Journal of Information Security and Applications 93 (2025) 104162

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

DroidTTP: Mapping android applications with TTP for Cyber Threat Intelligence

Dincy R. Arikkat [a], Vinod P. [b,a], Rafidha Rehiman K.A. [a], Serena Nicolazzo [d], Marco Arazzi [c], Antonino Nocera [c,*], Mauro Conti [b]

[a] Department of Computer Applications, Cochin University of Science and Technology, Kerala, India
[b] Department of Mathematics, University of Padua, Padua, Italy
[c] Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Pavia, Italy
[d] Department of Computer Science, University of Milan, Milan, Italy

ARTICLE INFO

Keywords:
Cyber Threat Intelligence
Tactic technique and procedure
Feature selection
Large Language Model
Retrieval augmented generation

ABSTRACT

The widespread use of Android devices for sensitive operations has made them prime targets for sophisticated cyber threats, including Advanced Persistent Threats (APT). Traditional malware detection methods focus primarily on malware classification, often failing to reveal the Tactics, Techniques, and Procedures (TTPs) used by attackers. To address this issue, we propose DroidTTP, a novel system for mapping Android malware to attack behaviors. We curated a dataset linking Android applications to Tactics and Techniques and developed an automated mapping approach using the Problem Transformation Approach and Large Language Models (LLMs). Our pipeline includes dataset construction, feature selection, data augmentation, model training, and explainability via SHAP. Furthermore, we explored the use of LLMs for TTP prediction using both Retrieval Augmented Generation and fine-tuning strategies. The Label Powerset XGBoost model achieved the best performance, with Jaccard Similarity scores of 0.9893 for Tactic classification and 0.9753 for Technique classification. The fine-tuned LLaMa model also performed competitively, achieving 0.9583 for Tactics and 0.9348 for Techniques. Although XGBoost slightly outperformed LLMs, the narrow performance gap highlights the potential of LLM-based approaches for Tactic and Technique prediction.

1. Introduction

In recent years, the digital landscape has undergone significant transformations, with critical assets such as financial data and personal information increasingly shifting to online platforms [1]. The ubiquity of mobile and IoT devices in daily life has primarily driven this shift. Unlike traditional Personal Computers (PCs), these devices not only facilitate daily tasks but also handle a vast amount of sensitive data essential for activities ranging from banking [2] to online shopping and entertainment [3]. By 2023, Android had solidified its position as the world's leading mobile operating system, capturing over 70% of the global market share and surpassing its competitors.[1] Its popularity comes from its open-source, cost-effective, and user-friendly nature [4], factors that have contributed to the proliferation of over 2.61 billion applications[2] available through the Google Play Store.

However, this widespread adoption has also made Android devices attractive targets for cybercriminals, particularly Advanced Persistent Threat (APT) groups [5]. The escalation of smartphones has allowed users to make mobile payments and store sensitive information such as login credentials, increasing the risks for both individuals and organizations. As a result, the past decade has seen a significant rise in mobile attacks, introducing a range of sophisticated threats that specifically target mobile platforms [6]. The nature of threats emerging from mobile devices often parallels those affecting traditional PCs. These threats are not limited to conventional malware like worms, Trojans, and viruses; they extend to sophisticated cyber attacks that can compromise security and privacy or even gain complete control over a device. Also, mobile attacks spread malicious content rapidly through technologies such as 5G and Wi-Fi, which provide seamless Internet connectivity. This connectivity renders mobile devices particularly attractive to cyber criminals, who can deploy a variety of attack vectors, from exploiting mobile sensors to executing application-level malware. As such, understanding and mitigating these attacks is

* Corresponding author.
E-mail address: antonino.nocera@unipv.it (A. Nocera).

[1] https://www.statista.com/statistics/921152.
[2] https://www.businessofapps.com/data/google-play-statistics.

https://doi.org/10.1016/j.jisa.2025.104162

Available online 14 July 2025

crucial in safeguarding both personal and organizational assets from the diverse array of threats targeting mobile platforms.

Existing studies mainly focused on mobile malware detection and classification to address various security challenges [7,8]. However, there is a notable lack of research on cyber attribution, which aims to track attack Tactics and Techniques employed by an attacker. Current Machine Learning (ML) methods typically offer classification for mobile users and app security analysts, indicating whether an application is likely to be malicious or benign. This approach falls short of fully addressing the complexities of malware detection. For example, suppose that a mobile application is found to exhibit suspicious behavior, such as accessing sensitive user data without permission. Simply labeling this app as malicious or belonging to a malicious family, does not explain how it conducts its attack. Understanding the specific Tactics, like exploiting a vulnerability in the operating system or using social engineering to trick users, can provide deeper insights into the threat and inform better defensive strategies. Mapping Tactics, Techniques, and Procedures (TTPs) to malware behavior would empower security professionals to identify threats and understand the attacker's motives and methods.

The MITRE ATT&CK[3] framework provides a structured overview of TTPs associated with attacks. It serves as a valuable resource for security professionals, providing a common language and detailed documentation that helps to understand attacker behavior. By categorizing TTPs, the framework enables security analysts to better identify, analyze, and respond to threats in a systematic way. Moreover, the Android Matrix[4] is a variant of the ATT&CK framework tailored for mobile threats involving Android devices. It breaks down the typical steps an attacker might follow when targeting mobile systems into Tactics and Techniques. However, there is currently a lack of automated methods to correlate Android applications with particular Tactics and Techniques. As mobile malware becomes increasingly sophisticated, attackers are constantly evolving their tactics, making it essential for security analysts to have tools that can automatically map application behaviors to TTPs. To bridge this gap, we propose *DroidTTP*, a system designed to identify the TTPs associated with Android malware.

To build the DroidTTP framework, we have created a new dataset to train automated models that map TTPs to Android applications. Subsequently, we implemented a multilabel Tactic and Technique classification model to identify the TTPs associated with these applications (apps). Our methodology leverages Problem Transformation Approaches (PTA), particularly useful in multi-label classification, and ML Techniques to enhance TTP prediction. Beyond traditional models, we explore the potential of Large Language Models (LLMs) by implementing Retrieval-Augmented Generation (RAG) with prompt engineering for dynamic and context-aware predictions. In particular, we aimed to evaluate the effectiveness of LLMs in Android malware detection by analyzing features extracted from application components and mapping them to attacker behaviors. This approach seeks to determine whether LLMs can infer malicious intent based on patterns of permissions, activities, receivers, services, and inter-component communications, thereby enhancing behavioral analysis in malware classification. Additionally, we fine-tuned an LLM to evaluate its effectiveness in predicting TTPs based on Android application features. Consequently, we formulate the following research question:

RQ_1: Can DroidTTP effectively infer the adversarial Tactics employed by attackers based on the features derived from the Android applications?

RQ_2: To what extent can DroidTTP accurately identify and predict the attack Techniques used by adversaries through the analysis of attributes from Android applications?

RQ_3: What is the potential of LLMs in predicting adversarial attack behaviors based on the features of Android applications, and how do they compare to traditional Machine Learning models?

The main contributions of our work are as follows.

- We introduce a novel dataset designed explicitly for mapping MITRE TTPs to Android applications. This dataset enables researchers and security professionals to analyze the behavior of various Android apps in relation to established cybersecurity frameworks.
- We propose an enhanced feature selection strategy for multi-label classification that builds upon the *SelectKBest*[5] method, a commonly used univariate feature selection method that identifies the top K features by applying statistical tests like chi-square or ANOVA F-value. Our approach involves a two-step process: first, performing label-specific feature selection to capture the most relevant features for each label, and second, generalizing this selection across multiple datasets to identify robust features.
- We develop an automated system that maps Android applications to different MITRE Tactic labels. This system utilizes advanced multi-label classification algorithms to analyze app behavior and associate it with specific Tactics.
- In addition to Tactic mapping, we implement an automated system that assigns Techniques from the MITRE framework to Android applications.
- We investigate the potential of LLMs for predicting Tactics and Techniques in both Retrieval-Augmented Generation (RAG) and fine-tuning settings. Our study evaluates various LLMs, explores different prompt engineering strategies, and analyzes their effectiveness in enhancing TTP prediction.

The remainder of this paper is organized as follows. Section 2 discusses related works. Section 4 explains the methodology we used to develop DroidTTP. Section 5 presents the experimental evaluation and discussion of our findings. Finally, Section 7 concludes with information and outlines future research directions.

2. Literature review

Recently, numerous studies have focused on Android malware detection, leveraging various techniques such as static features, dynamic features, hybrid features, and visualization analysis [9]. Despite advancements in malware detection, the identification of attack behaviors such as MITRE Tactics and Techniques remains an underexplored area in the current literature. In recent years, few studies have focused on malware capability detection. One such work is MalXCap [10], which introduces a novel approach for detecting capabilities in Windows malware by utilizing dynamic analysis to capture patterns in API call sequences. The authors employed multi-label classification to identify multiple malicious functionalities within a single malware sample and contributed a dataset comprising 12 distinct capabilities extracted from 8000 malware samples. While MalXCap focuses on Windows malware, recent research efforts have also begun exploring capability detection in Android malware. The authors of [11] proposed a technique called A3CM (Automatic Capability Annotation for Android Malware) that can annotate the capabilities of malware using the semantic features extracted from the raw software package. They employed a multi-label classification model to identify and label malicious capabilities and evaluated the method on a new dataset comprising 6899 annotated samples from 72 malware families. A3CM demonstrated accuracy scores of 1.00 for known malware, 0.98 for small malware families, and 0.63 for zero-day malware. In another study [12], Qiu et al. proposed the Multiview Feature Intelligence (MFI) approach to identify security and privacy-related capabilities of Android malware, instead of traditional malware family classification. MFI extracts and integrates semantic string features from source code, structural features from API call graphs, and sequential Smali opcode features to create a comprehensive representation of apps. These combined features are encoded

[3] https://attack.mitre.org/.

[4] https://attack.mitre.org/matrices/mobile/android.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html.

and input into a deep neural network to detect specific malicious activities such as botnet attacks and information theft. Experimental results demonstrated that the multiview feature approach consistently outperforms single-view methods, achieving a 1%–4% increase in accuracy and a 2%–4% improvement in F1 score, including for zero-day malware detection.

Beyond capability detection, several studies have explored the identification of MITRE Tactics and Techniques across diverse data sources, including threat intelligence reports, network traffic logs, Windows systems, and Android applications. While some studies, including TTP-Drill [13] and rcATT [14], focus on extracting TTPs from threat reports, their reliance on term frequency-based methods limits the accuracy and completeness of TTP identification. These methods fail to capture the contextual relationships within CTI reports, thereby limiting their effectiveness. Shin et al. [15] proposed a TTP extraction approach based on GloVe embedding, which uses a co-occurrence matrix to capture semantic relationships. Their results show a correlation of up to 0.96 between the co-occurrence matrix and the embedding performance for each Tactic. Li et al. [16] employed DistilBERT to classify Tactics and Techniques from unstructured threat reports at the sentence level. They enhanced classification accuracy by using a post-processing step that corrects misclassifications based on the hierarchical relationships between Tactics, Techniques, and sub-techniques. In [17], Liu et al. employed an Attention-based Transformer Hierarchical Recurrent Neural Network to extract TTPs from unstructured CTI. Initially, they implemented a Transformer Embedding Architecture to capture high-level semantic representations of both the CTI and the ATT&CK taxonomy. Next, an Attention Recurrent Structure was developed to model the dependencies between tactical and technical labels in ATT&CK. Finally, a joint Hierarchical Classification module was introduced to predict the final TTPs. The UniTTP [18] framework maps unstructured threat reports to MITRE ATT&CK Tactics and Techniques by leveraging SecureBERT for Tactic classification and hierarchical attention mechanisms for enhanced semantic representation. It embeds semantic dependency graphs to model relationships between Tactics and Techniques, and combines outputs from Tactic classification and embedding modules for Technique identification. LLM is further employed to refine and validate the coherence of the mapped TTPs against the original threat narratives.

Recent research also investigates mapping TTPs to network traffic. In [19,20], Sharma et al. introduced an ML-based approach for malware detection in network traffic by utilizing adversarial behavior knowledge represented as TTP. They extracted TTP features from network traffic and integrated them into ML models, demonstrating superior performance compared to state-of-the-art malware detection methods. Vasan et al. [21] developed DEEPCAPA, a method for detecting malicious capabilities in Windows malware by utilizing the MITRE ATT&CK framework. Their work builds upon prior tools such as CAPA,[6] a widely used analysis tool for identifying malware capabilities based on manually crafted rules. In contrast, DEEPCAPA extracts API call sequences from memory snapshots taken during sandboxed executions, capturing both active and dormant code, including packed or runtime-injected components. DEEPCAPA constructs Control Flow Graphs (CFGs) and performs probabilistic random walks to model malware behaviors. These sequences are then analyzed using a Deep Neural Network based on transformer architecture.

In the Android security domain, Fairbanks et al. [22] identified attack Tactics by analyzing control flow graphs in malware samples. Their methodology employed the Inferential SIR-GN model for node representation and utilized Random Forest classification, with SHAP analysis providing insight into feature importance and subgraph relevance. This approach achieved a 92.7% F1-score in Tactic detection. However, the scope of their research was limited to seven specific Tactics and did not extend to the broader range of attack Techniques used by malicious actors. While these studies have made significant advancements in malware detection and capability annotation, few have explored the mapping of attack behaviors, particularly MITRE Tactics and Techniques, to Android applications. To the best of our knowledge, no dataset exists that captures these relationships in detail. To provide a clearer comparison with existing studies, Table 1 summarizes the state of the art related to our study. We compare each study based on the problem addressed, the source/type of data used, whether the methods included Tactic and Technique prediction, and whether LLMs have been employed. As shown in the table, while prior works have made important contributions to malware analysis and Tactic mapping, very few have explored fine-grained mapping of Techniques, particularly within the Android ecosystem. Moreover, the integration of LLMs for TTP prediction remains largely unexplored.

Our research addresses these critical gaps through several key innovations. First, we compiled a benchmark dataset that captures multiple Tactics and Techniques related to Android applications. This dataset serves as a foundation for our multi-label classification system, which maps applications to their corresponding MITRE Tactics and Techniques. This work also introduces an adaptive feature selection framework that optimizes feature sets based on the characteristics of different training datasets, leading to enhanced model performance and broader applicability. To the best of our knowledge, while LLMs have been widely applied in various cybersecurity tasks [23–26], no existing study has specifically explored their use in mapping Android applications to TTPs. In this work, we investigate the potential of LLMs by leveraging RAG and fine-tuning techniques for TTP prediction.

3. Background

In this section, we provide the necessary background information for our study, covering Tactics, Techniques, and Procedures, Adversarial Tactics, Techniques, Common Knowledge (ATT&CK), and Retrieval Augmented Generation.

Table 2 summarizes the acronyms used in this paper.

3.1. Tactics, Techniques, and Procedures (TTPs)

In cybersecurity, understanding the methodologies, intentions, and actions of adversaries is fundamental to building an effective defense framework. An effective way to analyze and classify adversarial activities is by understanding TTPs. TTPs describe the behavior of attackers that provides both a strategic overview and tactical insights into their operations. Although Indicators of Compromise (Io) such as IP addresses, file hashes, or domain names are invaluable for real-time threat detection, they typically lack the context necessary to interpret the intentions and behavior of attackers. TTPs, on the other hand, serve as higher-level IoCs that provide a more comprehensive view of how an attack unfolds. The three components of TTPs are:

- *Tactics*. Represents the high-level *goals or objectives* that an adversary aims to achieve during an attack. In the context of Android security, these objectives could range from data theft and system infiltration to denial of service.
- *Techniques*. Describe how the attacker will perform the Tactic. They are the specific *methods or actions* used to achieve the objective, providing more granular insights into the attacker's approach. In Android security, several Techniques can be employed to execute a particular Tactic. For example, if the attacker's Tactic is to gain access to the system, a common Technique could be *phishing*.
- *Procedures*. Procedures are the exact *steps or actions* that provide a step-by-step description of how the attacker operationalizes their methods to achieve their objectives. For example, consider the procedure steps where the adversary utilizes phishing as a

[6] https://github.com/mandiant/capa.

Table 1
Comparison with state of art.

Ref.	Problem	Source/Type	Tactic prediction	Technique prediction	LLM
Saha et al. [10]	MCI	Windows	×	×	×
Qiu et al. [11]	MCI	Android	×	×	×
Qiu et al. [12]	MCI, MD	Android	×	×	×
TTPDrill [13]	TTM	Threat report	✓	✓	×
rcATT [14]	TAM	Network	✓	×	×
Shin et al. [15]	TTM	Threat report	✓	✓	×
Li et al. [16]	TTM	Threat report	✓	✓	×
Liu et al. [17]	TTM	Threat report	✓	✓	×
UniTTP [18]	TTM	Threat report	✓	✓	✓
Sharma et al. [19,20]	NTA, TTM	Network	✓	✓	×
DEEPCAPA [21]	TTM	Windows	✓	✓	×
Fairbanks et al. [22]	TAM	Android	✓	×	×
DroidTTP	TTM	Android	✓	✓	✓

MD - Malware Detection, **MCI** - Malware Capability Identification, **TTM** - Tactic, Technique Mapping, **TAM** - Tactic Mapping, **NTA** - Network Traffic Analysis.

Table 2
Summary of the acronyms used in the paper.

Symbol	Description
ANN	Approximate Nearest Neighbor
APK	Android Package Kit
APT	Advanced Persistent Threat
BR	Binary Relevance
C2	Command and Control
CC	Classifier Chain
DL	Deep Learning
DT	Decision Trees
ICS	Industrial Control System
IoC	Indicator of Compromise
IoT	Internet of Things
LLM	Large Language Model
LP	Label Powerset
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
PC	Personal Computer
PTA	Problem Transformation Approach
RAG	Retrieval Augmented Generation
RF	Random Forest
SHAP	SHapley Additive exPlanations
SLM	Small Language Model
TTPs	Tactics, Techniques, and Procedures

Technique: *(i)* the attacker crafts a phishing email disguised as an urgent message from the victim's Android banking app, stating that the account has been locked and needs to be reset. *(ii)* The email contains a link to a fake login page, which is designed to look identical to the official banking app login screen. *(iii)* The victim, believing the message is legitimate, clicks the link and enters their login credentials into the fake page. *(iv)* The attacker then captures the credentials and uses them to access the victim's bank account, potentially conducting fraudulent transactions.

3.2. Adversarial Tactics, Techniques, and Common Knowledge

The MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) framework is a comprehensive knowledge base that catalogs adversary behaviors. Developed by MITRE, this framework provides a structured approach to understanding how attackers operate. It plays a vital role in threat intelligence, incident response, red teaming, and adversary emulation by mapping detected behaviors to known attack Techniques. The ATT&CK framework is organized into multiple matrices, each tailored to specific environments and attack vectors. These include the *Enterprise*, *Mobile* (Android and iOS), and *Industrial Control Systems (ICS)* matrices, which help analysts focus on different operating systems or environments.

Each matrix is further broken down into Tactics (e.g., Initial Access, Execution, Exfiltration, etc.). Techniques are then listed under each Tactic, providing detailed descriptions of the methods adversaries may employ to achieve their goals (e.g., phishing, credential dumping, or lateral movement). For example, consider the `Initial Access (TA0027)` Tactic in an Android attack scenario. A common technique used by attackers to achieve this goal is `Phishing (T1660)`,[7] where social engineering is used to trick victims into giving the attacker access to their devices. The attack progresses through a sequence of meticulously planned steps, further explained in the procedure examples and referenced within the specific Technique page.

Moreover, Techniques can have sub-Techniques which further details specific variations of the Technique. For example, the `Input Capture (T1417)`[8] Technique focuses on how adversaries intercept user input on a mobile device. This Technique is divided into sub-Techniques that outline distinct methods of input interception: `Keylogging (T1417.001)`, and `GUI Input Capture (T1417.002)`. A single Technique can belong to multiple Tactics. For example, the Input Capture Technique is used in both Credential Access and Collection Tactics.

3.3. Retrieval Augmented Generation

RAG [27] represents a fusion of information retrieval and language generation technologies. This approach enhances AI language models by connecting them with external knowledge sources that enable more informed and accurate responses. Unlike traditional language models that rely solely on their training data, RAG actively draws upon current information when generating responses.

The RAG workflow is illustrated in Fig. 1. At its core, RAG functions through a two-stage process. The first stage involves information *retrieval*, where the system searches through external knowledge sources to find content relevant to the current query. This works by converting both the user's question and the available reference documents into mathematical representations called embeddings. It then scans similarly encoded documents in its knowledge base, using mathematical comparison of embeddings to identify the most semantically relevant information.

The second stage leverages an LLM's *generative* capabilities. Rather than relying solely on its pre-trained knowledge, the model receives both the user's query and the retrieved relevant information (context) from the vector database. This allows the LLM to craft responses that incorporate specific, factual details from the retrieved sources while maintaining natural language fluency. The result is more accurate and contextually appropriate than what could be achieved by either retrieval or generation alone. This combination of dynamic knowledge access and sophisticated language generation represents a significant leap forward in Artificial Intelligence's ability to provide precise, contextually enriched responses.

[7] https://attack.mitre.org/techniques/T1660.
[8] https://attack.mitre.org/techniques/T141.

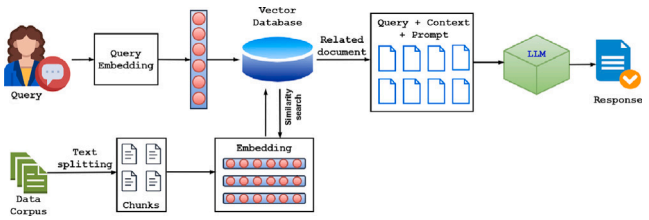


Fig. 1. Architecture of Retrieval Augmented Generation.

4. Methodology

In this section, we present the proposed methodology employed in the study. The overall architecture of the system is illustrated in Fig. 2. The process begins with the identification of Android-related hashes from the MITRE ATT&CK framework through three key steps: *(i)* Procedure References Analysis, *(ii)* Verification Using VirusTotal, and *(iii)* Labeling of Techniques and Tactics. Following data collection, we extract static features from Android applications using Androguard. These features are then refined through a feature selection mechanism to identify the most relevant attributes. Using the selected features, we develop a Tactic and Technique classification model based on the Problem Transformation Approach (PTA) combined with various ML classifiers. Furthermore, to investigate the potential of LLMs in predicting Tactics and Techniques, we develop a RAG for which we also carry out LLM fine-tuning. A detailed explanation of each phase of the methodology is provided in the following subsections.

4.1. Data collection

In this phase, we gather data to generate a model that maps Android applications to MITRE Tactics and Techniques. To the best of our knowledge, no existing dataset provides MITRE Tactics and Techniques classification for Android apps. Therefore, we initiated the data collection process by compiling hashes of Android applications. By compiling these hashes, we acquired the exact APK files referenced in the MITRE framework, enabling us to analyze their behaviors and accurately map them to specific Techniques. This process involved analyzing the Tactic and Technique information outlined in the MITRE ATT&CK Mobile Android Matrix.[9]

As outlined in Section 3, the MITRE framework categorizes adversarial behavior into distinct, high-level Tactics, each of which encompasses multiple specific Techniques. The Android matrix covers Tactics, including *Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Command and Control, Exfiltration,* and *Impact*. Moreover, each Technique is accompanied by technical procedures, and we used these procedure details to collect the Android applications. The data collection process involves the following three steps:

1. **Procedure References Analysis.** The first step involved a detailed examination of the procedure references associated with each Technique in the MITRE ATT&CK Mobile Android Matrix. These references, contributed by security practitioners, outline how attackers exploit specific vulnerabilities and techniques to carry out malicious activities on Android devices. They provide critical technical details such as permissions requested by the applications, inter-process communication intents, app activities, and additional IoCs, such as suspicious URLs or IP addresses. For example, the `Audio Capture (T1429)`[10] Technique, under the `Collection` Tactic, is linked to 49 references detailing scenarios in which malware records audio without user consent, abuses sensitive permissions, or exfiltrates captured data to external servers. Many of these references also provide cryptographic hashes (e.g., SHA1) of malicious files. For instance, one reference from Lookout[11] provides SHA1 hashes of *BouldSpy* samples along with their associated Command and Control details. By systematically analyzing these procedure references, we can compile a set of hashes. We crawled each reference link provided in the technical details and collected the corresponding hashes. In total, 1315 procedure references across various Techniques in the Android MITRE ATT&CK Matrix were analyzed to collect the hashes. To ensure consistency and technical relevance, we analyzed only those procedure references that contained concrete indicators such as cryptographic hashes, permissions, intents, or behaviors tied to malware samples. References lacking such technical artifacts e.g., those offering only high-level descriptions or news coverage were excluded from this process. Only those apps whose hashes were explicitly mentioned in these official procedure references were considered for inclusion. This ensures that every app selected is grounded in a verifiable and public report detailing its adversarial behavior in the context of a MITRE Technique. The collected hashes served as entry points for retrieving the corresponding APK files, which were subsequently analyzed to extract critical static features. These hashes were then verified and filtered using VirusTotal, as explained in the next step. The number of hashes collected for this study is detailed in Section 5.2.
2. **Verification Using VirusTotal.** The next step involved verifying the collected hashes using VirusTotal,[12] a widely recognized threat intelligence platform that provides comprehensive security reports on files, URLs, and IPs. VirusTotal aggregates data from multiple antivirus engines and other security tools, making it a powerful platform for threat analysis. It scans each hash to determine the file type (e.g., APK, Windows, etc.) and assesses whether it is malicious. To search for hashes, we created a VirusTotal account and received an academic API key from the VirusTotal team. This key enabled us to query bulk threat indicators, with a daily limit of 20,000 queries. Using the API key, we developed a code snippet to automatically scan all the hashes in bulk. For example, we verified a hash extracted from a Lookout Threat Intelligence reference on BouldSpy spyware using VirusTotal.[13] The report confirmed that the hash was compatible with an Android app, as shown in Fig. 3. Through this process, we were able to eliminate irrelevant data, such as hashes that did not match Android applications.
3. **Labeling Techniques and Tactics.** The final step of the data collection process involved labeling each verified hash with one or more MITRE ATT&CK Techniques and their corresponding Tactics based on the procedure references. This labeling is based on the procedure references analyzed in the first phase. For each verified hash, we identified the associated Technique(s) and Tactic(s) it represented. For example, suppose that a hash is included in the procedure reference for the *Audio Capture* Technique. We label it with the Audio Capture Technique ID and associate it with the corresponding Tactic, which in this case is *Collection*. Moreover, since an app can exhibit multiple behaviors corresponding to different Techniques and Tactics, we adopted a multi-labeling approach to assign all relevant labels

[9] https://attack.mitre.org/matrices/mobile/android.
[10] https://attack.mitre.org/techniques/T1429.
[11] https://www.lookout.com/threat-intelligence/article/iranian-spyware-bouldspy.
[12] https://www.virustotal.com/gui/home/search.
[13] https://www.virustotal.com/gui/file/f919be6a1920b2c206c62ae03ac69fad9955564618874245e91cd0aed051ed78/details.

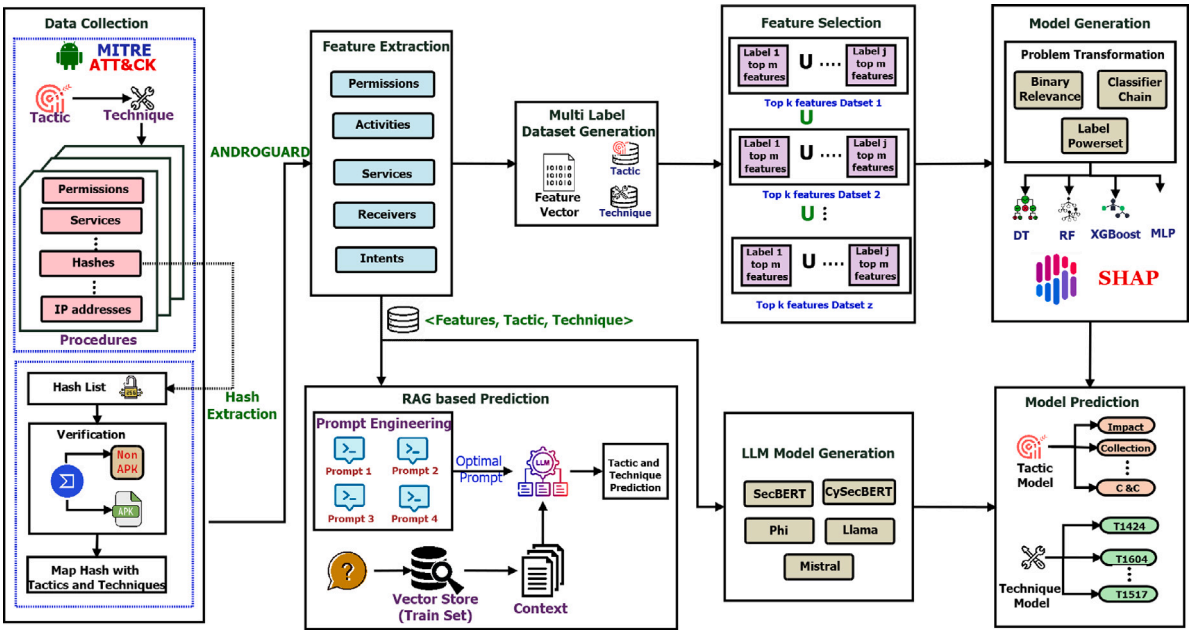
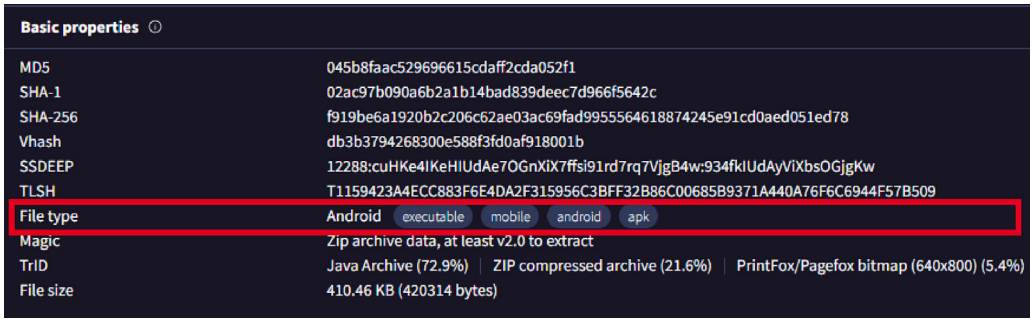


Fig. 2. Architecture of DroidTTP for mapping Android application with Tactics and Techniques.



Basic properties	
MD5	045b8faac529696615cdaff2cda052f1
SHA-1	02ac97b090a6b2a1b14bad839deec7d966f5642c
SHA-256	f919be6a1920b2c206c62ae03ac69fad9955564618874245e91cd0aed051ed78
Vhash	db3b3794268300e588f3fd0af918001b
SSDEEP	12288:cuHKe4IKeHIUdAe7OGnXiX7ffsi91rd7rq7VjgB4w:934fkIUdAyViXbsOGjgKw
TLSH	T1159423A4ECC883F6E4DA2F315956C3BFF32B86C00685B9371A440A76F6C6944F57B509
File type	Android executable mobile android apk
Magic	Zip archive data, at least v2.0 to extract
TrID	Java Archive (72.9%) | ZIP compressed archive (21.6%) | PrintFox/Pagefox bitmap (640x800) (5.4%)
File size	410.46 KB (420314 bytes)

Fig. 3. VirusTotal report confirming the hash matches an Android app.

to each app. For instance, an app exploiting the *Call Control* Technique could be labeled under the *Collection* Tactic (due to data gathering), *Command and Control (C2)* Tactic (for communication with a C2 server), and *Impact* Tactic (because of potential disruption to user communication). We applied this approach to label each app based on all the relevant Techniques and Tactics it exhibited. As a result, we created a dataset that includes Android app hashes along with their associated Technique IDs and corresponding Tactics.

4.2. Feature extraction

After collecting the hashes of Android apps, the next step involves extracting relevant features for a supervised learning classification task. In this study, we focus on extracting Android app features that are commonly referenced in the MITRE procedure details. For instance, the procedure reference for the `T1398 – Boot or Logon Initialization Scripts` Technique under the `Persistence` Tactic[14] examines the permissions requested by the malware (*Android.Oldboot bootkit*), providing insight into its persistence mechanisms. Similarly, static features such as activities, intents, services, receivers, strings, etc., are also detailed in the procedure reference documents. Based on the frequency of these static features mentioned in the references, we extracted the following features.

- *Permissions*. These are access rights requested by the app to perform specific actions. For example, navigation apps and social media platforms often request permission called `ACCESS_FINE_LOCATION` to access the device's GPS location for location-based features.
- *Activities*. Activities represent individual screens or interactions within the app's user interface. Each activity defines a specific user experience. A login screen (`LoginActivity`) enables users to enter their credentials for authentication.
- *Services*. These are background processes that operate independently of the user interface. Services can handle long-running tasks, such as playing music or fetching data from the Internet. For example, the `MusicPlayerService` in music streaming apps like Spotify plays music in the background, even when the app is minimized or the screen is off.
- *Receivers*. Receivers are Android components designed to listen for system events or inter-application broadcasts, enabling applications to dynamically respond to environmental changes. They facilitate event-driven execution, allowing apps to react to network state transitions, incoming messages, or system-level triggers. For instance, `BatteryReceiver` detects battery level fluctuations, enabling applications to implement power-saving mechanisms or monitor energy consumption efficiently.

[14] https://thehackernews.com/2014/01/first-widely-distributed-android.html.

- *Intents*. Intents facilitate communication between application components and the system, enabling seamless interaction and execution of predefined actions. Intent actions define the specific operations an application can perform, while intent categories provide contextual information to refine the behavior. For example, the "`ACTION_VIEW`" intent with a URL parameter triggers a web browser to open a webpage. Malware often exploits this mechanism to redirect users to malicious websites, leveraging intent-based redirection as a vector for phishing, drive-by downloads, or credential theft.

To extract the features, we performed a static analysis. Static analysis examines the application's code without executing it, thus avoiding the risks associated with running potentially malicious software. Static analysis typically involves reverse engineering Android Package (APK) files, which are compressed archives containing essential components such as the manifest file and other resources. The *AndroidManifest.xml* file within the APK is a key component for static analysis, as it contains valuable information about the app's structure and capabilities. To automate the feature extraction process, we used VirusTotal, which provides static details about Android APKs through Androguard.[15] Androguard offers comprehensive information about the app's permissions, activities, services, etc. To collect features, we first created a VirusTotal account and developed a script to query the platform for each app's hash. VirusTotal responds with data in JSON format, which our script processes to extract static features. Finally, we converted these features into binary feature vectors suitable for ML algorithms.

We denote this feature vector as $\mathcal{F} = (f_1, f_2, \ldots, f_s)$, where s is the number of unique static features. These unique features are determined by consolidating all features from the various hashes and identifying the distinct ones. For each specific hash, the features are encoded as:

$$f_i = \begin{cases} 1 & \text{if the } i\text{th feature is present} \\ 0 & \text{otherwise} \end{cases}$$

4.3. Feature selection

This phase addresses the crucial ML aspect of dimensionality reduction. High-dimensional datasets can lead to overfitting, in which the model performs well on the training data but struggles with unseen data. In addition, processing a large number of features can be computationally expensive. Thus, effective feature selection is essential to improve the accuracy of the model.

SelectKBest is a widely used [28,29] feature selection method that identifies the top k features based on their statistical significance in relation to the target variable. This approach evaluates each feature individually and ranks attributes according to a chosen statistical test, such as the chi-squared test or ANOVA F-value. Although this approach is effective in many cases, however, it has limitations in multi-label classification tasks. Specifically, features relevant to one label may be overlooked when evaluated independently, as this approach does not account for inter-dependencies among labels. Moreover, feature selection is often carried out on a specific training set determined by a random seed. However, this practice can lead to significant variability in results. Features selected from one dataset may not perform optimally when applied to a different dataset generated with a different random seed. This inconsistency arises because the selected features are solely based on the characteristics of a single training dataset, which may not be representative of other datasets. To address these challenges, we propose a novel feature selection strategy that leverages *SelectKBest* but enhances it for multi-label classification tasks. The proposed mechanism follows a two-step process that ensures consistent feature selection while accounting for label interdependencies to improve classification accuracy, namely:

1. **Label Specific Feature Selection**. The first step focuses on identifying features that are most relevant to each label within a specific dataset. To achieve this, we used the chi-square test in combination with the SelectKBest method, which ranks top m features according to their association with the target label. To calculate the chi-square value, we computed the Observed Frequency and the Expected Frequency. The Observed Frequency $\mathbf{O_{ij}}$ quantifies the number of occurrences in which feature f_i appears alongside label y_j in the dataset. In contrast, the Expected Frequency $\mathbf{E_{ij}}$ represents the anticipated occurrence of f_i given y_j, assuming that there is no statistical dependency between them. Expected Frequency (E_{ij}) is calculated by Eq. (1):

$$E_{ij} = \frac{T_{f_i} \times T_{y_j}}{N} \tag{1}$$

where T_{f_i} is the total occurrences of feature f_i, T_{y_j} is the total occurrences of label y_j, and N is the total number of instances in the dataset. The chi-square statistic is then computed for each feature-label pair using Eq. (2):

$$\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{2}$$

After calculating the chi-squared statistic for each feature-label pair, the *SelectKBest* method ranks the features based on their chi-squared values. Features with higher chi-squared scores are considered more statistically significant in their relationship with the label and are therefore selected for further use. For each label, the top m features with the highest chi-squared scores are selected.

2. **Generalizable Feature Subset**. In the second step, we apply the same procedure as in the first step to z different datasets, each generated using a different random seed. We analyze each dataset separately to identify the most important features. Finally, we combine the top features from all z datasets and extract only the unique ones. The comprehensive process of feature selection is presented in Algorithm 1.

Algorithm 1: Feature Selection Process

Input: $\{D_1, D_2, \ldots, D_z\}$: z datasets generated by different random seeds, $\mathcal{L}$: Set of labels, m: Number of top features to select for each label

Output: $\mathcal{F}_\mathcal{G}$: Generalizable subset of unique top features across all z datasets

```
Initialize F_G ← ∅
for i ← 1 to z do
    Initialize F_L^(i) ← ∅
    for each label l ∈ L do
        for each feature f_j in D_i do
            χ² = Σ_{i,j} (O_ij − E_ij)² / E_ij ;
            Store χ² score for feature f_j;
        Rank features f_j by their χ² score in descending order;
        Select top m features f_1, f_2, ..., f_m from ranked list;
        F_L^(i) = F_L^(i) + {f_1, f_2, ..., f_m};
    F_L^(i) = set(F_L^(i)) ;
    F_G = F_G + F_L^(i);
F_G = set( F_G);
return F_G
```

4.4. Tactic and Technique prediction using PTA

In this phase, we develop a classification model for predicting Tactics and Techniques associated with Android apps. To achieve this,

[15] https://github.com/androguard/androguard.

we used the Problem Transformation Approach. Traditional supervised learning typically involves single-label models, where each instance is assigned only one label. However, this single-label approach is not always practical. Attackers or threat groups behind malicious applications typically use a combination of different Tactics and Techniques to carry out their attacks. Thus, instead of the traditional single-label approach, we need to adopt a multi-label classification approach, where multiple labels are assigned to a single instance. Unlike a single-label dataset, which has only one column for labels, a multi-label classification dataset uses multiple columns for labels, with each column representing a different label. If two labels are associated with a single instance, the corresponding columns for those labels will have a value of 1 for that instance; otherwise, the value will be 0. To perform multi-label classification, we utilized the PTA [30]. This technique involves converting the multi-label classification task into several single-label classification problems. In our study, we used three PTAs that are commonly used in the literature [31–33].

- *Binary Relevance (BR):* Binary relevance decomposes the multi-label classification task into multiple independent single-label binary classification problems. Specifically, for each label $l_i \in \mathcal{L}$, a binary classifier h_i is trained to predict whether l_i is a part of Y_i. The binary classifier h_i outputs 1 if the label l_i is present and 0 otherwise. Mathematically, each classifier learns a function $h_i : X \rightarrow \{0,1\}$. The overall multi-label prediction for a new instance x is obtained by combining the predictions of all binary classifiers:

  $$\hat{Y} = \{l_i \in \mathcal{L} \mid h_i(x) = 1\}$$

- *Classifier Chain (CC):* In this approach, instead of training independent binary classifiers, a sequence (or chain) of binary classifiers is trained. Each classifier in the chain uses not only the original features, but also the predictions of previous classifiers as additional features. Each binary classifier h_i for label l_i predicts y_i using the feature vector x and the predictions of previous classifiers $y_1, y_2, \ldots, y_{i-1}$. For example, $x \in X$, the process starts with the original features x. The first classifier h_1 predicts y_1 using x. The second classifier h_2 predicts y_2 using x and y_1. This process continues until the last classifier h_t predicts y_t using x and all previous predictions. The final multi-label prediction for x is the set of all labels l_i for which $y_i = 1$.
- *Label Powerset (LP):* LP, also known as Label Combination (LC), converts the original multi-label classification task into a standard multi-class classification problem. Instead of modeling each label independently, LP treats each unique label combination that appears in the training data as a distinct class.
  Let $\mathcal{L} = \{l_1, l_2, \ldots, l_t\}$ be the set of all possible labels, where $t = |\mathcal{L}|$. The total number of possible label combinations is 2^t, i.e., the power set of $\mathcal{L}$, excluding the empty set if it is not used in the dataset.
  Each instance $x_i \in X$ is associated with a label set $Y_i \subseteq \mathcal{L}$. LP creates a new single-label class $z_i \in \mathcal{Z}$, where:

  $$z_i = \text{class}(Y_i), \quad \text{with} \quad \mathcal{Z} \subseteq \mathcal{P}(\mathcal{L})$$

  Here, $\mathcal{P}(\mathcal{L})$ denotes the power set of $\mathcal{L}$, and each element of $\mathcal{Z}$ corresponds to a unique label combination seen in the training data.
  Thus, the multi-label classification problem is transformed as:

  $$(x_i, Y_i) \rightarrow (x_i, z_i)$$

  where z_i is a single class label representing a unique combination of labels.
  For example, if there are three labels $\{l_1, l_2, l_3\}$, LP creates classes corresponding to all observed combinations of these labels in the dataset, such as $\{l_1\}$, $\{l_1, l_3\}$, $\{l_2, l_3\}$, $\{l_1, l_2\}$, $\{l_1, l_2, l_3\}$, etc. The transformed dataset is then used to train any standard multi-class classifier (e.g., Decision Tree, Random Forest, XGBoost, etc.), which predicts a single labelset for each instance. This predicted class is then mapped back to the corresponding subset of labels in $\mathcal{L}$. The primary advantage of LP is that it preserves label correlations, which are often ignored in methods that treat labels independently, such as Binary Relevance (BR).

For the implementation of these PTAs, we employ a diverse set of classifiers that are effective in multi-label classification. Random Forest (RF) [34] and Decision Trees (DT) [35] excel at modeling complex decision boundaries while leveraging ensemble methods to capture label correlations. XGBoost enhances performance in structured data by efficiently handling imbalanced multi-label distributions and optimizing multiple objectives through boosting techniques [36]. Meanwhile, Multi-Layer Perceptron (MLP) leverages DL to learn hierarchical representations and capture label dependencies through hidden layers, improving predictive accuracy in multi-label scenarios [37]. We selected these classifiers due to their demonstrated effectiveness in previously published research on similar tasks [20,38].

4.5. Tactic and Technique prediction using LLM

In this phase, LLM-based methods are investigated alongside traditional ML classifiers to investigate their potential as complementary or alternative approaches for Tactics and Techniques prediction. The advancement of LLMs has greatly enhanced Natural Language Processing (NLP) tasks, including threat intelligence [25,26], due to their ability to understand complex semantics and scale with large datasets. Recent studies [18,39] have shown the effectiveness of LLMs in the processing TTPs. In this work, LLM techniques comprising prompt engineering, Retrieval-Augmented Generation (RAG), and fine-tuning are employed to enhance the model's ability to interpret static application features in the context of known threat behaviors. Prompt engineering is used to guide the LLM's predictions by formulating inputs that align with the desired TTP outputs. We used RAG to enable the LLM to incorporate up-to-date external knowledge, crucial for the dynamic nature of cybersecurity threats, and to mitigate the risk of generating inaccurate information. Moreover, fine-tuning is applied to specialize the LLM on Android malware data, addressing potential domain gaps and mitigating model hallucination, with the goal of producing more reliable and precise TTP predictions. The effectiveness of these LLM based methods is quantitatively evaluated and compared against conventional ML approaches to assess their practical value in enhancing TTP prediction capabilities.

4.5.1. Prompt engineering

In this stage, we develop a prompt-based approach to predict TTPs from static features of Android applications using LLMs. Designing prompts that effectively guide LLMs in generating accurate and relevant responses is a challenging task [40]. A well-designed prompt should strategically direct the model to leverage its internal knowledge effectively. Prompt engineering involves designing inputs that align model outputs with desired objectives [41]. The effectiveness of a prompt is influenced by several factors, such as its structure, formatting, and linguistic subtleties, as LLMs are highly sensitive to even small changes in these aspects [42]. To optimize the extraction of relevant TTPs, we employ prompt engineering. We have developed four distinct prompting strategies, each designed to extract TTPs from static features with varying levels of specificity in user instructions. After evaluating the performance of these prompts, we incorporate a RAG approach to further enhance model accuracy and contextual relevance. In the following, we outline the four prompt templates used in this study. Each prompt used in the study is provided in Appendix.

1. The first prompt strategy provides straightforward instruction to analyze the static features and predict the associated Tactics and Techniques, as shown in Table A.15. This prompt includes clear fundamental parameters for the task, a structured format for the output, and a fallback response, "Not enough information" if the data are insufficient for making predictions.
2. The second prompting strategy (see Table A.16) introduces a domain-expert context by framing the query from the perspective of a cybersecurity expert. This helps to activate the domain-relevant knowledge of the model, potentially improving prediction accuracy through expert-aligned reasoning patterns. The prompt incorporates references to the MITRE ATT&CK framework, includes a precise and professional output format, and uses domain-specific language to simulate expert-level analysis.
3. The third prompt represents an advanced strategy for analyzing Android application security by leveraging static features to identify associated MITRE ATT&CK Tactics and Techniques as shown in Table A.17. It incorporates *(i)* a structured definition of Tactics and Techniques, *(ii)* explicit references to the MITRE ATT&CK Android matrix for threat intelligence validation, *(iii)* a scoring mechanism to quantify the likelihood and relevance of identified TTPs, and *(iv)* a standardized response format to ensure consistency and interpretability in security assessments.
4. The fourth strategy represents the most advanced approach, incorporating comprehensive technical context about Android static features and their security implications, as shown in Table A.18. This prompt includes *(i)* detailed technical specifications of Android static features, *(ii)* definitions of Tactics and Techniques, *(iii)* references to threat intelligence platforms, and *(iv)* a standardized output schema optimized for consistency in predictions.

4.5.2. Retrieval Augmented Generation for Tactics and Techniques prediction

This stage explores the effectiveness of the RAG model in predicting Tactics and Techniques based on static features. As explained in the background 3, the RAG framework comprises two primary components: the *retriever* and the *generator*. The retriever is tasked with locating relevant information within a large knowledge base, which is stored in a vector database. This database holds precomputed dense vector representations of the indexed information. To generate the knowledge base, we utilized the same training samples as in the ML-based approach but stored them differently in CSV format. Each training sample comprises three key elements: *(i)* Description represents a descriptive summary of the app's features, formatted as follows: "*Activities related to app* ⟨*Hash* ⟩ *are:* ⟨*Activities* ⟩. *Permissions required:* ⟨*Permissions* ⟩. *Services used:* ⟨ *Services* ⟩. *Receivers included:* ⟨*Receivers* ⟩. *Intent Actions:* ⟨*Intent Actions* ⟩ *and Intent Categories:* ⟨*Intent Categories* ⟩". *(ii)* Tactic represents the associated Tactics used by the attacker using this particular app *(iii)* Technique specifies the relevant Techniques associated with the attack.

We use the CSVLoader to load the dataset and divide it into smaller, manageable segments for easier processing. Each segment is then transformed into a dense vector representation using an embedding model. Once converted into dense vectors, the dataset is indexed in a vector database. This indexing facilitates efficient similarity searches, enabling the retrieval of the most contextually relevant documents in response to queries.

When a mobile security analyst submits a query, it undergoes processing and conversion into a vector representation using the same embedding model. This vector representation is then used with an Approximate Nearest Neighbor algorithm (ANN) to search for the most similar documents in the vector database, using cosine similarity as the metric. The ANN algorithm ensures that only the top-k most relevant documents are retrieved, reducing the computational burden while ensuring that the most pertinent context is available for generating predictions. Once the retriever identifies the top-k relevant documents, these, along with the query, are passed to the generator component. The generator, typically a pretrained LLM, is responsible for producing the final output based on the provided context and prompt. Specifically, the LLM is given the relevant context retrieved by the retriever, reducing the likelihood of hallucinations (i.e., generating incorrect or fabricated information). To evaluate how the amount of information provided to the generator impacts its performance, we experiment with varying context sizes (k). By adjusting the number of relevant documents fed into the generator, we can assess how increasing or decreasing the amount of contextual data influences the accuracy, coherence, and relevance of the model's output. In our approach, the prompt used for the generator is the same prompt identified in the previous phase as the most effective for generating accurate responses. This methodology significantly mitigates the risk of irrelevant or fabricated information, as the model relies solely on the context retrieved from the vector database.

4.5.3. Fine tuning LLM

When adapting pre-trained LLMs for specific applications, fine-tuning avoids the resource-intensive process of complete parameter retraining. In the context of cybersecurity, a significant challenge arises from the lack of domain-specific information in pre-training datasets, leading to hallucination issues. Fine-tuning addresses this gap by incorporating specialized knowledge, thereby reducing hallucinations and improving accuracy in predicting attackers' Tactics and Techniques. In this study, we experimented with a range of open-source pre-trained LLMs with various strengths and architectures, including SecBERT[16] and CySecBERT [43], which were developed specifically for cybersecurity tasks. We selected *Phi-3-mini-4k-instruct* for its ability to generalize well, despite its compact size. The Phi series of Small Language Models (SLMs) is notable for its unique combination of high performance and cost-effectiveness, consistently outperforming models of similar or larger size in various linguistic tasks [44]. Moreover, we experimented with *Meta-LLaMa-3-8B-Instruct* for its versatile capabilities and efficient architecture, which are used in various downstream tasks [45]. This 8-billion parameter model strikes an optimal balance between computational requirements and performance capabilities. Similarly, we experimented with *Mistral-7B-Instruct-v0.2* for its optimal performance and efficiency [46].

To optimize the fine-tuning process, we employed Quantized Low-Rank Adaptation (QLoRA) [47] Technique with 4-bit quantization. This approach minimizes computational demands while maintaining model effectiveness, making it particularly valuable for updating large models with limited computing resources.

5. Experiment and evaluation

This section describes the experiments carried out to test the effectiveness and robustness of our approach. In particular, in the following sections, we will describe the testbeds, the data, and the evaluation metrics used to test our framework, and we report the obtained results along with their critical analysis.

5.1. Experimental setup

We conducted the implementation on an Intel i9 Windows system equipped with a 5 GB GPU. We also used a workstation equipped with an Intel i9-14900KF, dual RTX4090 GPUs, and 128 GB of RAM for LLM experimentation. For the fine-tuning process, we leveraged the Hugging Face Transformers library, utilizing pre-trained language models and optimizing them with PyTorch. In the RAG framework, we

[16] https://github.com/jackaduma/SecBERT.

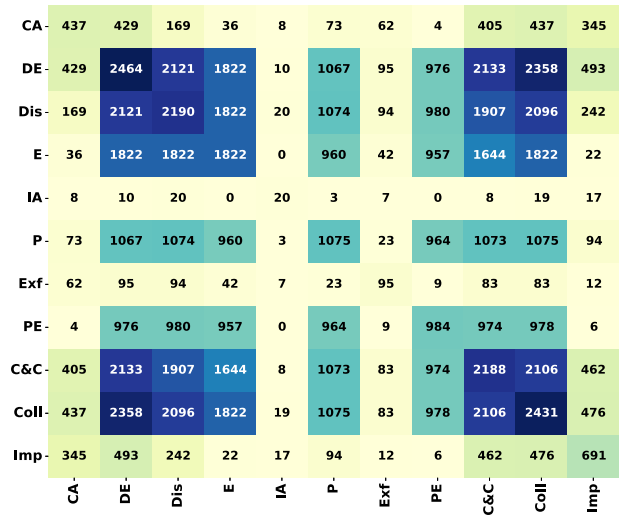


Fig. 4. Frequency of samples across various classes in the Tactic dataset. **CA-** Credential Access, **DE-** Defense Evasion, **Dis-** Discovery, **E-** Execution, **IA-** Initial Access, **P-** Persistence, **Exf-** Exfiltration, **PE-** Privilege Escalation, **C&C-** Command and Control, **Coll-** Collection, **Imp-** Impact.

integrated Facebook AI Similarity Search (FAISS)[17] for efficient document retrieval and LangChain[18] to manage the retrieval and response generation pipeline. For multi-label classification tasks, we employed the scikit-multilearn library. To mitigate class imbalance, we incorporated the Multi label Synthetic Minority Over-sampling Technique (MLSMOTE) [48] for the generation of synthetic samples. To enhance model interpretability, we utilized the SHAP library and, for data visualization, we used the Python library matplotlib.

5.2. Dataset

As discussed in Section 4.1, to the best of our knowledge, no existing dataset was available. Therefore, we created our own dataset for experimentation following the process described in Section 4.1. We initially extracted 3034 hashes from the procedure references from the MITRE ATT&CK knowledge base. These hashes were subsequently verified using VirusTotal, which revealed that 261 hashes did not correspond to Android applications. As a result, these non-Android hashes were excluded from further analysis, leaving us with a dataset of 2774 hashes representing Android applications. To gain a deeper understanding of the nature of these Android applications, we performed a detailed analysis of the 2774 hashes on VirusTotal. This analysis provided information on the malware families associated with these applications. The findings revealed that the majority of the applications were primarily categorized as Trojan malware, with other notable categories including Adware, Spyware, Banker, Dropper, and Ransomware. We found that some of these applications were specifically designed to steal sensitive financial information, intercept SMS messages, and overlay fake login pages. Key families within this category include well-known threats such as Bankbot, Anubis, Riltok, Ginp, Bank, TrickMo, Svpeng, Flubot, Marcher, CBTx, etc. Additionally, our analysis identified several Spyware/Info-Stealer families, which are used for surveillance, data exfiltration, and persistent monitoring of victims. These include SpyAgent, Boogr, Sunbird, Hornbill, Monitor, ProjectSpy, SpyC23, Micropsia, GoldcupSpy, ViceLeaker, Domestickitten, StrongPity, Bahamut, AbstractEmu, and others. Another significant group in our dataset consisted of SMS Stealers/SMS Interception apps. These applications primarily target SMS data, often intercepting two-factor authentication (2FA) codes or exploiting premium-rate SMS abuse. Examples of these include SmsSpy, SmsBot, SmsAgent, SMSReg, SmsThief, Sriy, Srhu, Firq, etc. In addition to the categories mentioned above, we also discovered a number of Fake Apps/Adware/Hidden Apps. These applications either impersonate legitimate apps or conceal their presence from the user to avoid detection. Our dataset also included instances of Click Fraud/Advertising Abuse, where infected devices are exploited to generate ad clicks or install additional malicious apps. The associated malware families in this category include Judy, Ewind, Plankton, MobWin, and others. Moreover, we identified several Rooting/Exploit-based Malware samples in our dataset. These attempts to gain elevated privileges (root access) on the device or exploit known vulnerabilities for persistence or privilege escalation. To create the Tactic classification dataset, we used 2774 hashes, each associated with corresponding Tactic labels. From these labels, we excluded the `Lateral Movement` Tactic from the dataset due to its association with only a single hash, which lacks sufficient data for effective model training and generalization. After completing all necessary steps, we compile a Tactic classification dataset consisting of 2774 apps with 13 324 features and 11 Tactic labels. The distribution of the samples from the Tactic data set is illustrated in Fig. 4. Since the dataset is multi-labeled, an app can exhibit multiple Tactics. Fig. 4 reports the number of samples for each Tactic class as well as the co-occurrence of samples across multiple Tactic classes. For example, the dataset includes 2464 samples labeled with the `Defense Evasion` Tactic and 20 samples labeled with the `Initial Access` Tactic. This indicates that the dataset is highly imbalanced. Furthermore, the figure reveals that the 429 samples are labeled with both the `Credential Access` and `Defense Evasion` Tactics. Similarly, we generated a Technique classification dataset with the same number of samples and feature sets. While MITRE ATT&CK defines 85 Techniques for Android, many of these Techniques lack sufficient procedure references with hashes. As part of our data curation process, we established a minimum sample threshold of 10 instances per Technique to ensure reliable model training and evaluation. Techniques with fewer than 10 associated samples were excluded. Based on this criterion, we retained 48 Techniques that met the threshold. The distribution of Techniques is illustrated in Fig. 5. Specifically, most Techniques are associated with a similar percentage of samples, with T1636 (Protected User Data) and T1406 (Obfuscated Files or Information) appearing most frequently. In contrast, Techniques such as T1481 (Web Service) and T1662 (Data Destruction) occur at significantly lower rates.

Furthermore, the dataset includes samples specifically related to adversarial behaviors designed to evade detection through anti-analysis Techniques. A prominent Technique, Obfuscated Files or Information (T1406), is observed in 76.53% of the samples. This Technique involves adversaries making payloads difficult to detect or analyze by obfuscating their contents through encryption, encoding, or compression. The high prevalence of this Technique underscores its widespread use by malware to circumvent traditional static analysis methods. Also, Virtualization/Sandbox Evasion (T1633) appears in 4.65% of the samples. This Technique involves adversaries detecting and evading virtualized environments or sandboxes commonly used in malware analysis. By identifying system artifacts indicative of a virtual machine or sandbox, adversaries can alter the malware's behavior to avoid detection or conceal key functionalities of the payload. Another Technique, Download New Code at Runtime (T1407), is present in 35.72% of the samples. This Technique allows adversaries to download and execute dynamic code after the initial app installation. It is often employed to evade static analysis checks and pre-publication scans in official app stores, enabling malware to bypass traditional security mechanisms that rely on static inspection of application packages.

[17] https://ai.meta.com/tools/faiss.
[18] https://www.langchain.com.

Table 3
Optimized hyperparameters for Tactic and Technique classification.

Classifier	Parameter	PTA					
		BR		CC		LP	
		Tactic	Technique	Tactic	Technique	Tactic	Technique
DT	Criterion	Entropy	Gini	Entropy	Gini	Gini	Entropy
	max_depth	41	63	91	64	31	46
	min_samples_leaf	3	3	2	2	4	3
	max_leaf_nodes	–	50	100	20	50	None
	min_samples_split	17	12	14	5	19	4
RF	max_depth	None	20	Auto	Auto	log2	Auto
	min_samples_leaf	1	2	1	1	1	1
	min_samples_split	3	3	5	5	6	5
	n_estimators	78	64	96	178	199	153
XGBoost	Gamma	0.632	1.679	0.644	1.896	0	0
	learning_rate	0.242	0.124	0.243	0.253	0.3	0.3
	max_depth	9	14	5	12	6	6
	n_estimators	82	156	191	54	100	100
MLP	hidden_layer_sizes	100	100	100	100	100	100
	Activation	Relu	Relu	Relu	Relu	Relu	Relu
	Solver	Adam	Adam	Adam	Adam	Adam	Adam
	learning_rate	0.001	0.001	0.001	0.001	0.001	0.001

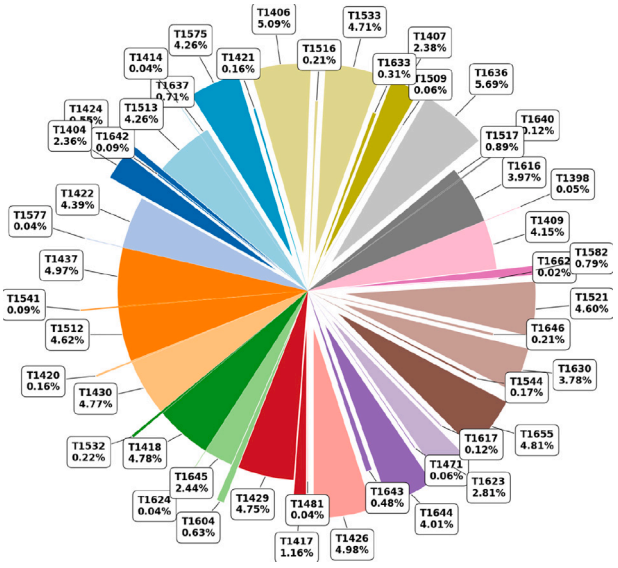


Fig. 5. Technique ID distribution in DroidTTP technique dataset.

5.3. Hyperparameter setup

The performance of the model is highly dependent on the hyperparameters selected for each classifier. To optimize these parameters, we applied Randomized Search Cross-Validation (CV) to the Decision Tree, Random Forest, and XGBoost classifiers. The optimized hyperparameters for each classifier, tailored for both Tactic and Technique classification tasks, are summarized in Table 3.

5.4. Evaluation metrics

To assess the performance of our Tactic and Technique classification model, we employed the evaluation metrics, such as Accuracy (A), Precision (P), Recall (R), F1-score (F1), Jaccard Similarity (JS), and Hamming Loss (HL).

- *Accuracy (A)*. It measures the proportion of samples that were correctly classified by the model.

$$\mathrm{A} = \frac{1}{N}\sum_{i=1}^{N}(y_i = \hat{y}_i) \tag{3}$$

where N is the total number of instances, y_i is the true label for the ith instance, and $\hat{y}_i$ is the predicted label for the ith instance.

- *Precision (P)*. Precision evaluates the proportion of correctly predicted labels among all predicted labels and is defined as:

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{4}$$

where, TP (True Positives) refers to the number of instances correctly predicted as belonging to a particular class, while FP (False Positives) denotes the number of instances incorrectly predicted as belonging to that class. Weighted precision ($P_{weighted}$) adjusts for class imbalance by computing the precision for each label and weighting it by the number of true instances. Macro precision (P_{macro}), on the other hand, calculates the unweighted mean of precision across all labels, treating all classes equally regardless of their frequency.

- *Recall (R)*. Recall measures the proportion of correctly predicted labels among all true labels and is defined as:

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{5}$$

where, FN (False Negatives) refers to the number of instances that belong to a class but were not predicted as such.

- *F1-score (F1)*. The F1-score is the harmonic mean of precision and recall, and is defined as:

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}} \tag{6}$$

- *Jaccard Similarity (JS)*. It measures how similar the predicted set of labels is to the true set of labels. A higher Jaccard Similarity indicates better model performance.

$$\mathrm{JS} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|} \tag{7}$$

where n is the total number of instances. $|y_i \cap \hat{y}_i|$ is the size of the intersection of the true and predicted label sets (i.e., the number of labels that are correctly predicted). $|y_i \cup \hat{y}_i|$ is the size of the union of the true and predicted label sets (i.e., the total number of unique labels in either the true or predicted set).

- *Hamming Loss (HL)*. Measures the average fraction of labels that are incorrectly predicted across all instances. *Hamming Loss (HL)* is calculated using Eq. (8). A lower Hamming Loss corresponds to better performance, with $HL = 0$ indicating perfect predictions (i.e., all labels are correctly predicted) and $HL = 1$ meaning that

Table 4
Performance comparison of Tactic classification models.

PTA	Classifier	A	$P_{weighted}$	$R_{weighted}$	$F1_{weighted}$	P_{macro}	R_{macro}	$F1_{macro}$	JS	HL
BR	DT	0.9218	0.9833	0.9829	0.9828	0.9657	0.9074	0.9273	0.9630	0.0159
	RF	0.9191	0.9831	0.9821	0.9817	0.9703	0.8716	0.9045	0.9634	0.0164
	XGBoost	0.9539	0.9883	0.9906	0.9894	0.9711	0.9373	0.9502	0.9765	0.0100
	MLP	0.9247	0.9833	0.9798	0.9811	0.9764	0.8909	0.9202	0.9610	0.0173
CC	DT	0.9286	0.9795	0.9836	0.9812	0.9623	0.9057	0.9251	0.9630	0.0175
	RF	0.9250	0.9819	0.9842	0.9822	0.9628	0.8767	0.9059	0.9652	0.0160
	XGBoost	0.9618	0.9866	0.9914	0.9889	0.9738	0.9404	0.9516	0.9779	0.0104
	MLP	0.9240	0.9847	0.9782	0.9810	0.9678	0.8865	0.9154	0.9596	0.0173
LP	DT	0.9587	0.9840	0.9876	0.9856	0.9468	0.9284	0.9318	0.9730	0.0135
	RF	0.9623	0.9851	0.9875	0.9861	0.9699	0.9267	0.9427	0.9749	0.0130
	XGBoost	**0.9694**	**0.9880**	**0.9905**	**0.9892**	**0.9712**	**0.9537**	**0.9604**	**0.9794**	**0.0102**
	MLP	0.9310	0.9759	0.9728	0.9737	0.9636	0.8855	0.9115	0.9523	0.0242

all labels are incorrectly predicted for all instances.

$$HL = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\mathcal{L}_i|}\sum_{j=1}^{|L_i|}\mathbb{I}(y_{ij} \neq \hat{y}_{ij}) \tag{8}$$

where n is the total number of instances. $|\mathcal{L}_i|$ is the number of labels for the ith instance. y_{ij} is the actual label of the jth label for the ith instance. $\hat{y}_{ij}$ is the predicted label of the jth label for the ith instance. $\mathbb{I}$(condition) is the indicator function, returning 1 if the condition is true and 0 otherwise.

5.5. Experimental evaluation of tactic identification using PTA

In this section, we report the results of our Tactic classification experiments designed to answer RQ_1: *Can DroidTTP effectively infer the adversarial Tactics employed by attackers based on the features derived from the Android applications?* As discussed in Section 2, we experimented with various PTAs and ML algorithms. Initially, we developed the Tactic model using the full feature set and tested it with ten different random seeds. To evaluate the performance of the Tactic models, we calculated the average score across these random seeds. Table 4 summarizes the performance of the Tactic models. The results indicate that the Label Powerset method combined with XGBoost outperforms other classifiers, achieving superior performance metrics including a low Hamming Loss of 0.0102, and a high Jaccard Similarity index of 0.9794. This strong performance can be attributed to LP's ability to explicitly model label correlations and XGBoost's strength in capturing complex, non-linear feature relationships. By treating each unique label combination as a distinct class, LP captures inter-label dependencies that Binary Relevance overlooks by modeling each label independently, thereby missing critical label interactions. Furthermore, LP avoids the sequential prediction dependencies found in Classifier Chains, which are prone to error propagation. XGBoost complements LP by effectively learning intricate patterns in the data, enhancing both accuracy and generalization.

We then applied the feature selection procedure outlined in Section 4.3 to reduce the feature set, optimizing computational efficiency and minimizing training time. As discussed, when the Tactic model was trained using the full feature set, the Label Powerset combined with XGBoost yielded the best performance; therefore, we applied feature selection specifically for the XGBoost classifier using the Label Powerset approach. As described in the feature selection procedure, we initially selected the top m features for each class, then combined these top m features across all labels. This process was repeated for 10 different random seeds to identify the final top features across the various datasets generated by random seeds and labels. To achieve this, we tested with values of m ranging from 100 to 13,300 and recorded the training time, top features, and various evaluation metrics. Fig. 6 presents the Jaccard Similarity score for the Tactic classification model after applying feature selection. We limited the plot to m values between 100 and 1500 after determining that beyond 1500 features, the results remained consistent when m was fixed at 1500.

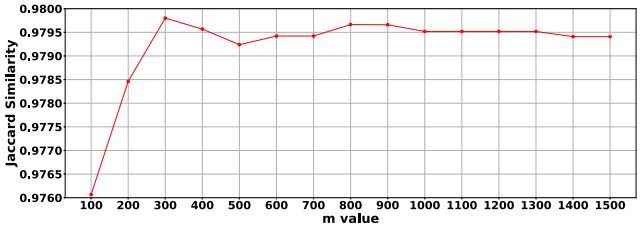


Fig. 6. Jaccard Similarity score of Tactic classification model when feature selection applied.

From Fig. 6, it is evident that the highest Jaccard Similarity score of 0.9798 is achieved when m is set to 300. When the top 300 features were selected for each label, we obtained 1737 final features aggregated from all ten datasets. Using this optimized feature set, we developed the Tactic classification model, which is trained and evaluated across 10 random seeds. This model is slightly higher than that of the model using the full feature set. Also, we observed that building a Tactic model using the full feature set required significantly longer execution time, taking 126.69 s. In contrast, the optimized feature set of 300 features reduced the execution time to just 18.61 s. This substantial reduction in training time demonstrates the effectiveness of feature selection in significantly enhancing model efficiency without compromising performance. To further analyze the classification results, we presented the individual weighted precision, recall, and F1-scores in Table 5. From this table, we can infer that most Tactic classes achieved high performance, with scores exceeding 0.95 across all metrics, except for the `Initial Access` and `Exfiltration` Tactics. The F1-score for `Initial Access` is 0.82, while the F1-score for `Exfiltration` is 0.89. This performance gap is primarily due to the low number of samples for these classes. Specifically, `Initial Access` has only 20 samples, and the `Exfiltration` Tactic is represented by just 95 samples, as illustrated in the Fig. 4. So, to address this issue, we augmented the dataset using the Multi-Label Synthetic Minority Over-sampling Technique (MLSMOTE) [48].

MLSMOTE is an adaptation of the popular Synthetic Minority Over-sampling Technique (SMOTE) [49], specifically developed for multi-label classification problems. In this technique, minority labels, such as `Initial Access` and `Exfiltration`, are identified based on their frequency within the dataset. After recognizing the minority labels, MLSMOTE identifies the minority instances, which are the data points associated with these underrepresented labels. To address the class imbalance, MLSMOTE generates synthetic samples for these minority labels. For each minority instance, it identifies the k Nearest Neighbors within the same label group (we used the default value of 5 nearest neighbors as specified in MLSMOTE). It then creates synthetic data by interpolating between the feature vectors of the minority instance and one of its neighbors, adding minor random variations. The synthetic instance is assigned the same labels as the original minority instance,

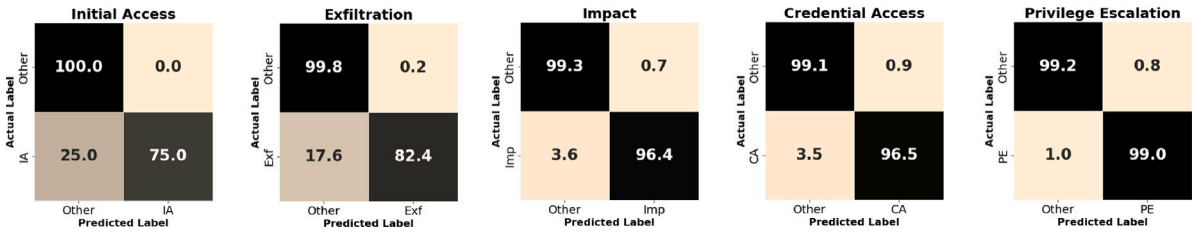


Fig. 7. Confusion matrix of Tactic classification model before Augmentation.

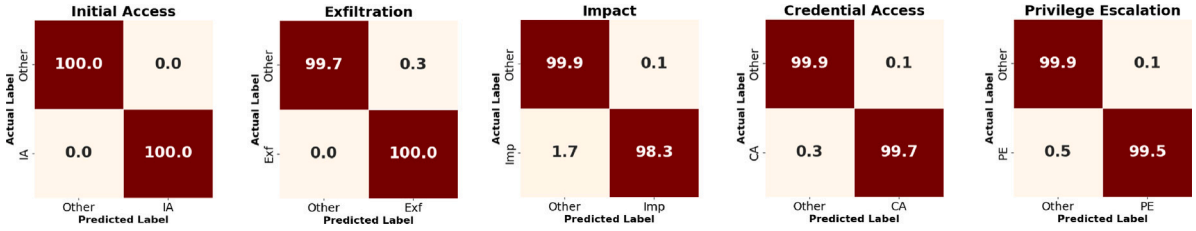
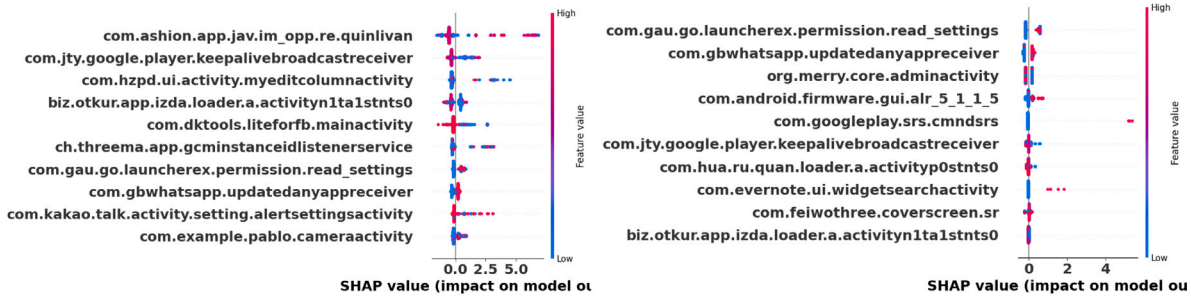


Fig. 8. Confusion matrix of Tactic classification model after augmentation.



(a) Exfilitration (b) Collection

Fig. 9. SHAP summary plot of Tactic model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5
Class-wise performance comparison between original and augmented datasets for Tactic classification model.

Tactic	Original dataset			Augmented dataset		
	P	R	F1	P	R	F1
Credential Access	0.95	0.97	0.96	0.99	0.99	0.99
Defence Evasion	0.99	0.99	0.99	1.00	1.00	1.00
Discovery	0.99	0.99	0.99	0.99	1.00	1.00
Execution	0.99	1.00	1.00	0.99	1.00	1.00
Initial Access	0.95	0.74	0.82	0.99	1.00	0.99
Persistance	0.99	1.00	0.99	0.99	1.00	0.99
Exfiltration	0.93	0.85	0.89	0.99	1.00	1.00
Privilage Escalation	0.99	0.99	0.99	0.99	0.99	0.99
Command and Control	0.99	0.99	0.99	0.99	1.00	0.99
Collection	0.99	0.99	0.99	1.00	1.00	1.00
Impact	0.98	0.97	0.97	0.99	0.98	0.98

and this process is repeated for other minority instances. To control the number of synthetic samples generated, we tested various sample sizes based on the maximum count of the majority class. As shown in Fig. 4, `Defense Evasion` has the highest count, with 2464 instances. For this, we generated synthetic samples at rates of 25%, 50%, 75%, and 100% of the highest class count, ensuring that the synthetic data did not exceed the original number of samples.

After augmenting the dataset, we generated the Label Powerset using the XGBoost model with 10 random seeds. The average results of the evaluation metrics for various sampling scenarios, along with those for the original dataset, are presented in Table 6. From the table, it is clear that as the number of synthetic samples increases, the model performance also improves. The best results were achieved when the number of synthetic samples matched the highest class count. In this case, the model achieved Jaccard Similarity score of 0.9893, and a Hamming Loss of 0.0054. Furthermore, our analysis shows that after data augmentation, the performance of all classes, particularly the minority classes such as `Initial Access` and `Exfiltration`, has significantly improved. As shown in Table 5, the F1-score for `Initial Access` increased from 0.82 to 0.99, while the F1-score for `Exfiltration` rose from 0.89 to 1.00. These improvements highlight that the augmented dataset enhances the effectiveness of Tactic classification.

To further analyze the classification results, we plotted a confusion matrix, shown in Fig. 7. Since we used the PTA Approach, this led to multiple binary classification confusion matrices, one for each class label. As a result, the overall confusion matrix consists of 11 individual matrices, each representing the classification performance for a specific Tactic. In the figure, we presented only 5 confusion matrices corresponding to the lowest performance in Tactic classification using the original data. Additionally, we plotted the confusion matrices for the same classes after augmentation (Fig. 8) to assess the impact on classification and misclassification rates. From these two confusion matrices, it is evident that the misclassification rate decreased after data augmentation. Specifically, before augmentation, the misclassification rate for `Initial Access` was 25%, which dropped to 0 after augmentation. Similarly, for the `Exfiltration` class, the misclassification rate decreased from 17.6% to 0. This pattern of improvement is observed across the other classes as well.

To thoroughly analyze and interpret the decision-making process of the model, we utilized the SHapley Additive exPlanations (SHAP)

Table 6
Tactic classification model performance metrics at different percentages of MLSMOTE.

(%)	A	$P_{weighted}$	$R_{weighted}$	$F1_{weighted}$	P_{macro}	R_{macro}	$F1_{macro}$	JS	HL
Original	0.9694	0.9881	0.9908	0.9894	0.9761	0.9564	0.9618	0.9798	0.0099
25	0.9768	0.9912	0.9936	0.9924	0.9885	0.9907	0.9895	0.9857	0.0074
50	0.9784	0.9920	0.9941	0.9930	0.9901	0.9917	0.9909	0.9868	0.0068
75	0.9807	0.9933	0.9947	0.9940	0.9917	0.9929	0.9923	0.9883	0.0058
100	**0.9819**	**0.9939**	**0.9950**	**0.9945**	**0.9925**	**0.9935**	**0.9930**	**0.9893**	**0.0054**

tool. SHAP provides granular insights into the contribution of each feature to the model's predictions, offering a comprehensive understanding of how static features influence the outcomes. We generated SHAP summary plots for each Tactic, with specific illustrations for the `Exfiltration` and `Collection` Tactics shown in Fig. 9. In these plots, each dot corresponds to a SHAP value representing an individual dataset sample. Features are ranked according to their average impact on the model predictions, as indicated on the vertical axis. The horizontal axis denotes the SHAP values, reflecting the extent to which a feature drives the prediction towards or away from the target class. The color gradient ranging from blue to red reflects the actual feature value for each instance, with blue indicating a lower value and red indicating a higher value.

In the SHAP summary plot for the *Exfiltration* Tactic features such as `com.ashion.app.jav.im_opp.re.quinlivan` and `com.jty.google.player.keepalivebroadcastreceiver` demonstrate a strong association with the model's prediction of exfiltration behavior. Higher SHAP values for these features (indicated by red dots) suggest that these specific app components are influential in predicting data exfiltration. For example, the feature `com.jty.google.player.keepalive broadcastreceiver` represents a background service that ensures the app remains active and continuously transmits data, even when the user is not directly interacting with the app.[19] On the other hand, for the *Collection* Tactic, features such as `com.gau.go.launcherex.permission.read_settings` and `com.gbwhatsapp.updatedanayppreceiver` significantly contribute to the model's prediction of data collection activities. The feature `com.gau.go.launcherex.permission.read_settings` is custom permission specific to the GO Launcher app, allowing other apps to read its settings.[20] Similarly, `com.gbwhatsapp.updatedanayppreceiver`, a custom receiver, has a positive impact on the model's prediction for *Collection* Tactics.

Furthermore, we identified the top ten most influential features for each Tactic using SHAP values and extracted the unique features across all Tactics to analyze their overall impact. The heatmap in Fig. 10 visualizes the relationship between static features and predicted Tactics. The horizontal axis represents the features (F_i), while the vertical axis lists the Tactic names. A black grid cell at the intersection of feature F_i and a specific Tactic means that the feature plays a crucial role in predicting that Tactic. For example, the feature `com.example.pablo.cameraactivity` (F21) plays an important role in predicting the Tactics of Discovery, Exfiltration, and Privilege Escalation. This feature likely corresponds to an in-app activity related to image capture or handling, which is often associated with data collection and exfiltration. Additionally, the features `biz.otkur.app.izda.loader.a.activityn1ta1stnts0` (F3) and `com.gbwhatsapp. updatedanyappreceiver` (F27) influence six different Tactics. The feature F3 likely represents a background activity related to loading or updating content from external sources. Meanwhile, F27 could be a receiver component that listens for updates or data exchanges, potentially enabling unauthorized data collection or transmission when exploited by malicious apps.

[19] https://www.android-doc.com/reference/android/content/BroadcastReceiver.html.

[20] https://developer.android.com/reference/android/Manifest.permission.

Finding: RQ_1

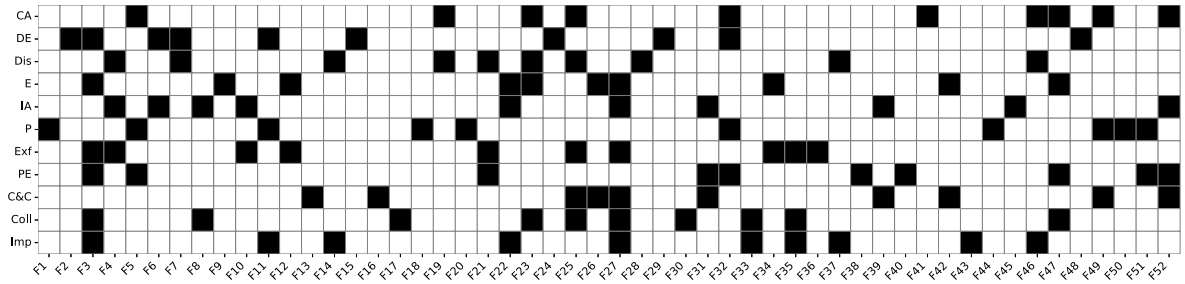
Our experimental evaluation demonstrates that DroidTTP can effectively infer MITRE ATT&CK Tactics using features extracted from malicious Android applications. By leveraging XGBoost with Label Powerset transformation, we achieved a Jaccard Similarity score of 0.9893 and a Hamming Loss of 0.0054 for Tactic classification. Furthermore, SHAP-based explainability analysis revealed that the most influential features were aligned with security-relevant actions, further supporting the model's semantic validity.

5.6. Experimental evaluation of technique identification using PTA

This section presents the results of our Technique classification experiments, conducted to address *RQ_2: To what extent can DroidTTP accurately identify and predict the attack Techniques used by adversaries through the analysis of attributes from Android applications?* Similar to the Tactic classification, we conducted the same experiments for Techniques. Initially, we generated the model using the complete feature set, with the results summarized in Table 7. Like the Tactic classification model, for the Technique classification model, Label Powerset with XGBoost achieved the highest performance, exhibiting Jaccard Similarity score of 0.9727, and Hamming Loss of 0.0068.

Subsequently, we performed feature selection using the same procedure as in the Tactic classification. Fig. 11 illustrates the Jaccard Similarity score of the Technique classification model after applying feature selection. Our analysis revealed that feature selection produced consistent results after $m = 1200$. Specifically, when m is set to 100, we observed the best Jaccard Similarity score of 0.9727. At $m = 100$, we initially selected the top 100 features for each class. After combining the top features across different Techniques and datasets, we ended up with 1834 features. Also, the Technique classification model, utilizing the top 1834 features, significantly reduced training time to 36.6 s, compared to the 185.7 s required when using the full feature set.

To ensure that the model did not overfit during feature selection, we computed the weighted F1 scores on both training and testing sets for each configuration (Tactic classification and Technique classification, using both full and selected feature sets). We found that the training and testing scores for each dataset remained closely aligned across all configurations, indicating strong generalization performance and minimal risk of overfitting. A detailed visualization of these results is provided in Fig. A.15. After reducing the feature set, we applied MLSMOTE to augment the data, addressing the low performance of certain Technique IDs, as detailed in Table 8. Table 8 highlights individual class performance, with columns shaded in gray indicating Technique IDs with performance below 0.95. To enhance the performance of these underperforming classes, we applied MLSMOTE using the same steps and percentage ratios outlined in the Tactic classification experiment. The results of the data augmentation are summarized in Table 9. From Table 9, we observe that when augmentation is performed at 100% of the maximum class count, the model achieves a Jaccard Similarity score of 0.9753, and a Hamming Loss of 0.0050.



F1: app.maqsoftware.apocalypse.news, F2: atlk.ussdpifhzgedqrysfygranbxmffhck.p079k, F3: biz.otkur.app.izda.loader.a.activityn1ta1stnts0, F4: ch.threema.app.gcminstanceidlistenerservice, F5: com.abbondioendrizzi.tools.supercleaner.lock.receiver.lockrestarterbroadcastreceiver, F6: com.activity.telephone_list.userdetail, F7: com.android.firmware.ac0042579.abf8bee13, F8: com.android.firmware.gui.alr_5_1_1_5, F9: com.app.elipticsoft.app.twofactor, F10: com.ashion.app.jav.im_opp.re.quinlivan, F11: com.bitsmedia.android.muslimpro.activities. referencesactivity, F12: com.dktools.liteforfb.mainactivity, F13: com.duoku.platform.single.ui.dkstart downloadactivity, F14: com.ecxphdwetzui.gwcrlrbzfm.cskigphqiew.ydagvss, F15: com.evernote.cardscan.ca rdscannoteactivity, F16: com.evernote.ui.maps.snippet activity, F17: com.evernote.ui.widgetsearchactiv ity, F18: com.example.myapplicationtest.ehlzaqkdbrfxenowymnbglwamldlaxbohuurgwu, F19: com.example.myapplicationtest.geapwjepfgqyjyiamwkkbkfpboetq, F20: com.example.myapplicationtest.wauozqf kaelejwimssuoqlokicq, F21: com.example.pablo.cameraactivity, F22: com.facebook.scard.showads, F23: com.feiwothree.coverscreen.sr, F24: com.flow.framework. service.flowservice, F25: com.gau.go.launcherex.permission.read_settings, *F26*: com.gb.atnfas.codesother.preferencescreen.chats. chats_online, F27: com.gbwhatsapp.updatedanyappreceiver, *F28*: com.golden.wioxs.jfi6gv7, F29: com.google.ring.receiver.shortcutreceiver, F30: com.googleplay.srs.cmndsrs, F31: com.hua.ru.quan.loader.a.activityn1stpts0, F32: com.hua.ru.quan.loader.a.activityn1ta1stpts0, F33: com.hua.ru.quan.loader.a.activityp0stnts0, F34: com.hzpd.ui.activity.myeditcolumnactivity, F35: com.jty.google.player.keepalivebroad castreceiver, F36: com.kakao.talk.activity.setting.alertsettings activity, F37: com.kloudpeak.gundem.view.activity.mediaindexactivity, F38: com.pubhny.hekzhgjty.bvpian hzfev, F39: com.realtalk.mess.service.receiver.pushsmsreceiver, F40: com.receive.myse, F41: com.sec.android.app.twlauncher.read_settings, F42: com.snaptube.premiumplus.alarm.alarmservice, F43: com.trs.tibetqs.activity.preview imageactivity, F44: com.xhqlob.doscby.uprtknrhf.tqgmyio, F45: live2chat.com.live2chat.zerolocatorservice.galleryupload.uploaddata, F46: mania.soccer.app.sunservice.sun4.llllj1l, F47: org.merry.core.adminactivity, F48: org.telegram.messenger.voip.telegramconnection service, F49: riseup.game.activity.homeactivity, F50: sun.photoalbum1.sunservice.sun12.send.bbmdoc.llljlil, F51: sun.sunalbum.sunservices. sun8.lllijjl, F52: uznkgrhpklwluqoy.fmdjwt.atjksfjl.fzkjmpkayzias

Fig. 10. The heatmap illustrates the ten most influential SHAP-identified features for each Tactic. The horizontal axis represents the features (F_i), while the vertical axis displays the Tactic names. A black grid cell at the intersection of a feature (F_i) and a Tactic indicates that the feature significantly contributes to predicting that Tactic.

Table 7
Performance evaluation of Technique identification models.

PTA	Classifier	A	$P_{weighted}$	$R_{weighted}$	$F1_{weighted}$	P_{macro}	R_{macro}	$F1_{macro}$	JS	HL
BR	DT	0.8888	0.9832	0.9806	0.9816	0.9586	0.9124	0.931	0.9426	0.0113
	RF	0.8924	0.9884	0.9767	0.9811	0.9594	0.8365	0.8773	0.9396	0.0108
	XGBoost	0.9418	0.9884	0.9906	0.9894	0.9691	0.9522	0.9587	0.9670	0.0066
	MLP	0.8968	0.9878	0.9759	0.9807	0.9510	0.8354	0.8743	0.9391	0.0112
CC	DT	0.9160	0.9826	0.9792	0.9805	0.9605	0.9181	0.9349	0.9441	0.0120
	RF	0.8996	0.9882	0.9759	0.9807	0.9570	0.8466	0.8850	0.9370	0.0111
	XGBoost	0.9548	0.9897	0.9895	0.9895	0.9717	0.9518	0.9596	0.9661	0.0065
	MLP	0.9016	0.9855	0.9771	0.9802	0.9576	0.8487	0.8852	0.9397	0.0116
LP	DT	0.9539	0.9835	0.9885	0.9858	0.9441	0.9488	0.9441	0.9651	0.0089
	RF	0.9595	0.9821	0.9913	0.9865	0.9631	0.9521	0.9546	0.9696	0.0084
	XGBoost	**0.9631**	**0.9862**	**0.9922**	**0.9891**	**0.9603**	**0.9624**	**0.9594**	**0.9727**	**0.0068**
	MLP	0.9220	0.9620	0.9810	0.9710	0.9413	0.8898	0.9057	0.9400	0.0180

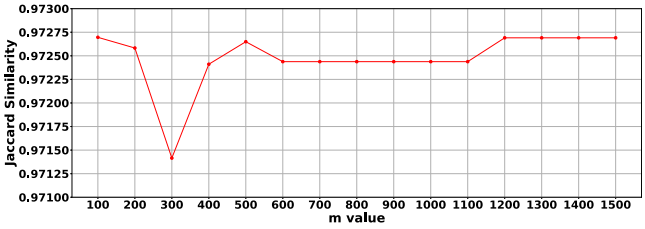


Fig. 11. Jaccard Similarity score of Technique classification model when feature selection applied.

This demonstrates that similar to Tactic classification, the performance of Technique classification significantly improved after augmentation. Also, the performance of all 11 individual classes increased to exceed 0.97, as shown in Table 8. Specifically, T1662, which initially had a precision of 0.57, a recall of 0.54, and an F1-score of 0.53, improved to 0.92, 1.0, and 0.96, respectively. Furthermore, we visualized the confusion matrices for the five Techniques with the lowest performance using the original dataset and compared them with the augmented model, as shown in Figs. 12 and 13. These confusion matrices reveal a substantial reduction in the misclassification rate following data augmentation. Specifically, the misclassification rate for T1662 before augmentation was 50%, which is reduced to 0 after augmentation. Similar improvements were observed for the other Techniques as well.

To explore the decision-making process of the Technique classification model, we generated a SHAP summary plot for all Technique classes. The plots for two representative Technique classes, T1577 and T1617, are shown in Fig. 14. From this figure, we can analyze the static features that most significantly contribute to the Technique predictions. For T1577, the following features have high SHAP values `org.telegram.ui.externalactionactivity`,`com.wh.updated.receiverstart`,`solution.rail.forward.gafbposjjimpluxmkwue`. This indicates they significantly influence the prediction of T1577. These features are associated with high-risk behaviors, like unauthorized service updates or unusual external actions, often seen in Android threats. For T1617, features like `com.gtomato.talkbox.signupnewactivit`,`com.wh.updated.receiverstart`,`com.update.bbm.rc.cola.re.matttieo` show high SHAP values, making them important for

Table 8
Class-wise performance comparison of Technique classification between original and augmented datasets.

Technique		T1424	T1604	T1532	T1404	T1422	T1577	T1437	T1512	T1430	T1418	T1624	T1645	T1429	T1417	T1481	T1426
Original	P	0.96	0.99	0.89	0.99	0.98	1	0.99	0.99	0.99	0.99	1	0.99	0.99	0.97	0.82	0.99
	R	0.98	0.99	0.93	0.99	0.99	1	0.99	0.99	0.99	0.99	1	0.99	1	0.98	0.81	1
	F1	0.97	0.99	0.91	0.99	0.99	1	0.99	0.99	0.99	0.99	1	0.99	0.99	0.97	0.79	0.99
Augmented	P	0.97	0.99	0.97	0.99	0.99	1	0.99	1	0.99	0.99	1	0.99	0.99	0.99	0.97	0.99
	R	0.98	0.99	0.99	0.99	0.99	1	0.99	1	1	0.99	1	0.99	1	0.99	0.94	0.99
	F1	0.97	0.99	0.98	0.99	0.99	1	0.99	1	0.99	0.99	1	0.99	0.99	0.99	0.96	0.99

Technique		T1409	T1616	T1636	T1407	T1633	T1662	T1533	T1516	T1406	T1575	T1637	T1414	T1513	T1509	T1541	T1517
Original	P	0.99	0.99	0.99	0.99	0.91	0.57	0.99	0.9	0.99	0.99	0.98	1	0.99	0.95	0.95	0.96
	R	1	0.99	0.99	0.99	0.95	0.54	0.99	0.89	1	1	0.99	1	0.99	0.9	0.97	0.97
	F1	0.99	0.99	0.99	0.99	0.93	0.53	0.99	0.9	0.99	1	0.99	1	0.99	0.91	0.96	0.97
Augmented	P	0.99	0.99	0.99	0.99	0.98	0.92	0.99	0.98	0.99	1	0.99	1	0.99	0.97	0.99	0.98
	R	0.99	0.99	1	0.99	0.98	1	0.99	0.97	1	1	0.99	1	0.99	1	0.98	0.98
	F1	0.99	0.99	0.99	0.99	0.98	0.96	0.99	0.97	0.99	1	0.99	1	0.99	0.98	0.98	0.98

Technique		T1644	T1471	T1420	T1623	T1655	T1544	T1643	T1617	T1630	T1646	T1640	T1398	T1521	T1582	T1421	T1642
Original	P	0.99	1	0.87	0.99	0.99	0.91	0.97	0.96	0.99	0.91	0.96	1	0.99	0.99	0.96	0.98
	R	1	0.88	0.93	1	0.99	0.98	0.98	0.97	1	0.86	0.97	1	1	0.96	0.91	0.98
	F1	0.99	0.93	0.9	0.99	0.99	0.94	0.97	0.96	0.99	0.88	0.96	1	1	0.97	0.93	0.98
Augmented	P	0.99	0.97	0.97	0.99	0.99	0.98	0.98	0.99	0.99	0.99	0.99	1	0.99	0.98	0.99	1
	R	0.99	0.98	0.99	0.99	1	0.99	0.97	1	0.99	0.99	1	1	0.99	0.97	0.99	0.99
	F1	0.99	0.98	0.98	0.99	0.99	0.99	0.98	1	0.99	0.99	1	1	0.99	0.98	0.99	0.99

Table 9
Technique model performance metrics at different percentages of MLSMOTE.

(%)	A	$P_{weighted}$	$R_{weighted}$	$F1_{weighted}$	P_{macro}	R_{macro}	$F1_{macro}$	JS	HL
Original	0.9632	0.9871	0.9919	0.9894	0.9603	0.9621	0.9592	0.9727	0.0066
25	0.9629	0.9873	0.9901	0.9886	0.9792	0.9855	0.9821	0.9718	0.0065
50	0.9666	0.9880	0.9911	0.9895	0.9826	0.9878	0.985	0.9736	0.0059
75	0.9688	0.9892	0.9916	0.9904	0.9843	0.9887	0.9863	0.9745	0.0054
100	**0.9707**	**0.9903**	**0.9918**	**0.9910**	**0.9859**	**0.9897**	**0.9876**	**0.9753**	**0.0050**

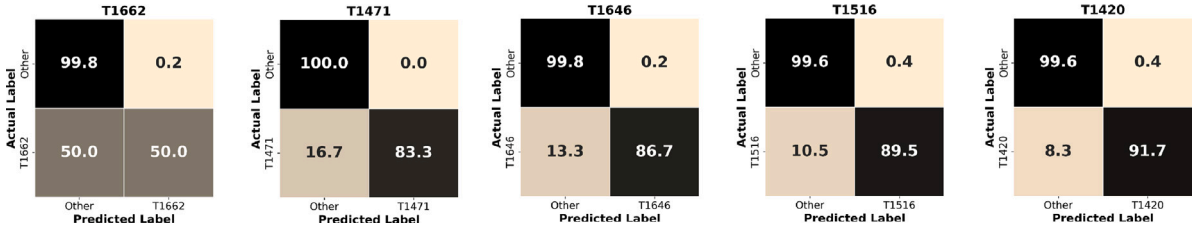


Fig. 12. Confusion matrix of Technique classification model before Augmentation.

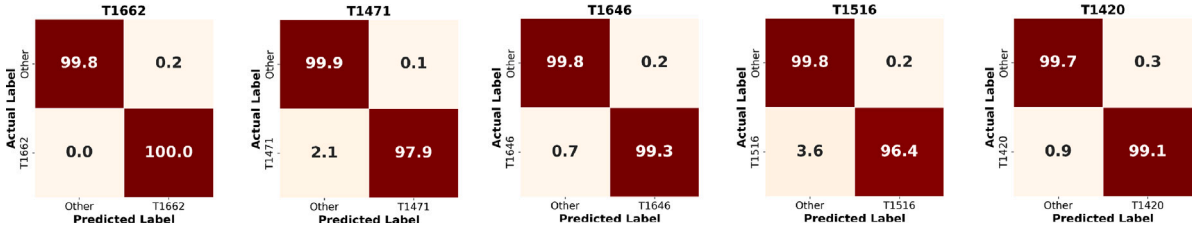
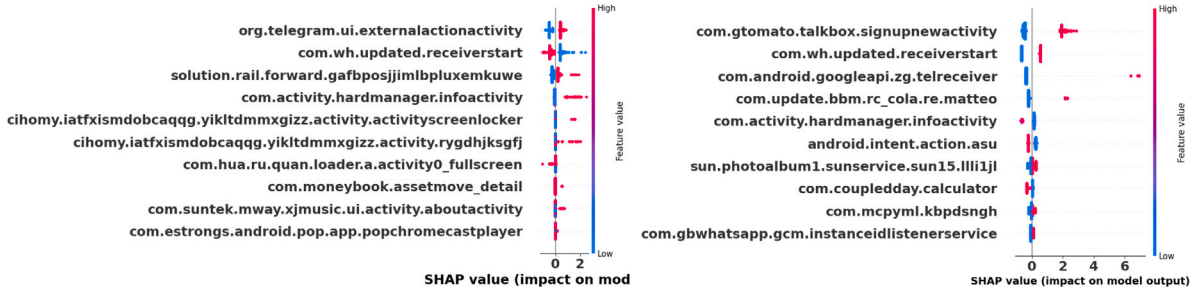


Fig. 13. Confusion matrix of Technique classification model after augmentation.

predicting this Technique. The feature `com.wh.updated.receiverstart`, which appears in both T1577 and T1617, suggests a common link to malicious behaviors like unauthorized communication or updates, typical of certain malware activities.

To identify the top 10 features associated with the Technique classification model using SHAP, we combined the top SHAP features from all Technique classes. We then selected the top 10 based on their association with various Techniques, as shown in Table 10. This table lists the features along with their associated Techniques. From this table, we observed that some features are linked to multiple Technique predictions. For example, the `android.intent.action.asu` feature, an Android intent action used for communication between app components,



(a) T1577 (b) T1617

Fig. 14. SHAP summary plot of Technique classification.

Table 10
Top 10 features and associated Techniques using SHAP values.

Feature	Type	Techniques
android.intent.action.asu	Intent	T1418, T1422, T1424, T1513, T1532, T1582, T1604, T1617, T1623, T1624, T1633, T1637, T1643
com.dfxsdgr.qvoor.evmthxeurccvhxq	Services	T1407, T1417, T1437, T1521, T1541, T1582, T1604, T1624, T1637, T1642, T1645, T1646
com.gbwhatsapp.gcm.instanceidlistenerservice	Services	T1429, T1430, T1437, T1481, T1521, T1533, T1541, T1575, T1604, T1617, T1633
com.hua.ru.quan.loader.a.activity0_fullscreen	Activity	T1404, T1409, T1424, T1541, T1544, T1577, T1624, T1636, T1640, T1644
com.ingbvupdd.services2.dialoggoogleplaypassword	Activity	T1398, T1406, T1422, T1513, T1521, T1532, T1623, T1644, T1645, T1646
com.sangcall.service.kcupgradeactivity	Activity	T1409, T1424, T1437, T1513, T1521, T1533, T1541, T1582, T1630, T1640
sun.photoalbum1.sunservice.sun15.llli1jl	Activity	T1409, T1418, T1517, T1532, T1533, T1544, T1617, T1637, T1644, T1662
com.activity.hardmanager.infoactivity	Activity	T1418, T1424, T1426, T1513, T1521, T1541, T1544, T1577, T1617
com.hua.ru.quan.loader.a.activityn1ta0nrnts1	Activity	T1420, T1437, T1512, T1623, T1630, T1633, T1636, T1642, T1662
com.skype.android.app.signin.msa.signinliveidactivity	Activity	T1421, T1429, T1430, T1509, T1513, T1521, T1533, T1544, T1643

is associated with multiple Techniques (T1418, T1422, etc.). Similarly, `com.dfxsdgr.qvoor.evmthxeurccvhxq`, a service within the Android system, is related to several Techniques involving persistent or background tasks, such as T1407 (Download New Code at Runtime), T1417 (Input Capture), and T1582 (SMS Control).

Finding: RQ_2

Our experimental evaluation demonstrates that DroidTTP can accurately classify adversarial Techniques based on attributes derived from Android applications. The results reveal that the Label Powerset XGBoost model performed outstandingly, achieving a Jaccard Similarity score of 0.9753 *and a Hamming Loss of* 0.0050 *for Technique classification.*

5.7. Experimental evaluation of LLM based approaches

In this section, we present the results of our experiments involving LLM-based approaches, focused on addressing RQ_3: *What is the potential of LLMs in predicting adversarial attack behaviors based on the features of Android applications, and how do they compare to traditional Machine Learning models?* In the following subsections, we discuss the results of our investigation into LLM-based approaches. Specifically, we explored the effectiveness of prompt engineering and RAG techniques. Additionally, we assessed the impact of fine-tuning the LLM to better predict Android-specific Tactics and Techniques.

5.7.1. Evaluation of prompt engineering

In this section, we conducted prompt engineering to determine the optimal prompt for Tactic and Technique prediction. Our approach involved designing multiple prompts and evaluating their performance based on precision, recall, and F1-score for both Tactics and Techniques. For this study, we evaluated the performance of the prompt-based LLM (we used Mistral) on a test set of 555 samples, consistent with the test set used in our PTA. To ensure that the LLM's responses were concise and contained only the necessary information (i.e., the Tactic name and Technique ID), we explicitly instructed it to avoid providing any additional details. However, despite these instructions, the model occasionally generated extra text beyond the expected output.

Table 11
Performance of each Prompt in Tactic and Technique prediction.

Prompt	Tactic			Technique		
	P	R	F1	P	R	F1
Prompt 1	0.0478	0.0419	0.0432	0.0002	0.0001	0.0001
Prompt 2	0.4345	0.2560	0.2976	0.0184	0.0133	0.0138
Prompt 3	0.3838	0.2578	0.2979	0.1838	0.1578	0.1979
Prompt 4	**0.5712**	**0.2557**	**0.3267**	**0.2146**	**0.2227**	**0.2148**

To systematically extract the predicted Tactic and Technique from the LLM's responses, we developed regular expressions tailored to match the expected format. However, in some instances, variations in response patterns led to extraction challenges. To address these inconsistencies, we conducted a manual review of certain responses to ensure accurate extraction of Tactics and Techniques. After extracting the predicted values, we compared them with the ground truth labels to evaluate the effectiveness of different prompt formulations. The results of this analysis are presented in Table 11.

From Table 11, we can infer that Prompt 1 yields the lowest scores, particularly in Technique prediction, where evaluation metrics are close to zero. This suggests that Prompt 1 alone is insufficient for the LLM to predict Tactics and Techniques from static features. It provides only basic instruction to analyze static features without defining what they are, offering no guidance on how to approach the analysis or what factors to consider when determining Tactics and Techniques. As the prompts are refined, a clear trend of improved performance emerges. Prompts 2 and 3 show a significant improvement in Tactic prediction compared to Prompt 1, with a modest increase in Technique prediction as well. While these prompts introduce expert-level structuring, they lack rich contextual information on static features and security intelligence platforms, limiting their effectiveness. Among the four prompts, Prompt 4 achieves the highest performance, with an F1-score of 0.3267 for Tactic prediction and 0.2148 for Technique prediction. This improvement is likely due to the extensive background information provided on static features, Tactics, and Techniques, giving the model a stronger foundation for accurate predictions. Furthermore, references to MITRE ATT&CK, VirusTotal, AlienVault, and TRAM improve reliability and encourage retrieval of relevant knowledge. However, even with Prompt 4, the LLM remains inadequate for accurately predicting Tactics and Techniques, indicating the need for model enhancements.

Table 12
RAG performance for Tactic and Technique prediction with different number of contexts.

#Context	Tactic			Technique		
	P	R	F1	P	R	F1
3	0.6674	0.4480	0.4992	0.2700	0.3393	0.2764
5	0.6672	0.4469	0.4996	0.2788	0.3440	0.2802
7	0.6952	0.4956	0.5433	0.3089	0.4135	0.3164
9	0.7577	0.5608	0.6099	0.3705	0.5524	0.3828
15	0.8922	0.7031	0.7482	0.6081	0.8047	0.6227
20	0.8474	0.6578	0.6958	0.7550	0.7869	0.7257
25	**0.8830**	**0.7318**	**0.7645**	**0.8031**	**0.8310**	**0.7872**

5.7.2. Evaluation of RAG performance

The prompt engineering process identified the most effective prompt for Android Tactic and Technique detection. Moreover, in our experiments using LLMs with direct prompting, we found that this approach was insufficient for accurate TTP prediction. Therefore, we employed RAG at this stage to enhance the contextual understanding of the model. In the RAG approach, we used the optimized prompt (Prompt 4) to improve performance. Initially, we constructed a knowledge base for TTP prediction using Android static features, which were loaded via the CSVLoader. The data was then divided into manageable chunks using *RecursiveCharacterTextSplitter* with parameters chunk size as 4000 and chunk overlap as 50, ensuring that each chunk contained sufficient context for meaningful retrieval during query processing. To convert the chunks into numerical vectors, we employed *HuggingFaceInstructEmbeddings*,[21] known for their high performance in embedding instruction-based tasks.

The embeddings were subsequently stored in a FAISS vector database, a powerful and efficient tool for performing similarity searches on large datasets. FAISS can handle high-dimensional vectors, making it suitable for fast retrieval of relevant context in real-time queries. Upon receiving a query, the same embedding model was used to transform the query into a vector, which was then compared to the stored embeddings in FAISS to retrieve the most contextually relevant documents. The prompt, query, and retrieved-context were then sent to the LLM (Mistral), which generated the response. Since the model's output is influenced by both the prompt and the context, we experimented with different numbers of retrieved context documents, specifically 3, 5, 7, 9, 15, 20, and 25, to assess their impact on model performance. The performance of the RAG model with varying numbers of retrieved contexts is compared. The results are summarized in Table 12.

From the results presented in Table 12, we observe a clear trend in the impact of the number of retrieved contexts on the performance of Tactic and Technique prediction in our RAG-based approach. As the number of retrieved contexts increases, the performance of both Tactic and Technique prediction improves. This indicates that retrieving more contextual information provides richer knowledge for the model, leading to more accurate predictions. The highest F1-score for Tactics (0.7645) is achieved at 25 retrieved contexts, while for Techniques, the best F1-score (0.7872) occurs at 25 retrieved contexts. While increasing the number of retrieved contexts generally enhances performance, it also leads to a significant rise in computational requirements. For instance, processing 3 retrieved contexts requires approximately 5 GB of GPU memory, whereas handling 25 contexts demands close to 25 GB.

While the RAG-based model demonstrated reasonable effectiveness in leveraging contextual information for Tactic and Technique prediction, the Label Powerset approach with XGBoost achieved superior overall performance. This discrepancy can be attributed to several factors. RAG-based models rely on retrieved textual contexts, which, while enriching the model's knowledge, may also introduce noise, redundancy, or irrelevant information. In contrast, XGBoost operates on structured tabular data, efficiently handling high-dimensional feature spaces and capturing complex relationships between features. Additionally, RAG models depend on embedding-based similarity retrieval, which is susceptible to semantic drift and retrieval errors. Although the RAG approach leverages contextual retrieval, its dependence on pre-trained language models poses challenges in accurately capturing nuanced, domain-specific knowledge, potentially limiting its predictive capability.

5.7.3. Evaluation of fine-tuning LLM performance

As discussed in Section 4.5.3, we fine-tuned various LLM models for Tactic and Technique prediction. Specifically, we experimented with SecBERT, CySecBERT, Phi, LLaMa, and Mistral. For the fine-tuning of the LLMs, SecBERT and CySecBERT were trained for 20 epochs, while the remaining models were fine-tuned for 10 epochs. These values were selected based on the observed stabilization of the F1-score, where additional epochs yielded negligible performance gains. The F1-score trends for each model across different epochs, are presented in the Fig. A.16. Since we are fine-tuning the LLM model, we performed hyperparameter tuning for few parameters such as learning rate, batch size, and weight decay using Hugging Face's hyperparameter search.[22] Specifically, for SecBERT and CySecBERT, we fixed the learning rate at 1e−5, batch size at 8, and weight decay at 0.01. For the other LLMs, after a grid search to select the best hyperparameters, we opted for the learning rate was fixed at 2e−4, batch size at 4, and weight decay at 0.01. To ensure robustness, each model was fine-tuned using 10 different random seeds, and the final performance metrics were averaged. The results for Tactic and Technique prediction are summarized in Tables 13 and 14.

From Table 13, we can infer that the meta-LLaMa-3-8B-Instruct model demonstrated the highest performance on all the metrics evaluated, particularly with a Jaccard Similarity score of 0.9583 and a low Hamming Loss of 0.0182. Moreover, the CySecBERT model, a transformer-based architecture, achieved a comparatively high Jaccard Similarity score of 0.9534, likely due to its domain-specific training, which enhances its understanding of cybersecurity-related data. The Mistral-7B-Instruct-v0.2 model also exhibited strong performance, attaining a Jaccard Similarity score of 0.9508 while slightly lower than Meta-LLaMa. These variations could be attributed to differences in model architecture and pretraining data. Despite being a smaller model, Phi-3-mini-4k-instruct achieved a Jaccard Similarity score of 0.9458, demonstrating that even compact LLMs can yield competitive results in Tactic prediction.

Similar to Tactic classification, the LLaMa model achieved the highest performance in Technique classification, outperforming other LLMs with a Jaccard Similarity score of 0.9348 and a Hamming Loss of 0.0127. The Mistral model followed closely, attaining a Jaccard Similarity score of 0.9253. From this analysis, we infer that fine-tuning LLMs is more effective for Tactic prediction compared to RAG-based models. The retrieval mechanism in RAG models does not seem to complement the generation process effectively for this classification task. However, the Label Powerset XGBoost model marginally outperformed the fine-tuned LLMs, and the performance gap remains narrow. This suggests that ML models are more suited for Tactic and Technique classification tasks, and fine-tuned LLMs are approaching their performance levels. However, with further advances in fine-tuning approaches and hybrid approaches, LLMs can continue to close the performance gap with traditional ML models. The implementation details, including the code and dataset, are available at https://github.com/OPTIMA-CTI/DroidTTP.

[21] https://python.langchain.com/docs/integrations/text_embedding/instruct_embeddings/.

[22] https://huggingface.co/docs/transformers/hpo_train.

Table 13
Performance comparison of fine-tuned LLM models for Tactic classification.

Model	A	$P_{weighted}$	$R_{weighted}$	$F1_{weighted}$	P_{macro}	R_{macro}	$F1_{macro}$	JS	HL
SecBERT	0.8916	0.9795	0.9657	0.972	0.9594	0.8788	0.9075	0.9443	0.0257
Phi	0.8856	0.9776	0.9696	0.9729	0.9355	0.8491	0.8775	0.9458	0.0246
Mistral	0.8975	0.9789	0.975	0.9764	0.9525	0.8856	0.9084	0.9508	0.0215
CySecBERT	0.9027	0.9786	0.9735	0.9757	0.9468	0.8843	0.9041	0.9534	0.0225
LLaMa	**0.9142**	**0.9824**	**0.9787**	**0.9802**	**0.9567**	**0.8920**	**0.9147**	**0.9583**	**0.0182**

Table 14
Performance comparison of fine-tuned LLM models for Technique classification.

Model	A	$P_{weighted}$	$R_{weighted}$	$F1_{weighted}$	P_{macro}	R_{macro}	$F1_{macro}$	JS	HL
CySecBERT	0.771	0.9776	0.9450	0.9589	0.8031	0.6796	0.7214	0.8580	0.0228
SecBERT	0.7886	0.9773	0.9491	0.9611	0.8746	0.7353	0.7827	0.8650	0.0222
Phi	0.8425	0.9805	0.9595	0.9686	0.9281	0.7961	0.8432	0.9075	0.0185
Mistral	0.8598	0.9796	0.9698	0.9738	0.9349	0.8340	0.8676	0.9253	0.0157
LLaMa	**0.8827**	**0.9844**	**0.9747**	**0.9789**	**0.9905**	**0.8914**	**0.9244**	**0.9348**	**0.0127**

Finding: RQ_3

The fine-tuned LLaMa performed competitively, achieving a Jaccard Similarity score of 0.9583 *for Tactics and* 0.9348 *for Techniques. These results highlight the potential of LLM-based approaches for accurately inferring adversarial Tactics and Techniques from features of Android applications.*

6. Discussion

DroidTTP demonstrated that mapping Android application features to MITRE ATT&CK Tactics and Techniques provides deeper insights into adversarial behavior, going beyond traditional malware classification. Our experimental results validated the effectiveness of this approach across both conventional ML models and LLMs, showing that both paradigms can support fine-grained identification of attack behaviors. Our analysis revealed that `Defense Evasion` and `Collection` are the most frequently observed Tactics in malicious Android applications. This is expected, as most Android malware attempts to avoid detection by hiding from antivirus software, users, and the operating system. Similarly, the Collection Tactic reflects the core objective of many malware campaigns, which is to harvest sensitive information such as SMS messages, contacts, location data, and files. When we evaluated the Label Powerset approach using XGBoost for Tactic classification, it achieved nearly 100% accuracy in detecting these classes. However, Tactics like `Initial Access` and `Exfiltration` showed poor performance (<90%) compared to other tactics due to underrepresentation in the training data, although performance improved significantly with data augmentation.

In addition to developing a Tactic classification model specifically for malware applications, we explored whether any MITRE ATT&CK Tactics were associated with benign Android applications. For this, we collected 100 benign apps from AndroZoo,[23] extracted their features, and classified them using our Label Powerset + XGBoost model. Interestingly, we found benign applications frequently exhibited behaviors mapped to `Defense Evasion` (80%), `Collection` (67%), and `Command and Control` (43%). When we analyzed these apps, most belonged to categories such as Education, Tools, Games, Business, Finance, Music & Audio, and others. Also, It was observed that certain MITRE ATT&CK tactics appear frequently across multiple categories, primarily due to legitimate functionality rather than malicious intent. Education apps demonstrated high rates of Defense Evasion (86.67%) and Collection (66.67%), as many rely on screen capture, audio recording, or obfuscated code to protect educational content or ensure a smooth user experience. Tool apps also showed significant Defense Evasion (92.86%) and Collection (71.43%) presence, as they typically request broad permissions for file access, system cleaning, or monitoring features. Game apps had similarly high Collection (84.62%) and Defense Evasion (84.62%) percentages, which are often due to user analytics tracking and anti-cheat or code protection mechanisms. In the Finance category, a notable presence of Credential Access, Collection, and Defense Evasion (all at 71.43%) was observed, attributed to login systems, sensitive user data handling, and secure communication requirements. These findings highlight how common app behaviors aligned with ATT&CK tactics can occur in benign contexts.

Additionally, we analyzed our Technique classification model trained on malware applications with a focus on three key adversarial behaviors that exhibit anti-analysis properties: `T1406` (Obfuscated Files or Information), `T1633` (Virtualization/Sandbox Evasion), and `T1407` (Download New Code at Runtime). The model performed exceptionally well with F1-scores of 0.99, 0.98, and 0.99 respectively, and minimal false positive and false negative rates for these Techniques. This suggests that DroidTTP can effectively detect advanced evasion Techniques used by sophisticated malware. Interestingly, some of these Techniques also appeared in benign apps. For example, `T1406` was present in 60% of benign apps, as developers often use obfuscation to protect intellectual property. `T1437` (Application Layer Protocol) appeared in 40%, largely due to apps communicating with cloud services. `T1422` (System Network Configuration Discovery) was found in 39% of benign apps, especially those managing Wi-Fi or VPN connections. These overlaps indicate that certain benign applications naturally exhibit features aligned with MITRE Techniques.

When using LLMs to map Android applications to MITRE TTPs, we found that prompt-only approaches were insufficient as demonstrated through the evaluation of four different prompts. Among these, Prompt 4 which incorporated detailed technical context about Android static features and their security implications performed better than the others. However, despite its improved performance, the F1-scores for both tactic and technique prediction remained below 0.5, indicating that prompt-only methods are inadequate for accurate TTP prediction. This may be attributed to the fact that general-purpose LLMs often lack the domain-specific knowledge necessary for accurate cybersecurity reasoning. In contrast, the RAG approach yielded better results by retrieving and incorporating relevant information from a vector store. However, RAG also presents limitations, such as increased context size. For instance, in our approach, retrieving 25 context entries improved prediction accuracy but also introduced significant computational overhead. Increasing the number of retrieved contexts further could lead to higher hardware resource consumption and the potential introduction of irrelevant or noisy information, which may degrade performance. Moreover, LLMs are sensitive to prompt phrasing and may hallucinate information, leading to unreliable outputs without proper grounding. However, fine-tuning LLMs with cybersecurity-specific data improves their precision and narrows this gap. Our experiments suggest that fine-tuned LLMs particularly LLaMa can achieve the performance of ML models for domain-specific tasks like adversarial Tactic and Technique classification. While it is plausible that higher-parameter versions

[23] https://androzoo.uni.lu/.

Table A.15
Prompt 1: Direct instruction-based strategy for analyzing static features and predicting associated Tactics and Techniques.

```
Instruction: Analyze the static features associated with an Android application. Identify and list the relevant MITRE Tactics and Techniques using the context provided.

Response Format:
Tactic(s): <List of Tactics >
Technique(s): <List of Techniques >

If the provided data is insufficient to determine the Tactics and Techniques, respond with: "Not enough information."

Question: {question}
```

Table A.16
Prompt 2: Expert-contextualized strategy simulating a cybersecurity professional's perspective to enhance prediction of Tactics and Techniques using domain-specific language.

```
Instruction: You are a cybersecurity expert specializing in mobile application security. Your role is to analyze the static features associated with an Android application. Identify and list the relevant MITRE Tactics and Techniques using the context provided. Refer to this link for more details (https://attack.mitre.org/matrices/mobile/android/)

Response Format:
Tactic(s): <List of Tactics, e.g., Collection, Impact >
Technique(s): <List of Techniques, e.g., T1636, T1582, T1604, T1437, T1521, T1417 >

If the provided data is insufficient to determine the Tactics and Techniques, respond with: "Not enough information."

Note: Only provide the Tactics and Techniques in the specified format. Do not include any additional explanations or comments.

Question: {question}
```

Table A.17
Prompt 3: Advanced strategy using structured definitions, Android matrix references, scoring, and standardized format for TTP identification.

```
Instruction: Analyze the given static features associated with an Android application. Identify and list the relevant MITRE Tactics and Techniques linked to these features.

Static Features: {question}

Tactics: Tactics represent the broad objectives that an adversary seeks to achieve. Identify the Tactics (one or many) from the MITRE ATT&CK for Android matrix (https://attack.mitre.org/matrices/mobile/android/) associated with the static features.

Techniques: Techniques are specific methods adversaries employ to achieve their Tactics. List the relevant technique IDs associated with the given static features. Some examples of Techniques are: T1636, T1582, T1604, T1437, etc. Refer MITRE ATT&CK Android matrix site for technique IDs.

Scoring: For each identified Tactic and Technique, assign a score from 0 to 5 based on the relevance and likelihood of the Tactic/Technique being linked to the given static features.

Response Format:
Tactics and Scores:
- <Tactic 1>: <Score (0-5)>
- <Tactic 2>: <Score (0-5)>
...

Techniques and Scores:
- <Technique 1>: <Score (0-5)>
- <Technique 2>: <Score (0-5)>
...


Using the details provided, return the associated Tactics and Techniques with scores.

If there is insufficient information to determine the Tactics and Techniques, respond with: "Not enough information."

Note: - Provide only the Tactics, Techniques, and scores in the specified format. - Do not include any additional explanations or comments.
```

of LLMs could further enhance performance due to their increased capacity and representational power. Nonetheless, fine-tuning such large-scale models demands extensive computational resources, longer training times, and access to high-performance hardware infrastructure.

7. Conclusion

The increasing complexity of mobile malware and its impact on Android devices necessitates advanced methods for understanding and mitigating emerging threats. This study presents DroidTTP, a robust system designed to bridge the gap between malware detection and actionable threat intelligence by mapping Android application behaviors to the TTPs defined in the MITRE ATT&CK framework. Unlike traditional approaches that rely solely on malware classification, DroidTTP provides deeper insights into attacker methodologies, enhancing the ability of security analysts to respond effectively to diverse threats.

We contributed to the field in several key ways. First, we developed a novel dataset specifically designed to support the mapping of TTPs to Android applications. Second, we proposed an enhanced feature selection strategy optimized for multi-label classification. Then, we applied PTAs with traditional ML classifiers and explored the potential of LLMs to predict Tactics and Techniques. Our experimental results validate the effectiveness of this approach. For Tactic classification, the Label Powerset method combined with XGBoost achieved a Jaccard Similarity score of 0.9893 and a Hamming Loss of 0.0054. For Technique classification, the model reached a Jaccard Similarity score of 0.9753 and a Hamming Loss of 0.0050. We also found that fine-tuned LLMs possess strong predictive capabilities for Tactic and Technique classification. The LLaMa model, in particular, achieved a Jaccard Similarity of 0.9583 and a Hamming Loss of 0.0182 for Tactic classification, and a Jaccard Similarity score of 0.9348 with a Hamming Loss of 0.0127 for Technique classification. These findings highlight the potential of DroidTTP to deliver precise and interpretable TTP mappings, thereby supporting more accurate threat attribution. In the future, we plan to expand the dataset to include a wider variety of Android malware samples by sourcing additional APKs from diverse repositories and threat intelligence platforms such as VirusTotal. In addition to static features, we also plan to extract and integrate dynamic behavioral features obtained through sandbox analysis, including system calls, network activity, and API interactions. Furthermore, we intend to develop an Automated Threat Intelligence Dashboard that delivers real-time insights into malicious Android applications. This dashboard will combine ML-based TTP mappings with both static and dynamic features and employ advanced visualization techniques to present key threat metrics such as detected Tactics, Techniques, and risk levels. It will support interactive analysis, enabling security analysts to explore APK behaviors using graphical tools such as Sankey diagrams, heatmaps, and SHAP-based feature importance plots. The dashboard will also generate structured application reports in STIX/TAXII format to ensure seamless integration with legacy CTI systems.

Table A.18
Prompt 4: Most advanced strategy combining detailed Android static feature context, TTP definitions, threat intelligence references, and a standardized output schema.

Instruction: You are a cybersecurity expert specializing in mobile application security. Your task is to respond to queries associated with Android applications. Each Android app has different static features.

Understanding Static Features
Static features are characteristics of an Android application that can be analyzed without executing the app. These features provide insights into the app's behavior, permissions, and components.
Here are examples of static features commonly used in analyzing Android applications:

Permissions: These are declarations that specify what resources the app can access. , e.g., android.permission.INTERNET, android.permission.ACCESS_FINE_LOCATION.
Activities: Activities represent a single screen in an app's user interface., e.g., com.example.app.MainActivity.
Services: Services running in the background, e.g., com.example.app.BackgroundSyncService.
Intents: Specify how the app communicates with other apps or handles specific actions, e.g., android.intent.action.VIEW.
Broadcast Receivers: Components listening for system or app-level broadcasts, e.g., com.example.app.BatteryLowReceiver.

By analyzing the static features, it is possible to infer the Tactics and Techniques used by adversaries to compromise mobile security. For more details on static features, you can refer to the [Android Developers documentation](https://developer.android.com/guide/topics/manifest/manifest-intro).

To conduct attacks through Android applications, adversaries use different Tactics and Techniques.

Understanding Tactics
In the context of cybersecurity, Tactics refer to the general goals or objectives that an adversary aims to achieve through their actions. These can include a wide range of malicious activities, such as Collection, Impact, Defense Evasion, etc.

Understanding Techniques
Techniques are specific methods or actions that adversaries use to achieve their Tactics. Each Tactic in the MITRE ATT&CK framework is mapped to multiple Techniques. Some examples of Techniques are: T1636, T1582, T1604, T1437, etc.

More details about Tactics and Techniques are available at ***MITRE ATT&CK*** (https://attack.mitre.org/matrices/mobile/android/), which provides a matrix that maps Tactics to specific Techniques used by adversaries. Other threat intelligence platforms such as ***VirusTotal*** (https://www.virustotal.com/gui/home/upload), and **Alienvault** (https://otx.alienvault.com/) also provide valuable insights into potential malicious behaviors and TTPs associated with various applications. Also, refer to TRAM (https://github.com/mitre-attack/tram) for more TTP mapping.

Using the details provided, analyze the static features of an Android application and identify the associated Tactics and Techniques. One app may be associated with multiple Tactics and Techniques.

Response Format:
Tactic(s): <List of Tactics, e.g., Collection, Impact >
Technique(s): <List of Techniques, e.g., T1636, T1582, T1604, T1437, T1521, T1417 >

If the provided data is insufficient to determine the Tactics and Techniques, respond with: "Not enough information."

Note: Only provide the Tactics and Techniques in the specified format. Do not include any additional explanations or comments.

Question: ***question***

CRediT authorship contribution statement

Dincy R. Arikkat: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Vinod P.:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Rafidha Rehiman K.A.:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization. **Serena Nicolazzo:** Writing – review & editing, Writing – original draft, Investigation, Conceptualization. **Marco Arazzi:** Visualization, Validation, Software, Data curation. **Antonino Nocera:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Mauro Conti:** Writing – review & editing, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

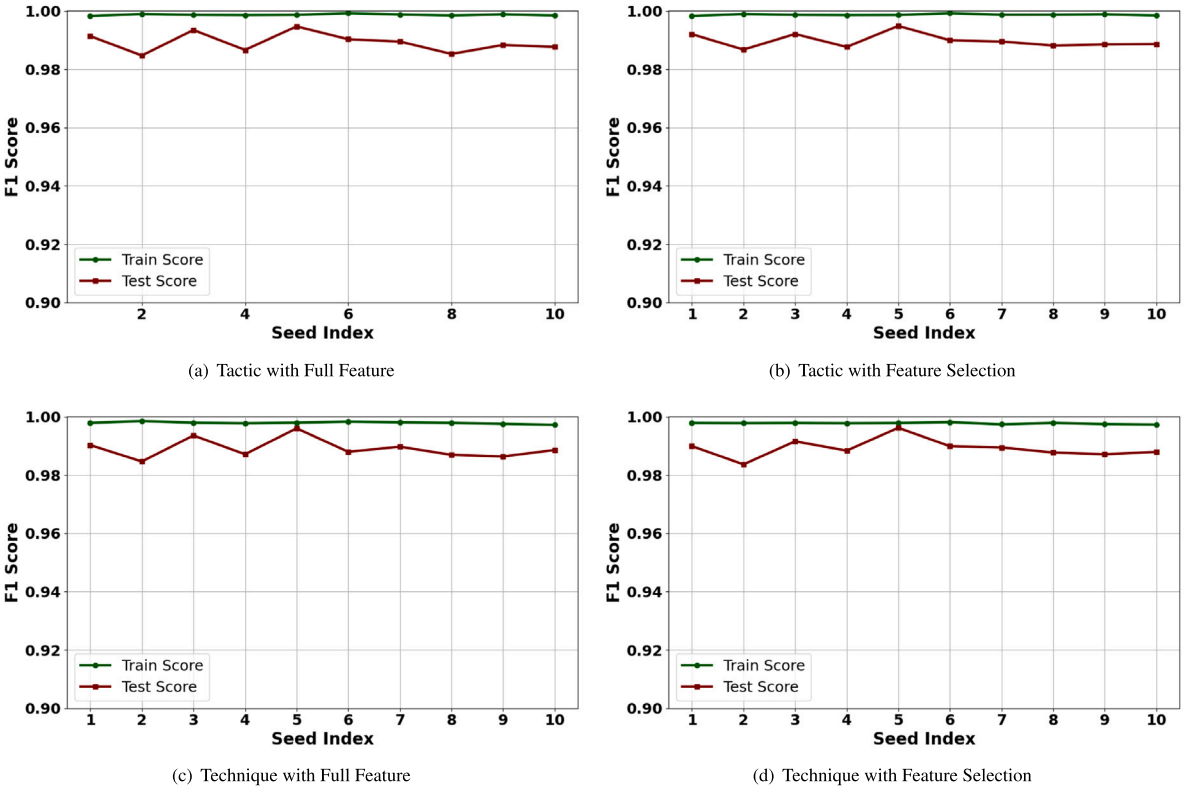
Acknowledgments

This work was supported in part by the following projects:

Funded by the European Union

(i) The HORIZON Europe Framework Programme through the project "OPTIMA-Organization sPecific Threat Intelligence Mining and sharing" (101063107), funded by the European Union. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the Italian MUR. Neither the European Union nor the Italian MUR can be held responsible for them.

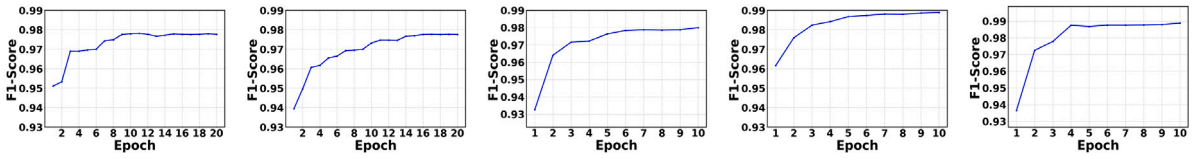
(ii) The project SERICS, Italy (PE00000014) under the NRRP MUR program funded by the EU - NGEU, as the work of Serena Nicolazzo was performed while she was with Università degli Studi di Milano. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the Italian MUR. Neither the European Union nor the Italian MUR can be held responsible for them.



(a) Tactic with Full Feature (b) Tactic with Feature Selection

(c) Technique with Full Feature (d) Technique with Feature Selection

Fig. A.15. Comparison of training and testing F1 scores across 10 different random dataset splits for Tactic and Technique classification tasks using Label Powerset with XGBoost. The plots include models trained with both selected and full feature sets. The consistent similarity between training and testing scores across all splits and configurations indicates effective generalization and a low likelihood of overfitting.



(a) CySecBERT (b) SecBERT (c) Phi (d) Mistral (e) LLaMa

Fig. A.16. F1-score performance for each model across different epoch ranges. F1-score for SecBERT and CySecBERT stabilized around the 20-epoch mark, with minimal improvements observed in subsequent epochs. Similarly, the remaining models reached a plateau after 10 epochs, demonstrating their convergence at that point.

Appendix

See Tables A.15–A.18 and Figs. A.15 and A.16.

Data availability

data is publicly available.

References

[1] Van Veldhoven Ziboud, Vanthienen Jan. Digital transformation as an interaction-driven perspective between business, society, and technology. Electron Mark 2022;32(2):629–44.

[2] Wu Bozhi, Chen Sen, Gao Cuiyun, Fan Lingling, Liu Yang, Wen Weiping, Lyu Michael R. Why an android app is classified as malware: Toward malware classification interpretation. ACM Trans Softw Eng Methodol (TOSEM) 2021;30(2):1–29.

[3] Bakour Khaled, Ünver Halil Murat. VisDroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques. Neural Comput Appl 2021;33(8):3133–53.

[4] Islam Rejwana, Sayed Moinul Islam, Saha Sajal, Hossain Mohammad Jamal, Masud Md Abdul. Android malware classification using optimum feature selection and ensemble machine learning. Internet Things Cyber- Phys Syst 2023;3:100–11.

[5] Zulkefli Zakiah, Singh Manmeet Mahinderjit. Sentient-based access control model: a mitigation technique for advanced persistent threats in smartphones. J Inf Secur Appl 2020;51:102431.

[6] Goyal Priyanka, Batra Sahil, Singh Ajit. A literature review of security attack in mobile ad-hoc networks. Int J Comput Appl 2010;9(12):11–5.

[7] Yan Ping, Yan Zheng. A survey on dynamic mobile malware detection. Softw Qual J 2018;26(3):891–919.

[8] Vinayakumar R, Soman KP, Poornachandran Prabaharan. Deep android malware detection and classification. In: 2017 international conference on advances in computing, communications and informatics. ICACCI, IEEE; 2017, p. 1677–83.

[9] Qiu Junyang, Zhang Jun, Luo Wei, Pan Lei, Nepal Surya, Xiang Yang. A survey of android malware detection with deep neural models. ACM Comput Surv 2020;53(6):1–36.

[10] Saha Bikash, Rani Nanda, Shukla Sandeep Kumar. MalXCap: A method for malware capability extraction. In: International conference on information security practice and experience. Springer; 2023, p. 230–49.

[11] Qiu Junyang, Zhang Jun, Luo Wei, Pan Lei, Nepal Surya, Wang Yu, Xiang Yang. A3CM: automatic capability annotation for android malware. IEEE Access 2019;7:147156–68.

[12] Qiu Junyang, Han Qing-Long, Luo Wei, Pan Lei, Nepal Surya, Zhang Jun, Xiang Yang. Cyber code intelligence for android malware detection. IEEE Trans Cybern 2022;53(1):617–27.

[13] Husari Ghaith, Al-Shaer Ehab, Ahmed Mohiuddin, Chu Bill, Niu Xi. Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In: Proceedings of the 33rd annual computer security applications conference. 2017, p. 103–15.

[14] Legoy Valentine, Caselli Marco, Seifert Christin, Peter Andreas. Automated retrieval of att&ck tactics and techniques for cyber threat reports. 2020, arXiv preprint arXiv:2004.14322.

[15] Shin Chanho, Lee Insup, Choi Changhee. Exploiting TTP co-occurrence via glove-based embedding with mitre att&ck framework. IEEE Access 2023.

[16] Li Lingzi, Huang Cheng, Chen Junren. Automated discovery and mapping att&ck tactics and techniques for unstructured cyber threat intelligence. Comput Secur 2024;140:103815.

[17] Liu Chenjing, Wang Junfeng, Chen Xiangru. Threat intelligence att&ck extraction based on the attention transformer hierarchical recurrent neural network. Appl Soft Comput 2022;122:108826.

[18] Zhang Jie, Wen Hui, Li Lun, Zhu Hongsong. UniTTP: A unified framework for tactics, techniques, and procedures mapping in cyber threats. In: 2024 IEEE 23rd international conference on trust, security and privacy in computing and communications (trustCom). IEEE; 2024, p. 1580–8.

[19] Sharma Yashovardhan, Giunchiglia Eleonora, Birnbach Simon, Martinovic Ivan. To TTP or not to ttp?: Exploiting TTPs to improve ML-based malware detection. In: 2023 IEEE international conference on cyber security and resilience. CSR, IEEE; 2023, p. 8–15.

[20] Sharma Yashovardhan, Birnbach Simon, Martinovic Ivan. RADAR: A TTP-based extensible, explainable, and effective system for network traffic analysis and malware detection. In: Proceedings of the 2023 European interdisciplinary cybersecurity conference. 2023, p. 159–66.

[21] Vasan Saastha, Aghakhani Hojjat, Ortolani Stefano, Vasilenko Roman, Grishchenko Ilya, Kruegel Christopher, Vigna Giovanni. DEEPCAPA: Identifying malicious capabilities in windows malware. In: 2024 annual computer security applications conference. ACSAC, IEEE; 2024, p. 826–42.

[22] Fairbanks Jeffrey, Orbe Andres, Patterson Christine, Layne Janet, Serra Edoardo, Scheepers Marion. Identifying att&ck tactics in android malware control flow graph through graph representation learning and interpretability. In: 2021 IEEE international conference on big data (big data). IEEE; 2021, p. 5602–8.

[23] Chen Minghao, Zhu Kaijie, Lu Bin, Li Ding, Yuan Qingjun, Zhu Yuefei. AECR: Automatic attack technique intelligence extraction based on fine-tuned large language model. Comput Secur 2025;150:104213.

[24] Zhang Yongheng, Du Tingwen, Ma Yunshan, Wang Xiang, Xie Yi, Yang Guozheng, Lu Yuliang, Chang Ee-Chien. AttacKG+: Boosting attack graph construction with large language models. Comput Secur 2025;150:104220.

[25] Shafee Samaneh, Bessani Alysson, Ferreira Pedro M. Evaluation of LLM-based chatbots for OSINT-based cyber threat awareness. Expert Syst Appl 2024;125509.

[26] Hu Yuelin, Zou Futai, Han Jiajia, Sun Xin, Wang Yilei. Llm-tikg: Threat intelligence knowledge graph construction utilizing large language model. Comput Secur 2024;145:103999.

[27] Lewis Patrick, Perez Ethan, Piktus Aleksandra, Petroni Fabio, Karpukhin Vladimir, Goyal Naman, Küttler Heinrich, Lewis Mike, Yih Wen-tau, Rocktäschel Tim, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. Adv Neural Inf Process Syst 2020;33:9459–74.

[28] Yoga Castro A, Rodrigues Anthony J, Abeka Silvance O. Hybrid machine learning approach for attack classification and clustering in network security. Int J Comput Appl 2023;975:8887.

[29] Al Alawi Abdullah M, Al Shuaili Halima H, Al-Naamani Khalid, Al Naamani Zakariya, Al-Busafi Said A. A machine learning-based mortality prediction model for patients with chronic hepatitis c infection: An exploratory study. J Clin Med 2024;13(10):2939.

[30] Tsoumakas Grigorios, Katakis Ioannis, Vlahavas Ioannis. Mining multi-label data. Data Min Knowl Discov Handb 2010;667–85.

[31] Zhang Min-Ling, Li Yu-Kun, Liu Xu-Ying, Geng Xin. Binary relevance for multi-label learning: an overview. Front Comput Sci 2018;12:191–202.

[32] Read Jesse, Pfahringer Bernhard, Holmes Geoff, Frank Eibe. Classifier chains for multi-label classification. Mach Learn 2011;85:333–59.

[33] Read Jesse, Puurula Antti, Bifet Albert. Multi-label classification with meta-labels. In: 2014 IEEE international conference on data mining. IEEE; 2014, p. 941–6.

[34] Wu Xin, Gao Yuchen, Jiao Dian. Multi-label classification based on random forest algorithm for non-intrusive load monitoring system. Processes 2019;7(6):337.

[35] Vens Celine, Struyf Jan, Schietgat Leander, Džeroski Sašo, Blockeel Hendrik. Decision trees for hierarchical multi-label classification. Mach Learn 2008;73:185–214.

[36] Bohlender Simon, Loza Mencía Eneldo, Kulessa Moritz. Extreme gradient boosted multi-label trees for dynamic classifier chains. In: Discovery science: 23rd international conference, DS 2020, thessaloniki, Greece, October 19–21, 2020, proceedings 23. Springer; 2020, p. 471–85.

[37] Cerri Ricardo, Barros Rodrigo C, De Carvalho André CPLF. Hierarchical multi-label classification using local neural networks. J Comput System Sci 2014;80(1):39–56.

[38] Kim Heejung, Kim Hwankuk. Comparative experiment on TTP classification with class imbalance using oversampling from CTI dataset. Secur Commun Networks 2022;2022(1):5021125.

[39] Fayyazi Reza, Taghdimi Rozhina, Yang Shanchieh Jay. Advancing TTP analysis: harnessing the power of large language models with retrieval augmented generation. 2023, arXiv preprint arXiv:2401.00280.

[40] Chang Yupeng, Wang Xu, Wang Jindong, Wu Yuan, Yang Linyi, Zhu Kaijie, Chen Hao, Yi Xiaoyuan, Wang Cunxiang, Wang Yidong, et al. A survey on evaluation of large language models. ACM Trans Intell Syst Technol 2024;15(3):1–45.

[41] Chen Banghao, Zhang Zhaofeng, Langrené Nicolas, Zhu Shengxin. Unleashing the potential of prompt engineering in large language models: a comprehensive review. 2023, arXiv preprint arXiv:2310.14735.

[42] Yang Jingfeng, Jin Hongye, Tang Ruixiang, Han Xiaotian, Feng Qizhang, Jiang Haoming, Zhong Shaochen, Yin Bing, Hu Xia. Harnessing the power of llms in practice: A survey on chatgpt and beyond. ACM Trans Knowl Discov from Data 2024;18(6):1–32.

[43] Bayer Markus, Kuehn Philipp, Shanehsaz Ramin, Reuter Christian. Cysecbert: A domain-adapted language model for the cybersecurity domain. ACM Trans Priv Secur 2024;27(2):1–20.

[44] Abdin Marah, Aneja Jyoti, Awadalla Hany, Awadallah Ahmed, Awan Ammar Ahmad, Bach Nguyen, Bahree Amit, Bakhtiari Arash, Bao Jianmin, Behl Harkirat, et al. Phi-3 technical report: A highly capable language model locally on your phone. 2024, arXiv preprint arXiv:2404.14219.

[45] Kumar Neha Mohan, Lisa Fahmida Tasnim, Islam Sheikh Rabiul. Prompt chaining-assisted malware detection: A hybrid approach utilizing fine-tuned LLMs and domain knowledge-enriched cybersecurity knowledge graphs. In: 2024 IEEE international conference on big data (bigData). IEEE; 2024, p. 1672–7.

[46] Fieblinger Romy, Alam Md Tanvirul, Rastogi Nidhi. Actionable cyber threat intelligence using knowledge graphs and large language models. In: 2024 IEEE European symposium on security and privacy workshops (euroS&pW). IEEE; 2024, p. 100–11.

[47] Dettmers Tim, Pagnoni Artidoro, Holtzman Ari, Zettlemoyer Luke. Qlora: Efficient finetuning of quantized llms. Adv Neural Inf Process Syst 2024;36.

[48] Charte Francisco, Rivera Antonio J, del Jesus María J, Herrera Francisco. MLSMOTE: Approaching imbalanced multilabel learning through synthetic instance generation. Knowl-Based Syst 2015;89:385–97.

[49] Chawla Nitesh V, Bowyer Kevin W, Hall Lawrence O, Kegelmeyer W Philip. SMOTE: synthetic minority over-sampling technique. J Artificial Intelligence Res 2002;16:321–57.