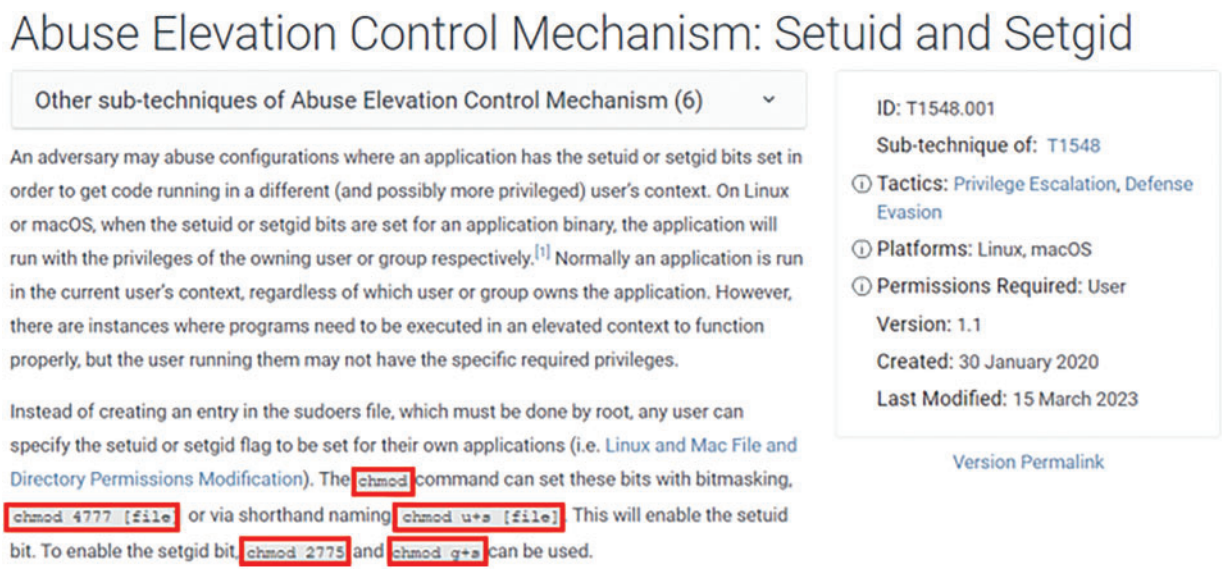

Figure 8: Illustration of the MITRE ATT&CK technical description

In addition to keyword extraction, the study also incorporates data sources, data components, and detection method information from ATT&CK. These elements define how a given technique can be practically detected. For example, as illustrated in Fig. 9, detection of T1548.001 may rely on two distinct methods:

- Command-line monitoring of tools such as chmod, categorized under the Command Execution component of the Command data source.
- File metadata inspection, categorized under the File Metadata component of the File data source.

By collecting and cross-referencing such detection methods, the study constructs a holistic framework for mapping behaviors not only to techniques but also to their observable characteristics in real systems.

Detection

ID	Data Source	Data Component	Detects
DS0017	Command	Command Execution	Monitor for execution of utilities, like chmod, and their command-line arguments to look for setuid or setguid bits being set.
DS0022	File	File Metadata	Monitor the file system for files that have the setuid or setgid bits set.
		File Modification	Monitor for changes made to files that may perform shell escapes or exploit vulnerabilities in an application with the setsuid or setgid bits to get code running in a different user's context.

Figure 9: Illustration of MITRE ATT&CK data element detection technology description

To improve semantic granularity, technical keywords are classified across multiple dimensions:

- Platform (e.g., Linux, Windows),
- Type (e.g., system call, file path, registry key),
- Keyword Role (e.g., command, parameter, file name),
- Execution Context (e.g., process-level, user-level).

Compound keywords—such as those composed of a command and its associated parameters or path—are further parsed and mapped into their components to allow precise classification and contextual interpretation.

Beyond static keyword classification, the study also performs behavioral analysis of each system call and command. This involves evaluating their function, operational context, and impact on the host system. Only those actions that reliably indicate adversarial behavior are retained and incorporated into the TTP Technical Keyword Correspondence Table, a core knowledge structure developed in this research.

To organize and utilize this information systematically, two mapping tables are constructed:

- Mapping Table of TTP Technique Keyword: Table 5 is a TTP Technology Keyword Mapping Table as defined by this research, which is categorized by different columns according to the keywords of the technology. The description of each field in the table is shown in the table. Each TTP technology keyword contains information about the data source, data component, TTP technology number, technology name, and technology keyword, which also contains information about the technology keyword, such as the platform, type, command, parameter, file path, and file for the technology keyword.
- Mapping Table of Data Component Detection: Table 6 is a mapping of the data element detection techniques defined in this study. Each data element is capable of detecting a number of different techniques and is described specifically according to the detection method. The individual columns of the table are described as shown in the table.

In summary, this keyword-based mapping framework bridges the gap between low-level malware behavior and high-level threat intelligence models. By integrating technique descriptions, command patterns, and data-driven detection logic, the proposed system enables fine-grained, explainable attribution of malware activities to ATT&CK-defined techniques. These mappings play a foundational role in the semantic classification pipeline and in the training of large language models to interpret dynamic malware behavior with precision and clarity.

Table 5: Mapping Table of TTP technology column description

Columns	Descriptions
data_source	In the MITRE ATT&CK framework definition, it is possible to detect the sources of information about the technology
data_component	In the MITRE ATT&CK framework definition, the data components that enable the detection of the technology
ttp_id	The number of the technology as defined in the MITRE ATT&CK framework, where a technology with a decimal point in the number represents a sub-technology
ttp_name	The name of the technology is defined in the MITRE ATT&CK framework
ttp_keyword	The technology keywords, which may be of different types of information
keyword_platform	The platform to which the keyword applies, such as Windows or Linux
keyword_type	The type of the keyword may be of different types, such as file or command.
command	The command used for the keyword
parameter	The parameters used for this keyword may be those used for commands or system calls
path	The file path used by the keyword
file	The file used for the keyword

Table 6: Mapping table of data component detection column description

Columns	Descriptions
data_source	In the MITRE ATT&CK framework definition, it is possible to detect the sources of information about the technology
data_component	In the MITRE ATT&CK framework definition, the data elements that enable the detection of the technology
ttp_id	The number of the technology as defined in the MITRE ATT&CK framework, where a technology with a decimal point in the number represents a sub-technology
ttp_detect_description	Description of the specific method by which the data element detects the technology

3.6 Data Preprocessing

The malware samples and structured datasets used in this study include a large volume of heterogeneous data types and complex textual descriptions. Therefore, appropriate preprocessing is essential to ensure compatibility with the fine-tuning requirements of large language models (LLMs). To facilitate the use of LLMs in answering user queries or understanding contextual inputs via system prompts, the information collected from the MITRE ATT&CK framework is subjected to a process of prompt engineering.

Additionally, because this study employs Gemma as the base model for fine-tuning, it is critical to ensure that the training data conforms to the specific formatting and structural requirements of Gemma fine-tuning. This includes adapting data for instruction tuning and supervised fine-tuning formats, both of which impose

unique constraints on input/output structures. Consequently, all data intended for model training must be transformed into a standardized format that aligns with the LLM's fine-tuning scheme.

3.6.1 Prompt Engineering

In this study, prompt engineering is applied to the Conceptual Definition Mapping Table and the TTP Technical Keyword Mapping Table derived from the MITRE ATT&CK framework. The goal is to enable the large language model (LLM) to learn and understand the key concepts defined within the framework and their interrelationships. These include data sources, data components, TTP techniques, and technical keywords. As illustrated in Fig. 10, data sources encompass multiple data components, which in turn can detect one or more TTP techniques. Each technique may be associated with several technical keywords that contribute to achieving specific tactical objectives.

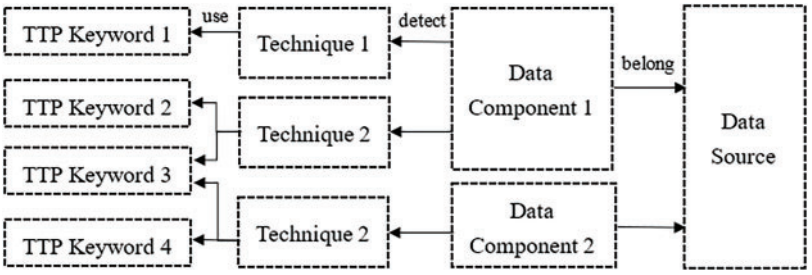

Figure 10: MITRE ATT&CK conceptual relationship diagram

To train the LLM effectively, the relationships between these elements are systematically organized and summarized into structured prompts. These prompts are carefully designed to be concise, reducing computational complexity during inference, improving response time, and lowering overall model usage cost. The following examples demonstrate several use cases of prompt engineering implemented in this study.

- Data Sources Recognition: The system prompt is configured to provide an object description, and the model is expected to identify the corresponding data source. As shown in Fig. 11, given a description of the "File" data source, the model correctly identifies it as "Data source: File."
- Data Component Recognition: As illustrated in Fig. 12, the prompt includes the name and description of a data source along with a data component description. LLM is tasked with recognizing the correct data component, e.g., identifying "File Creation" as a component of the "File" data source.
- TTP Technique Recognition: For technique identification, the system prompt provides a textual description of the technique. As shown in Fig. 13, the model receives the description and responds with the correct technique ID and name, such as "Technique: T1548.001, Setuid and Setgid."
- Technique Detection via Data Components: In this case, the prompt provides a data component and its detection description. The model is instructed to return the TTP techniques that can be detected using this information. As shown in Fig. 14, the model successfully identifies multiple techniques that correspond to the given data.
- Data Component Detection via System Calls: This prompt format aims to link system calls to their corresponding data components. As shown in Fig. 15, the system called "openat" with the description "open and possibly create a file" is correctly mapped to components such as "Process Creation," "Command Execution," and "OS API Execution."

- TTP Technique Identification via System Calls: Finally, Fig. 16 shows how the model is used to infer the corresponding TTP technique based on a system call and its description. Given the system called "openat," the model identifies techniques that may utilize this call.

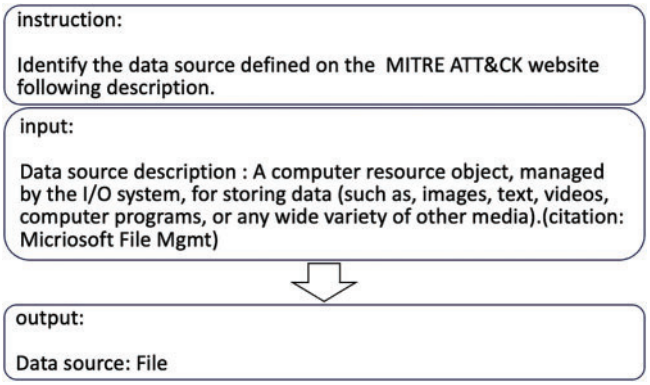

Figure 11: Prompt engineering of data sources

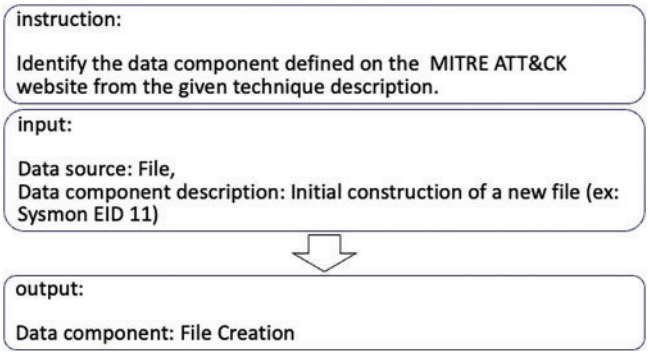

Figure 12: Prompt engineering of data components

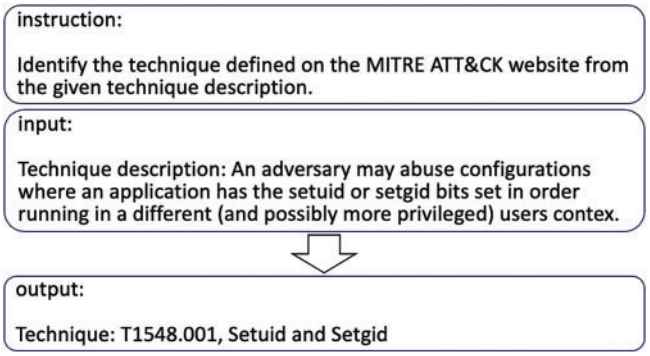

Figure 13: Prompt engineering of TTP technology

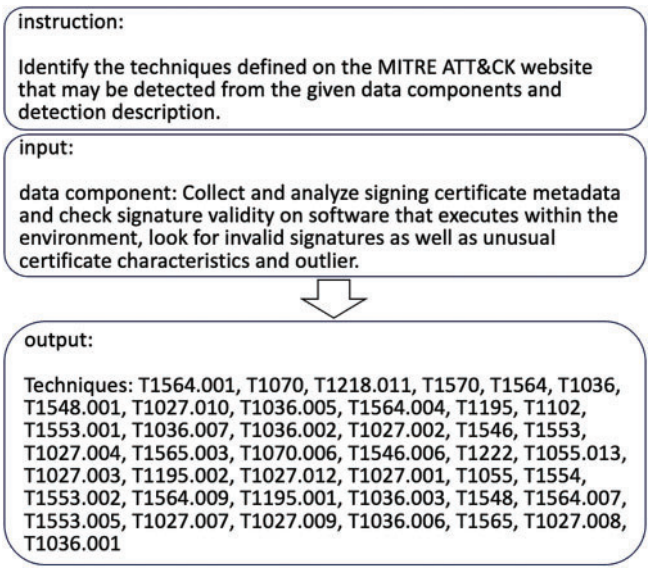

Figure 14: Prompt engineering of data component detect technique

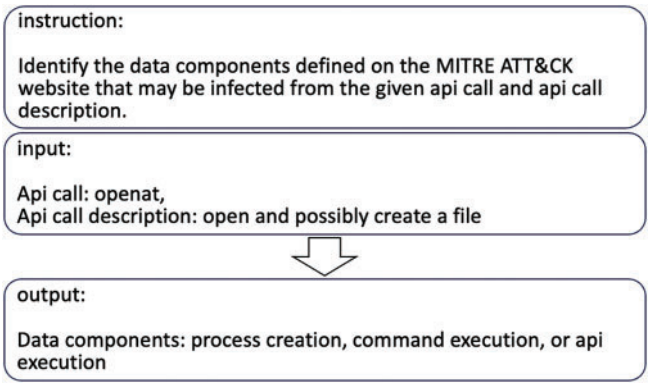

Figure 15: Prompt engineering of TTP technology keywords and corresponding data components

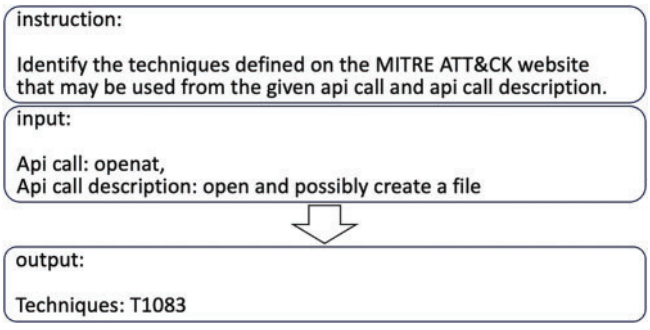

Figure 16: Prompt engineering of TTP technologies corresponding to TTP technology keywords

These structured prompt engineering methods enable the LLM to semantically understand and infer the relationships between low-level system behaviors and high-level adversarial techniques. This

facilitates accurate mapping of dynamic behavioral features to MITRE ATT&CK techniques during the detection phase.

3.6.2 Data Formatter for Gemma

For fine-tuning the Gemma large language model in this study, the Alpaca dataset format is adopted. The Alpaca dataset, originally generated using OpenAI's text-davinci-003 model, consists of multi-turn instruction-response pairs designed to fine-tune language models toward instruction-following behavior. The dataset is structured in a JSON format and includes four primary fields: instruction, input, output, and text.

As shown in Fig. 17, the instruction field specifies the task the model should perform, while the input field provides optional context or user queries. The output field contains the model's expected response, and the text field aggregates the full interaction for training purposes. This format helps ensure that the fine-tuned model can understand tasks clearly and generate relevant and structured outputs. The Alpaca-style formatting is thus adopted to convert data from the MITRE ATT&CK framework into training samples suitable for Gemma fine-tuning.


```
{
    "instruction": "Create a classification task by clustering the given list of items.",
    "input": "Apples, oranges, bananas, strawberries, pineapples",
    "output": "Class 1: Apples, Oranges\nClass 2: Bananas, Strawberries\nClass 3:
Pineapples",
    "text": "Below is an instruction that describes a task, paired with an input that
provides further context. Write a response that appropriately completes the
request.\n\n### Instruction:\nCreate a classification task by clustering the given list of
items.\n\n### Input:\nApples, oranges, bananas, strawberries, pineapples\n\n###
Response:\nClass 1: Apples, Oranges\nClass 2: Bananas, Strawberries\nClass 3:
Pineapples",
}
```


Figure 17: Schematic of Alpaca format

3.7 Technique Analyzer

Given the complex structure and diversity of data types in the malware samples and ATT&CK-based datasets used in this study, proper data preprocessing is essential. The collected data includes rich textual descriptions of behaviors, techniques, and indicators, all of which must be standardized and transformed to match the input requirements of large language models. By applying prompt engineering and aligning the data with the Alpaca format, the study ensures compatibility with the Gemma model's fine-tuning pipeline. These preprocessing steps enable LLM to understand cybersecurity-specific contexts and respond accurately to queries about malware behavior and ATT&CK techniques.

3.7.1 Large Language Model

Large language models (LLMs) are pre-trained on extensive textual corpora, enabling them to capture intricate patterns in natural language, including semantic relationships and contextual nuances. When fine-tuned on domain-specific data, these models can generalize effectively and provide accurate, context-aware responses to queries within that domain.

In this study, after preprocessing and formatting the data using prompt engineering and Alpaca-compatible structures, the fine-tuning process is applied to adapt the model for the detection and interpretation of malware behavior based on the MITRE ATT&CK framework.

3.7.2 Model Selection

In this study, the Gemma 2B large language model, developed by Google, was selected as the foundation for model fine-tuning and served as the predictive engine for detecting malware behaviors and analyzing attack techniques. Fine-tuning enables large language models to adapt to specialized domains by leveraging relevant datasets, thereby improving their task-specific performance with constrained computational resources.

Gemma 2 [30], released in 2024 by Google DeepMind, is an open-weight large language model designed for responsible AI development and efficient deployment. It is trained in a high-quality, curated corpus with alignment for safe and useful output. The Gemma family emphasizes both model performance and deployment efficiency, making it particularly suitable for academic research or lightweight inference environments.

To accommodate hardware limitations—specifically the availability of a single NVIDIA RTX 2080Ti GPU—this study adopted the Gemma 2B variant. Despite its compact size, Gemma 2B demonstrates competitive performance on a wide range of NLP benchmarks. It supports a context window of 8192 tokens, which is sufficient for most long-context reasoning and sequence classification tasks. Furthermore, Gemma's compatibility with lightweight fine-tuning techniques such as LoRA (Low-Rank Adaptation) allows it to be effectively customized on consumer-grade hardware without excessive memory overhead.

Gemma 2 comes in several variants:

- Gemma 2B: Optimized for single-GPU environments and prototyping under resource constraints.
- Gemma 9B: Provides stronger performance for larger-scale inference tasks.
- Gemma 27B: Targets competitive performance in open-weight LLM evaluations while maintaining cost-efficiency.

Although larger models like Gemma 2B may achieve higher benchmark scores, their deployment requires substantial hardware resources. In contrast, Gemma 2B offers a practical trade-off between computational feasibility and predictive performance, making it a compelling choice for this research scenario.

Therefore, Gemma 2B was ultimately selected as the final predictive model in this study. It aligns with the project's resource limitations while maintaining reliable capability for malware behavior analysis and attack technique identification.

Benchmark evaluations indicate that Gemma 2B achieves the following scores, as summarized in Table 7, which compares different Gemma model variants across major NLP tasks including MMLU, HumanEval, GSM-8K, and MATH:

Table 7: Gemma model capabilities evaluation

Model	MMLU	HumanEval	GSM-8K	MATH
Gemma 2B	56.1	36.6	68.6	36.6
Gemma 9B	71.3	40.2	68.6	36.6
Gemma 27B	75.2	51.8	74.0	42.3

4 Implementation and Result Analysis

This chapter introduces the system architecture of the design. Section 4.1 gives an overview of the system architecture and introduces the modules in each subsection. Section 4.1.1 describes the data collection module, Section 4.1.2 introduces the data analysis module, Section 4.1.3 describes the behavioral analysis module, Section 4.1.4 introduces the concept extraction module of the ATT&CK framework, Section 4.1.5 introduces the data preprocessing module, Section 4.1.6 presents the model training, and Section 4.2 introduces the experiments.

4.1 System Architecture

Fig. 18 illustrates the system architecture of this research. Initially, the required malware samples are collected and analyzed to generate behavioral trace reports. These reports are then used to extract behavioral features. This study also gathers information related to data sources, data components, and techniques defined in the MITRE ATT&CK framework, which are utilized as training data for fine-tuning a large language model (LLM). Following data preprocessing and model training, the fine-tuned LLM is employed to detect data sources, data components, and technology combinations within the context of the MITRE ATT&CK framework.

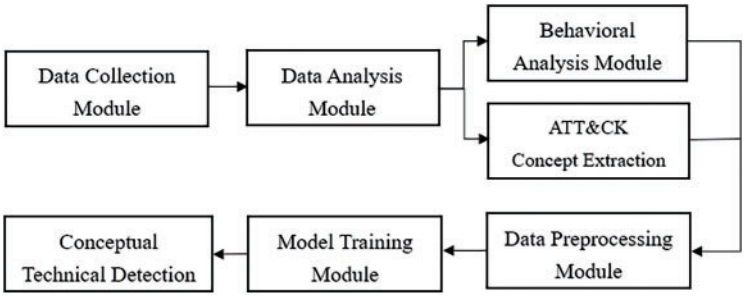

Figure 18: System architecture

4.1.1 Data Collection Module

Fig. 19 presents the data collection module. The malware samples for the research are collected based on various platforms, types, and families. The corresponding samples are downloaded from malware databases using the method described in Section 3.2.1, Data Collection Methods.

The Data Collection Module includes the following features:

- Condition-Specific Malware Search: Requires input of search condition and specified malware file type. Based on the criteria, it searches the relevant malware database. If matches are found, the function returns the SHA256 hash values.
- Specific Malware Download: Requires SHA256 hash values and download path. The specified malware is then downloaded from the relevant database to the designated path.

4.1.2 Data Analysis Module

Fig. 20 depicts the data analysis module. The analysis process is conducted in three stages: malware sample format confirmation, sample execution and logging, and behavior trajectory report download.

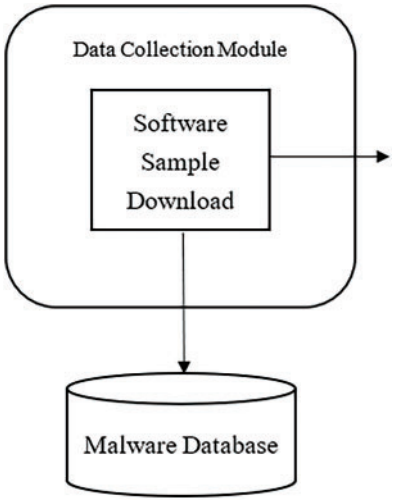

Figure 19: Data collection module

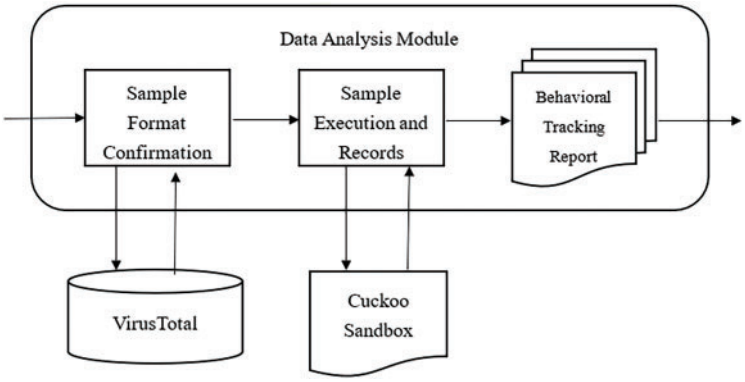

Figure 20: Data analysis module

In the first phase, the collected malware samples are verified using VirusTotal to confirm their compatibility with the Cuckoo Sandbox environment. This ensures proper execution. The process is detailed in Section 3.3.1 Data Normalization.

In the second stage, the malware is uploaded to the Cuckoo sandbox for dynamic analysis to record all the malicious attack behaviors during the execution.

The third stage is to download the behavioral trajectory report of the sample after the sample has been executed for subsequent behavioral analysis.

The Data Analysis Module consists of two parts: Sample Format Confirmation, Sample Execution and Recording. The following is a brief introduction of the functions of each part.

- Sample Format Confirmation includes the following features:
  a. Input the malware to be analyzed.
  b. Input the hash value of the malware to be analyzed. By using the API service provided by VirusTotal to obtain the analysis report of the malware, we can further obtain information about the file type and instruction set architecture in the analysis report. This function will return the malware's file type and instruction set architecture information.

- Sample Execution and Logging includes the following features:
  a. Execute the malware and retrieve information about its file type and instruction set architecture.
  b. The malware sample is submitted for analysis by specifying its file path through the API interface provided by Cuckoo Sandbox. This triggers an automated dynamic analysis process to monitor and record the sample's behavior during execution.
  c. By inputting the specified analysis ID and defining the log download path, the Cuckoo Sandbox API is utilized to retrieve and store the execution log corresponding to the given analysis. This execution log, saved at the designated path, serves as the behavioral trace report of the analyzed malware sample.

4.1.3 Behavioral Analysis Module

Fig. 21 shows the behavior analysis module of the system. The analysis process is divided into two stages: behavior feature extraction and behavior feature analysis.

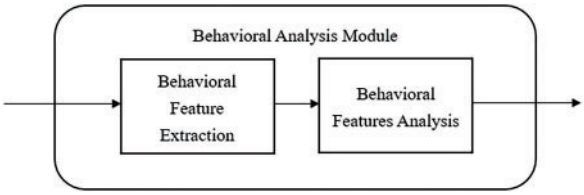

Figure 21: Behavioral analysis module

The first stage involves extracting behavioral features from the behavioral trajectory report by identifying fields that reflect the software's behavior during execution. These extracted fields are compiled into a behavioral feature set. The extraction method is described in Section 3.4.1 Feature Extraction.

The second stage focuses on analyzing the frequency and attack patterns within the behavioral feature set. By evaluating samples from a specific malware family, common attack behaviors can be identified. The analysis method is outlined in Section 3.4.2, Behavior Feature Analyzer.

The Behavior Analysis Module comprises two components: Behavioral Feature Extraction and Behavioral Feature Analysis. Each component is described as follows:

- Behavioral Feature Extraction: This function requires the file location of the incoming behavioral track report. It reads the calls field from the processes subfield of the behavior section, further extracts the api and argument subfields of the calls field. This information is compiled and returned as a collection of behavioral trace reports.
- Behavioral Features Analysis: This function takes as input a collection of behavioral tracks and the corresponding sample storage path. Based on the behavioral characteristics, it analyzes repeated behavioral patterns, calculates their frequencies, and consolidates the findings into a summarized representation. The resulting data is stored in CSV format at the specified path.

To formalize the behavioral analysis, we define the extracted runtime features as a system call set $B(m_i)$, where each call is structured as a tuple containing API, arguments, and return values. These features are aggregated across processes and used to construct a normalized behavior vector v_i, representing the observable behavioral profile of each sample. This model enables quantification and pattern matching in later stages of semantic analysis.

Given a malware sample $m_i \in \mathcal{M}$, we execute it within a sandbox environment to capture its runtime behavior. Let:

- $P_i = \{p_{i1}, p_{i2}, \ldots, p_{ik}\}$: processes spawned during execution
- $S(p_{ij}) = \{s_1, s_2, \ldots, s_j\}$: system calls invoked by process p_{ij}

We define the behavioral trace set of the sample mim_imi as:

$$B(m_i) = \bigcup_{j=1}^{k} S(p_{ij})$$

Each system call $s \in B(m_i)$ is a tuple:

$$s = (api, args, return\_value)$$

From this, we construct the raw behavior vector $v_i \in \mathbb{R}^d$ as:

$$v_i = [f_{s_1}(m_i), f_{s_2}(m_i), \ldots, f_{s_j}(m_i)]$$

where $f_{s_j}(m_i)$ is the normalized frequency of system call s_j in sample m_i.

4.1.4 ATT&CK Concept Extraction

Fig. 22 illustrates the ATT&CK framework information extraction module. The extraction process consists of two phases: the extraction of the conceptual definitions and the extraction of the TTP technical keywords from the MITRE ATT&CK framework.

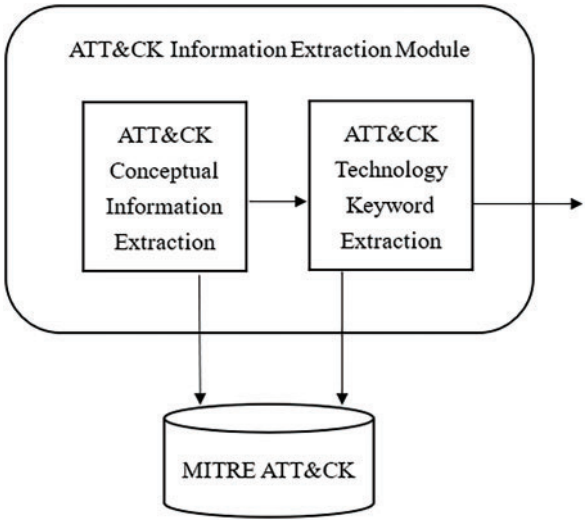

Figure 22: ATT&CK information extraction module.

In the first stage, the module extracts concepts related to data sources and data components as defined in the MITRE ATT&CK framework. The extraction method is described in Section 3.5.1, Conceptual Definition Mapping Method.

In the second stage, the module extracts technical keywords associated with the Tactics, Techniques, and Procedures (TTPs) defined in the MITRE ATT&CK framework. The extraction method is explained in Section 3.5.2, TTP Technical Keyword Mapping Method.

ATT&CK Framework Information Extraction Module comprises two subcomponents: ATT&CK Conceptual Information Extraction and ATT&CK Technical Keyword Extraction. Each is described below:

- ATT&CK Conceptual Information Extraction:
  a. Data Source Concept Extraction: Requires the MITRE ATT&CK STIX file and file storage path. STIX (Structured Threat Information Expression) is a standard for sharing Cyber Threat Intelligence (CTI), and the MITRE team formatted the MITRE ATT&CK Framework's STIX for users to analyze. This function extracts the names and descriptions of the data sources from the STIX file and saves these results in a CSV format.
  b. Data Component Concept Extraction: Requires the STIX file and a storage path. It extracts data component names, descriptions, and associated data sources, saving them as a CSV file.
  c. TTP Technical Concept Extraction: Also requires the STIX file and a storage path, this function retrieves the names, identifiers, and descriptions of TTP techniques, saving the information in CSV format.
- ATT&CK Technical Keyword Extraction:
  a. ATT&CK Technology Keyword Extraction: This function parses TTP descriptions from the STIX file to extract highlighted textual information and maps it to corresponding technique identifiers, storing the data in CSV format.
  b. ATT&CK Data Component Detection and Extraction: Extracts detectable data elements and their corresponding detection methods from the STIX file. The information is mapped to relevant techniques and saved in CSV format.
  c. Technology Keyword Classification: Based on the extracted TTP technical keyword mapping table, this function classifies keywords by data source, data component, platform, and type, and stores the categorized data in CSV format.

In this stage, each observed system call or behavioral keyword is semantically mapped to the corresponding MITRE ATT&CK techniques via the TTP Technical Keyword Mapping Table. Formally, a mapping function $\Phi_{keyword}(s)$ is defined to associate each keyword k extracted from the behavior trace with a subset of techniques, $T_j \in \mathcal{T}$. The resulting set $\mathcal{T}(m_i)$ serves as the preliminary semantic labeling for fine-tuning and inference. To convert low-level behavior into high-level threat knowledge, we introduce two structured mapping tables:

- TTP Technical Keyword Mapping Table: Maps extracted low-level indicators (e.g., execve, LD_PRELOAD, bash) to potential ATT&CK techniques. Each entry:

$$k \in \mathcal{K} \rightarrow T_j \in \mathcal{T}$$

where:

– $\mathcal{K}$: set of observed keywords from logs
– $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$: set of ATT&CK techniques

We define a mapping function:

$$\Phi_{keyword}(s) = \{T_j \in \mathcal{T} \mid k \in s \; and \; \mathcal{M}_K(k) = T_j\}$$

- Conceptual Definition Mapping Table: Maps high-level concepts and definitions from ATT&CK (e.g., Command and Scripting Interpreter, Shared Object Injection) to structured labels. These are used to train the LLM in understanding semantic meaning and context. Each entry includes:
  - Technique ID T_j
  - Concept name
  - Associated tactic
  - Descriptive text

  This table is used to construct prompt-response pairs for supervised fine-tuning of the LLM.
- Unified Technique Mapping for Sample m_i: Aggregated Semantic Associations

$$T(m_i) = \bigcup_{s \in B(m_i)} \Phi_{keyword}(s)$$

This yields the initial candidate ATT&CK techniques corresponding to the system-level behavior of the sample.

The Conceptual Definition Mapping Table complements this process by providing a structured semantic reference for each ATT&CK technique, which is later utilized in constructing high-quality prompt-response training pairs for the language model.

4.1.5 Data Preprocessing Module

Fig. 23 presents the data preprocessing module of the system, which comprises two primary stages: Prompt Engineering and Data Format Conversion.

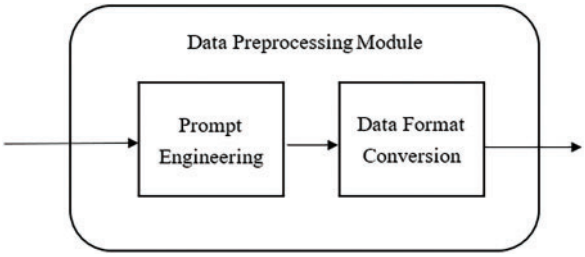

Figure 23: Data preprocessing module.

The first stage involves constructing prompts based on the conceptual elements defined in the MITRE ATT&CK framework, as outlined in Section 3.6.1, Prompt Engineering. In the second stage, the conceptual information and TTP technical keyword mappings are transformed into the Alpaca format, suitable for fine-tuning the Gemma model, as detailed in Section 3.6.2 Data Formatter for Gemma.

The Data Preprocessing Module is structured into two functional components—Prompt Engineering and Data Format Conversion. The specific functions of each component are elaborated in the following sections.

- Prompt Engineering contains the following functions:
  a. Concept Prompt Statement Composition: This function requires the input of all concept sets extracted from the ATT&CK framework. System prompts, user input, and model response statements are designed and optimized for each concept defined within the framework. This process generates an optimized set of concept prompt statements.

b. TTP Keyword Prompt Statement Composition: This function takes the ATT&CK Framework's TTP Technical Keyword Mapping Table as input. Based on this input, the system designs and optimizes the corresponding prompts, user input, and model response related to TTP technical keywords. The output is a refined set of conceptual prompt statements.
c. ATT&CK Data Component Detection Prompt Statement Composition: This function requires the Data Component Detection Technology Mapping Table from the ATT&CK framework. It constructs the TTP techniques and their corresponding detection methods. The output is an optimized collection of conceptual prompt statements for detection purposes.

- Data format conversion includes the following functions:
  a. Gemma Data Format Conversion: This function takes the optimized set of concept prompt statements and a specified dataset storage path as input. It converts the designed and optimized prompt-response pairs into the format required by the Alpaca dataset for training the Gemma model. The converted data is then stored in JSON format at the designed dataset storage location.

4.1.6 Model Training Module

The model training module of this study will fine-tune the model for the corresponding large language model to detect the concepts of the MITER ATT&CK framework. Based on the behavior-to-technique mapping, we construct instruction-response pairs (x_i, y_i), where x_i represents the prompt constructed from observed behavior, and y_i corresponds to one or more MITRE ATT&CK technique identifiers. The fine-tuning objective is to optimize the model parameters θ of Gemma 2B using a cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{(x_i, y_i) \in D_{train}} logP(y_i \mid x_i; \theta)$$

To improve training efficiency and memory usage, we adopt the LoRA (Low-Rank Adaptation) method to inject trainable low-rank matrices into the attention layers of the model. Once fine-tuned, the model can generalize behavior traces into ATT&CK techniques even when unseen combinations or obfuscated commands appear.

To generalize behavior beyond static mappings and enable semantic inference, we employ a large language model (Gemma 2B) fine-tuned via prompt engineering using both mapping tables.

- Training Data Generation:

  For each malware sample m_i, we generate a prompt:

$$x_i = Prompt\left(B\left(m_i\right)\right) = \text{“}Given\ these\ behaviors: [syscalls, keywords], what\ ATT\&CK\ techniques\ apply?\text{”}$$

Target label:

$$y_i = \mathcal{T}\left(m_i\right)$$

The training dataset:

$$D_{train} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

- Fine-Tuning via LoRA (Low-Rank Adaptation):

The LLM learns the mapping $\mathcal{F}_\theta: x \mapsto y$ by minimizing cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{(x_i, y_i) \in D_{train}} logP(y_i \mid x_i; \theta)$$

Fine-tuning uses LoRA modules $A, B \in \mathbb{R}^{d \times r}$ such that:

$$W' = W + \alpha AB$$

where W is the frozen base weight and α is a scaling factor.

- Inference:

At inference time, for an unknown malware sample m^*, we extract $B(m^*)$, construct a semantic prompt x^*, and obtain:

$$\hat{y}^* = \mathcal{F}_\theta(x^*) = Predicted\ ATT\&CK\ techniques$$

During inference, the system constructs a prompt x^* from newly observed behavior and queries the fine-tuned LLM $\mathcal{F}_\theta$ to predict the corresponding set of techniques $\hat{y}^*$, effectively converting low-level execution details into high-level threat intelligence.

4.2 Experiment

4.2.1 Experimental Model

This chapter describes the model used in the experiment, including the model creation process, model fine-tuning process, and model-related settings, such as model initial settings and model hyperparameter settings.

This study chose to use the Gemma 2B large language model for model fine-tuning to detect concept detection in the MITER ATT&CK framework.

4.2.2 Gemma 2B

In this study, the experimental environment is centered around fine-tuning the Gemma 2B large language model using parameter-efficient fine-tuning (PEFT) via LoRA (Low-Rank Adaptation) on a local workstation equipped with an NVIDIA RTX 2080 Ti GPU, as illustrated in Fig. 24. Compared to cloud-based platforms such as Google Colab, a dedicated GPU environment offers full control over hardware allocation, removes runtime restrictions, and supports longer and more stable training sessions for deep learning experiments.

To improve training efficiency and reduce GPU memory consumption, this experiment adopts the open-source PEFT framework with LoRA. LoRA enables the training of a minimal subset of model parameters by injecting trainable low-rank matrices into selected components of the transformer's architecture. This method is particularly suitable for constrained environments, such as single-GPU workstations, and maintains high model performance while minimizing memory overhead.

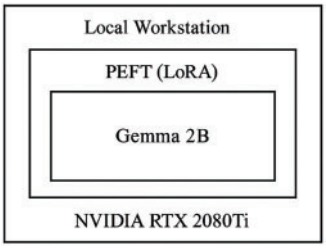

Figure 24: Gemma 2B experimental environment architecture diagram (source: this study)

The fine-tuning pipeline begins by loading the "google/gemma-2b" tokenizer and base model from Hugging Face. The model is converted to torch.float16 precision and transferred to the GPU to optimize memory usage. A structured dataset in Alpaca-style format is used for instruction tuning, comprising three fields: instruction, input, and output, which describe malware behavior prompts and their corresponding MITRE ATT&CK technique labels.

Fine-tuning settings are defined using a LoraConfig object, with parameters carefully aligned to the architectural components of the Gemma 2B model, as illustrated in Fig. 25. This configuration supports parameter-efficient fine-tuning under constrained GPU resources by injecting low-rank matrices into specific transformer layers.

```
# Apply LoRA configuration
lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    # Select target modules according to Gemma architecture
    target_modules=["q_proj", "k_proj",
                    "v_proj", "o_proj",
                    "gate_proj", "up_proj",
                    "down_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type=TaskType.CAUSAL_LM
)
model = get_peft_model(base_model, lora_config)
```

Figure 25: Gemma 2B fine-tuning model settings (source: this study)

The following configuration parameters were applied:

- r = 8: rank of the low-rank adaptation matrices
- lora_alpha = 32: scaling factor applied to the LoRA output
- lora_dropout = 0.05: dropout applied within LoRA layers
- task_type = TaskType.CAUSAL_LM: specifies a causal language modeling objective
- target_modules = ["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]: modules targeted for LoRA injection

Python configuration of LoRA parameters applied to Gemma 2B.

These target modules encompass both the core components of the attention mechanism (q_proj, k_proj, v_proj, o_proj) and the feed-forward network (gate_proj, up_proj, down_proj). This configuration follows

best practices in LoRA-based fine-tuning and provides high adaptability while only updating a fraction of the total model parameters. It is especially effective for transformer architectures like Gemma 2B, which follows a Gemma-style design.

These settings are passed to get_peft_model (base_model, lora_config), which wraps the base model with LoRA adapters, enabling efficient fine-tuning without modifying the pre-trained weights—a critical advantage for running large models on limited hardware such as the RTX 2080Ti.

Training arguments are then specified using the TrainingArguments class, as illustrated in Fig. 26, which outlines the full configuration used during instruction tuning. These settings are optimized for memory-efficient training while maintaining stable convergence under constrained GPU capacity.

```
# Trainer configuration
training_args = TrainingArguments(
    output_dir="gemma_lora_output",
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    num_train_epochs=4,
    learning_rate=2e-4,
    fp16=True,
    save_strategy="epoch",
    logging_steps=200,
    remove_unused_columns=False
)
```

Figure 26: Training parameters (source: this study)

The configuration parameters include:

- output_dir = “gemma_lora_output”
- per_device_train_batch_size = 1
- gradient_accumulation_steps = 4
- num_train_epochs = 4
- learning_rate = 2×10^{-4}
- fp16 = True (for mixed precision training)
- logging_steps = 100

Python script used to define training arguments in the Hugging Face Trainer.

The Hugging Face Trainer is then used to launch the training process. Loss values are monitored at regular intervals to assess convergence. For example, by step 800, the training loss typically converges to approximately 0.3621, indicating effective learning of behavior-to-technique mappings from the instruction-tuning dataset.

After the training phase, the model transitions into inference mode. For each test example, the behavioral input prompt is formatted and tokenized before being passed into the generate() function of the language model. This function outputs a sequence of tokens, which are then decoded using the decode() method to produce human-readable text.

The decoded predictions are further processed to extract the predicted MITRE ATT&CK technique IDs. Similarly, the ground truth labels are parsed from the expected output. To handle multi-label classification, both the predicted and true techniques are converted into binary vectors using a predefined set of technique labels.

If no technique is detected in either the prediction or the ground truth, a zero vector is used to represent the absence of TTPs. The comparison between these vectors allows for precise evaluation of the model's ability to infer multiple ATT&CK techniques from a single Linux command or behavioral input.

4.3 Experimental Results

4.3.1 Evaluation Indicators

This study uses F1-Score as an indicator to evaluate model performance. The calculation method of F1-Score is shown in Formula (3).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

F1-Score can be calculated by weighting precision and recall, where precision is the ratio of the predicted category to the actual category, and the calculation method is shown in Formula (1). Recall is the ratio of the correct prediction of the actual category, and the calculation method is shown in Formula (2). TP (True Positive) is a true positive, which means that the prediction is true and the actual is also true. FP (False Positive) is a false positive, which means that the prediction is true, but the actual result is not true. FN (False Negative) is a false negative, which means that the judgment is false, but the actual is true.

4.3.2 Model Evaluation

The experimental results of fine-tuning the MITRE ATT&CK framework concept for the fine-tuned Gemma 2B model are shown in Table 8. The F1-Score of using the fine-tuned Gemma 2B to detect the data sources, data components, and technology combinations defined by the MITRE ATT&CK framework is 86%.

Table 8: Evaluation metrics of the fine-tuned Gemma model for TTP detection

	Fine Tuned Gemma 2B
F1-Score	0.8633
Precision	0.8601
Recall	0.8950

There is no charge for using the Gemma model provided by the Meta AI team. In contrast, the Gemma model requires its own experimental environment. The Gemma model is quite large in scale and parameters, and requires high-performance computing equipment to ensure the normal operation of the model. At the same time, a large amount of storage space is required to save the model weights and training data. Therefore, the requirements for system equipment to establish the Gemma model are quite high.

Therefore, although this study uses the Gemma 2B model, it is also necessary to consider factors such as model usage fees and research experimental environment to obtain the best research results for the research topic.

5 Conclusion and Future Work

This study presents a Linux-specific malware detection framework that combines dynamic behavioral analysis with the semantic reasoning capabilities of large language models (LLMs), fine-tuned using the MITRE ATT&CK framework. By executing malware samples in sandboxed environments, behavioral features such as system calls and command-line interactions are extracted and mapped to ATT&CK-defined data sources, components, and Tactics, Techniques, and Procedures (TTPs). This mapping is supported by two key constructs—Conceptual Definition Mapping and TTP Technical Keyword Mapping—derived from official MITRE documentation. These enable the LLM to semantically interpret behavioral patterns and associate them with corresponding attack techniques.

The experimental results confirm that the fine-tuned Gemma 2B model significantly enhances the precision and recall of technique-level detection, especially for malware variants with ambiguous or previously unclassified behaviors. By uncovering consistent attack patterns across malware families, the framework not only identifies malicious intentions and targets but also improves interpretability and situational awareness in Linux environments. This semantic understanding bridges raw behavioral data and structured threat intelligence, contributing to a more explainable and scalable detection pipeline.

Looking ahead, we plan to extend this detection process to other platforms (e.g., Windows, Android) by adapting behavior extraction pipelines and enriching the TTP mapping with platform-specific intelligence sources. These efforts aim to build a scalable, explainable, and cross-platform malware detection system driven by behavioral semantics and threat intelligence integration.

Acknowledgement: The authors are grateful to all the editors and anonymous reviewers for their comments and suggestions.

Funding Statement: This work was supported by the National Science and Technology Council under grant number 113-2221-E-027-126-MY3.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and methodology, Ping-Feng Wang; funding acquisition and supervision, Jong-Yih Kuo; writing—original draft preparation, Ti-Feng Hsieh; writing—review and editing, Cheng-Hsuan Kuo. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the corresponding author, Ping-Feng Wang, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Fortinet. 1H 2023 global threat landscape report [Internet]. [cited 2023 Dec 5]. Available from: https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/threat-report-1h-2023.pdf.
2. Trend Micro. 2022 annual cybersecurity report [Internet]. [cited 2023 Dec 5]. Available from: https://www.trendmicro.com/vinfo/tw/security/research-and-analysis/threat-reports/roundup/rethinking-tactics-annual-cybersecurity-roundup-2022.
3. Trend Micro. 2023 mid-year cybersecurity report [Internet]. [cited 2023 Dec 5]. Available from: https://www.trendmicro.com/content/dam/trendmicro/global/zh_tw/security-intelligence/threat-report/report/2023%20mid-year%20roundup%20full%20report_Final.pdf.
4. Cuckoo sandbox [Internet]. [cited 2023 Dec 5]. Available from: https://cuckoosandbox.org/.

5. MITRE. MITRE ATT&CK framework [Internet]. [cited 2023 Dec 5]. Available from: https://attack.mitre.org/.
6. MITRE. MITRE ATT&CK software [Internet]. [cited 2023 Dec 5]. Available from: https://attack.mitre.org/software/.
7. Google DeepMind. Gemma [Internet]. [cited 2024 Jun 2]. Available from: https://ai.google.dev/gemma?hl=zh-tw.
8. Hu E, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, et al. LoRA: low-rank adaptation of large language models. arXiv:2106.09685. 2021.
9. Catak FO, Yazi AF. A benchmark API call dataset for windows PE malware classification. arXiv:1905.01999. 2021.
10. Chierzi V, Mercês F. Evolution of IoT linux malware: a MITRE ATT&CK TTP based approach, Virtual. In: APWG Symposium on Electronic Crime Research; 2021 Dec 1–3; Virtual. Piscataway, NJ, USA: IEEE; 2021. p. 1–11.
11. Ahn G, Kim K, Park W, Shin D. Malicious file detection method using machine learning and interworking with MITRE ATT&CK framework. Appl Sci. 2022;12(21):10761.
12. Sartoli S, Wei Y, Hampton S. Malware classification using recurrence plots and deep neural network. In: 9th IEEE International Conference on Machine Learning and Applications; 2020 Dec 14–17; Miami, FL, USA. p. 901–6.
13. Shiva Darshan SL, Jaidhar CD. Windows malware detector using convolutional neural network based on visualization images. IEEE Trans Emerg Top Comput. 2021;9(2):1057–69. doi:10.1109/tetc.2019.2910086.
14. Xu A, Chen L, Kuang X, Lv H, Yang H, Jiang Y, et al. A hybrid deep learning model for malicious behavior detection. In: IEEE 6th International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, and IEEE International Conference on Intelligent Data and Security; 2020 May 25–27; Baltimore, MD, USA. Piscataway, NJ, USA: IEEE; 2020. p. 55–9.
15. Zhang J. DeepMal: a CNN-LSTM model for malware detection based on dynamic semantic behaviours. In: International Conference on Computer Information and Big Data Applications; 2020 Apr 17–19; Guiyang, China. Piscataway, NJ, USA: IEEE; 2020. p. 313–6.
16. Anandhi V, Vinod P, Menon VG, AK ER, Shilesh A, Viswam A, et al. Malware detection using dynamic analysis. In: International Conference on Advances in Intelligent Computing and Applications; 2023 Feb 1–3; Kochi, India. Piscataway, NJ, USA: IEEE; 2023. p. 1–6.
17. He Y, Geng S, Zhao Y, Feng Y. Research on intelligent detection method of malicious behavior based on self-attention. In: 2nd International Conference on Machine Learning and Computer Application; 2021 Dec 17–19; Shenyang, China. Piscataway, NJ, USA: IEEE; 2021. p. 1–5.
18. Or-Meir O, Cohen A, Elovici Y, Rokach L, Nissim N. Pay attention: improving classification of PE malware using attention mechanisms based on system call analysis, Virtual. In: International Joint Conference on Neural Networks; 2021 Jul 18–22; Piscataway, NJ, USA: IEEE; 2021. p. 1–8.
19. Walker A, Shukla RM, Das T, Sengupta S. Ohana means family: malware family classification using extreme learning machines. In: 19th Annual Consumer Communications & Networking Conference; 2022 Jan 8–11; Las Vegas, NV, USA. Piscataway, NJ, USA: IEEE; 2022. p. 534–42.
20. An BY, Yang JH, Kim S, Kim T. Malware detection using dual siamese network model. Comput Model Eng Sci. 2024;141(1):563–84. doi:10.32604/cmes.2024.052403.
21. Sekharan CN, Vuppala PS. Fine-tuned large language models for improved clickbait title detection. In: Congress in Computer Science, Computer Engineering, & Applied Computing; 2023 Jul 24–27; Las Vegas, NV, USA. Piscataway, NJ, USA: IEEE; 2023. p. 215–20.
22. Li L, Gong B. Prompting large language models for malicious webpage detection. In: 4th International Conference on Pattern Recognition and Machine Learning; 2023 Aug 4–6; Urumqi, China. Piscataway, NJ, USA: IEEE; 2023. p. 393–400.
23. OpenAI [Internet]. [cited 2024 Jun 2]. Available from: https://openai.com/.
24. Li H, Shan L. LLM-based vulnerability detection. In: International Conference on Human-Centered Cognitive Systems; 2023 Dec 16–17; Cardiff, UK. Piscataway, NJ, USA: IEEE; 2023. p. 1–4.
25. VirusShare [Internet]. [cited 2023 Dec 5]. Available from: https://virusshare.com/.
26. MalShare [Internet]. [cited 2023 Dec 5]. Available from: https://malshare.com/.

27. abuse.ch. MalwareBazaar [Internet]. [cited 2023 Dec 5]. Available from: https://bazaar.abuse.ch/.
28. vx-underground [Internet]. [cited 2023 Dec 5]. Available from: https://vx-underground.org/.
29. MITRE. ATT&CK data sources [Internet]. [cited 2023 Dec 5]. Available from: https://github.com/mitre-attack/attack-datasources.
30. HuggingFace. Gemma 2 [Internet]. [cited 2024 Jun 2]. Available from: https://huggingface.co/google/gemma-2b.