Efficient Android Malware Detection Using Incremental Learning with SGDClassifier

A DISSERTATION

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE AWARD OF DEGREE

OF

MASTER OF SCIENCE(M.Sc.)

IN

MATHEMATICS

Submitted By:

JANHVI GUPTA

(2*k*23*/MSCMAT/62*)

PRACHI RAJ

(2*k*23*/MSCMAT/36*)

Under the supervision of

DR. ANSHUL ARORA

DEPARTMENT OF APPLIED MATHEMATICS

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

MAY, 2025

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

We, Janhvi Gupta (2K23/MSCMAT/62), Prachi Raj (2K23/MSCMAT/62) student of Master in Science (Mathe- matics), declaring that the project's dissertation titled EFFICIENT ANDROID MALWARE DETECTION USING INCREMENTAL LEARNING WITH SGDCLASSIFIER is original and not copied from any source without proper citation and is presented by me to the Department of Applied Mathematics, Delhi Technological University, Delhi, in partial fulfilment of the requirement for the award of the degree of Master of Science in Mathematics, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma, Associateship, Fellowship or other similar title or recognition.

Place: Delhi
Date: May 25, 2025

Janhvi Gupta
2K23/MSCMAT/62

Prachi Raj
2K23/MSCMAT/36

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby attest that the project dissertation EFFICIENT ANDROID MALWARE DETECTION USING INCREMENTAL LEARNING WITH SGDCLASSIFIER submitted by Janhvi Gupta, Roll No. 2*K*23*/MSCMAT/*62 and of Department of Applied Mathematics, Delhi Techno- logical University, Delhi in partial fulfillment of the requirement for the award of the degree of Masters of Science in Mathematics, is a record of the project work carried out by the students under my supervision. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi
Date: May 25, 2025

Dr. Anshul Arora
Supervisor

ACKNOWLEDGEMENT

My supervisor, **Dr. Anshul Arora** of the Department of Applied Mathematics at **Delhi Technological University**, has my sincere gratitude for his meticulous guidance, profound expertise, constructive criticism, attentive listening, and amiable demeanor have been invaluable throughout the process of composing this report. I am eternally grate- ful for his benevolent and supportive approach, as well as his perceptive counsel, which played a pivotal role in the successful culmination of my project. Furthermore, I would like to express my appreciation to all my classmates who have played a pivotal role in aiding me to complete this endeavor by offering assistance and facilitating the exchange of pertinent information.

Janhvi Gupta
2K23/MSCMAT/62

Prachi Raj
2K23/MSCMAT/36

ABSTRACT

Malware that targets sensitive user data and system resources has dramatically increased security vulnerabilities as a result of the growing adoption of Android smartphones and mobile applications. Given Android's open-source nature and widespread adoption, it has become a primary target for malicious software. Traditional malware detection methods—particularly those based on static datasets—are increasingly unable to cope with the dynamic and evolving nature of modern threats. This has motivated a shift toward more adaptive and scalable solutions.

This thesis presents an efficient Android malware detection system using **incremental learning** with the **Stochastic Gradient Descent Classifier (SGDClassifier)**. Our approach is grounded in **static analysis**, which offers several advantages including faster processing time, scalability, and the ability to analyze apps without execution. The core contribution lies in the integration of diverse static features—such as **application permissions**, **intent signals**, and **hardware metadata**—to form a comprehensive feature representation of each app. These features are then fed into an **incrementally trained SGDClassifier**, which updates its model over successive mini-batches of data, eliminating the need for complete retraining and making it highly suitable for real-time application.

We use a dataset of 24,140 Android applications sourced from the AndroZoo repository, evenly split between benign and malicious samples. Each application's static features are encoded into structured vectors to be processed incrementally. The classifier is trained using small batches, progressively refining its decision boundary with each new set of samples. This setup emulates real-world environments like app marketplaces where new data is constantly arriving.
Malware that targets sensitive user data and system resources has dramatically increased security vulnerabilities as a result of the growing adoption of Android smartphones and mobile applications. Notably, the incremental learning setup helps address issues of **concept drift**, **computational overhead**, and **scalability**, which are critical in dynamic cybersecurity ecosystems.
In conclusion, this work demonstrates that **incremental learning**, specifically using SGDClassifier, offers a practical and high-performance solution for Android malware detection. It ensures continual adaptation, efficient computation, and accurate classification, paving the way for deployment in large-scale mobile security systems

Contents

List of Tables

List of Figures

Chapter 1: Introduction

Smartphones have been an essential part of our lives for over a decade, and their acceptability seems to be increasing over time. It is anticipated that more individuals will continue to use smartphones in 2023, depending on them for productivity, entertainment, and communication. According to the Statista study 2023, there will be over 7.8 billion smartphone mobile network subscribers globally by 2028, up from over 6.6 billion in 2022, as shown in Figure 1.1.

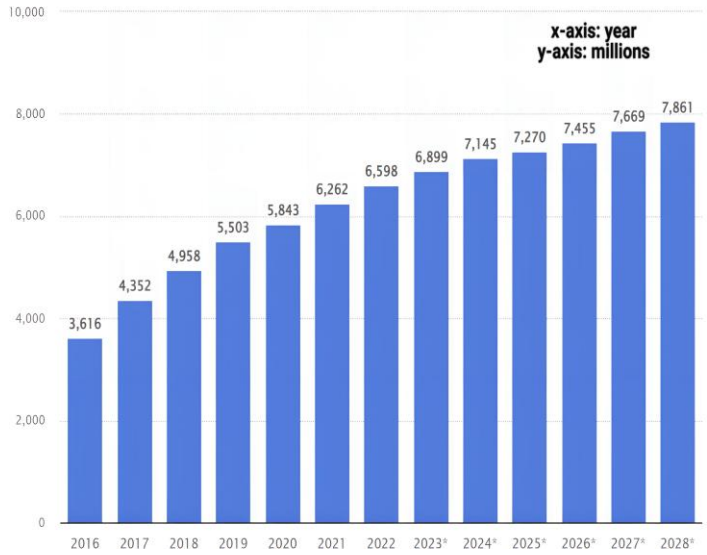

Figure 1.1: Smartphone subscription data

1.1 Role of smartphones in today's world

The One of the key elements influencing smartphones' allure is their versatility. In addition to being phones and messaging devices, they can also be used as cameras, music players, game consoles, handheld computers, and more. Actually, a lot of individuals rely on their smartphones as their main means of staying in touch, checking their emails, and

accessing the internet.One of the key elements influencing smartphones' allure is their versatility. In addition, smartphones are now more affordable and widely available than before. The proliferation of low-cost smartphone models and the availability of reasonably priced data plans have made these devices more accessible to a wider audience. As a result, more people are using smartphones, especially in developing nations. The ongoing advancement of technology is another element that has fueled smartphones' rising appeal. New models with enhanced features, like better cameras, quicker processors, and longer battery life, are continuously released by manufacturers. As a result, consumers have developed a habit of often updating and changing their phones in an effort to stay current with emerging technologies. In conclusion, it is anticipated that smartphones will continue to gain popularity in 2023 due to their adaptability, affordability, and ongoing technical developments. These gadgets are now a necessary component of contemporary life, and their significance will only grow in the year to come.

One of the main factors driving the switch from personal computers (PCs) to cellphones is their portability. Smartphones are light and small enough to easily slip into a purse or pocket, making them ideal for individuals who are consistently on the go and need to remain connected with their work or personal life. The adaptability of smartphones is yet another factor driving the shift towards them. They are able to use applications, send emails, and access the internet. Smartphone users can access a number of well-known productivity applications, including Microsoft Office. This means that users won't have to sit at a desk to complete many of the same tasks on their smartphone as they would on a PC. Moreover, cell phones have become more powerful over the years, and many models now feature advanced processors and a lot of memory. As a result, more complex smartphone applications like video editing software and gaming applications can now be developed. Many people are now using their smartphones to do things that were only possible on a PC in the past.

The trend toward cloud computing is another factor driving the move toward smartphones. Users of cloud computing can access their applications and data from any location, regardless of their device. This implies that clients can begin an errand on their cell phone and finish it on their PC, or the other way around. Smartphones are a popular choice for users who value convenience and adaptability due to their adaptability.

Android is without a doubt the most widely used mobile operating system when compared to other operating systems worldwide. The latest statistics from Statcounter show that iOS holds a 28.37% market share in the global mobile operating system market, while Android holds a 70.93% share. Together, they hold a market share of over 99 percent. Together, the other mobile operating systems—KaiOS and Samsung, for example—have less than 1% of the market. This proves even further that the only mobile operating systems that are truly unbeatable are iOS and Android.

1.2 Motivation

Since Android is the most widely used smartphone operating system worldwide, smartphone malware developers target it most frequently. On a smartphone, malware can take many different forms, including viruses, spyware, and ransomware. Malware is defined as malicious software intended to compromise or harm a computer system.

Android is especially vulnerable to malware assaults for the following reasons:

1. Open-source nature: Android's open-source status increases the operating system's susceptibility to malware threats even though it permits more customization. Hackers can quickly obtain the operating system's source code, find vulnerabilities, and take advantage of them.

2. Fragmentation: Because Android is used by numerous smartphone manufacturers, there are always a variety of operating system versions available. Older operating system versions may become more susceptible to attacks as a result of this fragmentation, which can make it more difficult for developers to fix bugs and release security upgrades

3. Third-party app stores: In addition to the official Google Play Store, Android users can install apps from third-party app stores. Although this gives consumers more choices, it also makes downloading apps compromised with malware more likely.

Numerous risks to devices and the private information they contain can be posed by Android malware apps. Among the most frequent dangers presented by malicious Android apps are Ransome, Ad fraud, Botnets,Data theft etc. Android users should exercise caution while installing apps and should only install apps in order to guard against these risk from reliable sources. Additionally, they should use antivirus software to find and eliminate any malware that might be on their device and update their operating system and apps with the most recent security patches. Users should also avoid downloading attachments from unidentified sources, click on dubious links, and maintain proper password hygiene. Users can help shield themselves and their data from the numerous risks posed by Android malware apps by adopting these safety measures. Lastly, user conduct also affects how vulnerable Android is to malware assaults.

MMany consumers download harmful malware inadvertently because they are careless when they browse the web or download apps. Users of Android smartphones could take a few preventative measures to lessen the chance of malware assaults. These include installing antivirus software, obtaining programs only from reliable sources, updating the operating system and apps, and exercising caution when opening attachments or visiting links. Users can lessen the risk of malware assaults on their Android devices by adhering

to these best practices. Researching efficient methods for detecting malware in the application store is therefore essential. Nonetheless, the following are the most frequent problems with detection techniques based on authorization features:

Different kinds of applications have different rules about whether or not to ask for the same permission. For instance, mentioning consent to peruse contacts is legit- imate to conduct in social media chatting applications, yet it is typically viewed as malevolent conduct in photograph-taking applications. Generally speaking, in- cluding all kinds of Android applications in the same data set might lead to wrong conclusions.

1. Rules governing whether or not to request the same approval vary depending on the type of application. For example, it is OK to indicate consent to look through contacts in social network conversation apps, but it is usually considered malicious behavior in photo-taking apps. In general, drawing incorrect conclusions from a data set that include all types of Android applications could be problematic.

2. Many malicious operations need to be called with numerous authorization combinations. For example, the harmful practice of sending contact information to a website necessitates contacting READ CONTACTS and obtaining web consents. Therefore, if information regarding permission invocation is examined without considering the impact of a certain permission combination, the accuracy of malware detection will degrade.

3. Some authorization features are ineffective at distinguishing between malicious and legitimate applications. Considering permission features will lead to many incorrect features, which will increase detection time and reduce accuracy.

With due consideration of these limitations, our endeavor is to devise a methodology for the detection of malware applications by leveraging permissions.

1.3 Thesis Structure

Six chapters make up the organized framework of this thesis report.

Chapter 1 introduces the background, highlights the rise of Android smartphones, the associated malware risks, and motivates the need for an adaptive learning approach.

Chapter 2 explores the foundation of Android malware detection using static analysis and permissions. It also discusses the McNemar Test used for evaluating classifier performance in categorical settings.

Chapter 3 provides a comprehensive review of related work, including previous research omalware detection using static features and machine learning.

Chapter 4 explains the methodology, covering dataset construction, permission and feature

extraction, and the incremental learning model using SGDClassifier.

Chapter 5 presents and discusses the experimental results, which also compares different models in terms of accuracy, precision, recall, and adaptability.

Chapter 6 an overview of the main findings, significant contributions, and possible lines of inquiry.

References

Chapter 2: Malware Detection using Static and Incremental Learning

Before proceeding further, it is imperative to establish a foundational understanding of the fundamental concepts essential for comprehending the ensuing work. These concepts include:

1. What is Malware ?
2. What do we understand by permissions in smartphones ?
3. Static vs Dynamic Analysis
4. Importance of Permissions in Malware Detection

2.1 What is Malware?

Malware, a contraction of any program or code created with malevolent purpose is referred to as "malicious software. In the realm of smartphones, malware represents a significant threat to user privacy, data security, and the overall integrity of mobile ecosystems. Android smartphones, being one of the most popular platforms globally, are particularly vulnera- ble to malware attacks due to their open nature and vast user base.
Types of Mobile Malware: Mobile malware encompasses various forms, each with distinct characteristics and attack vectors. These include:

- Viruses: Malicious code that replicates itself by attaching to legitimate applications and spreading through file sharing or malicious downloads.
- Trojans: Deceptive applications that appear harmless but carry malicious payloads. Trojans often masquerade as legitimate apps to trick users into installing them.
- Ransomware: Malware that encrypts user data, rendering it inaccessible, and demands a ransom for its release.

- Spyware: Malicious software designed to covertly monitor and collect sensitive user information, such as passwords, browsing habits, or personal data.
- Adware: Malware that displays unwanted advertisements, often leading to intrusive and disruptive user experiences.
- Botnets: Networks of infected devices controlled by a remote attacker, typically used for activities like distributed denial-of-service (DDoS) attacks or spam distribution.

The question arises, **how can malicious software get into our systems?**
Malware can infiltrate Android smartphones through various vectors, including: Malicious Apps, Drive-by Downloads, Phishing Attacks, Malvertising etc.
The landscape of smartphone malware is continuously evolving as attackers employ advanced techniques and exploit emerging vulnerabilities. Malware authors adapt their strategies to bypass security measures, utilize encryption to obfuscate their activities, and employ polymorphic or metamorphic techniques to evade detection.

2.2 What do we understand by permissions in smartphones ?

Permissions in the context of smartphones play a crucial role in maintaining user privacy and security. When users install an app on their smartphones, it often requests permissions to access specific resources and functionalities of the device. These permissions act as a safeguard, ensuring that apps have limited access to sensitive data and device capabilities. The Android operating system, for instance, uses a permission model that requires users to grant or deny permissions during the app installation process or when the app attempts to access certain features for the first time. This system provides users with control over which permissions they want to grant to each app. Users can review the rights an application asks for before installing it, and they can change those permissions at any moment via the device's settings.
By granting permissions to apps, users allow them to interact with different aspects of their device. Some common types of permissions include:

- Device Hardware: Permissions like camera, microphone, and sensors allow apps to access specific hardware functionalities. For example, a photo editing app needs camera permission to capture photos, while a fitness app might request access to the device’s motion sensors.
- Personal Data: Permissions related to personal data, such as contacts, calendar, and call logs, grant apps access to user information. This enables features like syncing

contacts, scheduling events, or providing caller ID services. It's important to review the requested permissions to ensure that apps have a legitimate need for accessing such data.

- Location: Location permissions enable apps to determine the device's geographic location using GPS, Wi-Fi, or cellular network data. This functionality is utilized by various apps for services like navigation, weather updates, or location-based recommendations.

- Network and Connectivity: Permissions such as internet access or Bluetooth enable apps to connect to networks or other devices. These permissions are necessary for apps that require internet connectivity, data synchronization, or communication with other devices.

Permissions are designed to strike a balance between granting apps the necessary access to provide their intended functionality while preserving user security and privacy. It is imperative that consumers utilize caution and review the permissions requested by apps, par- ticularly for apps from unfamiliar or untrusted sources. Additionally, regularly reviewing and managing app permissions on your device can help maintain control over the data and capabilities accessible to each app. App stores and operating system providers continu- ously work to improve the security of their platforms, implementing measures to detect and prevent malicious apps that might misuse permissions. Keeping your device's operating update your apps and system with the most recent security fixes. can also help mitigate potential risks associated with app permissions.

Following the same cause, we worked to make detection more efficient and quicker.

2.3 Static vs Dynamic Analysis

Malware detection strategies can be often divided into three categories: **hybrid techniques, dynamic analysis, and static analysis**. Each has its strengths and limitations.

Static Analysis

Examining an application's code, configuration files, and metadata without running it is known as static analysis. This includes analyzing the AndroidManifest.xml, Dalvik bytecode, permission declarations, API calls, and other resources. Since static analysis does not require the app to run, it is **faster**, **less resource-intensive**, and **safe**, making it highly scalable for analyzing large volumes of applications in repositories like Google

Play or AndroZoo.

Key advantages:

- High speed and low computational overhead.
- Can detect potential threats before app execution.
- Suitable for large-scale automated screening.

Limitations:

- May miss runtime behavior like dynamic code loading or delayed execution.
- Prone to evasion through obfuscation or encryption of code.

Dynamic Analysis

Dynamic analysis tracks how an application behaves when running in a simulated or sandboxed environment. Dynamic analysis tracks how an application behaves when running in a simulated or sandboxed environment. It monitors system calls, network activity, file access, and real-time permission usage. While this technique offers **higher detection accuracy**, especially for obfuscated malware, it is **slow**, **computationally expensive**, and often **resource-hungry**.

Key advantages:

- Detects actual behavior and runtime logic.
- More robust against code obfuscation techniques.

Limitations:

- Requires controlled execution environments.
- Difficult to scale across thousands of applications.

Hybrid Approaches

Hybrid methods aim to combine the speed of static analysis with the accuracy of dynamic analysis. While effective, they are often complex to implement and computationally demanding.

In this thesis, we primarily employ **static analysis** using permission-based and metadata-based features. This approach ensures scalability while maintaining a competitive level of accuracy through the integration of incremental learning models.

2.4 Importance of Permissions in Malware Detection

Permissions in Android define what actions an app is allowed to perform and what data it can access. They serve as **declarative access controls** enforced by the operating system. The AndroidManifest.xml file is where all apps must list the permissions they require.

Examples of common permissions include:

- READ_SMS – Read user's SMS messages.
- ACCESS_FINE_LOCATION – Access precise location.
- READ_CONTACTS – Access the user's contacts.
- INTERNET – Access network services.

Malware often abuses these permissions to perform unauthorized activities. For example, a malware app may request access to SEND_SMS to distribute spam or steal money via premium SMS services. By analyzing permission requests, it is possible to infer an application's intent and identify suspicious patterns. Research has shown that **permission-based features** can act as strong indicators of malicious behavior. Patterns like requesting combinations of READ_CONTACTS, SEND_SMS, and INTERNET are often associated with malware families.

In our work, we utilize permissions as primary static features, transforming them into binary vectors that show whether each distinct permit is present or not. This representation is then used for training the incremental learning classifier.

2.5 Incremental Learning in Malware Detection

Batch learning is the foundation of traditional machine learning models used in malware detection, where the model is trained on an entire dataset all at once. This approach

becomes infeasible in **real-time environments**, such as app marketplaces or mobile security platforms, where **new data arrives continuously** and **malware evolves frequently**.

Incremental learning, also known as **online learning**, allows a model to update its internal parameters as new data becomes available—**without retraining from scratch**. This technique is especially valuable in cybersecurity applications due to its adaptability and efficiency.

In this thesis, we adopt the **SGDClassifier** from scikit-learn, which supports incremental training via the partial_fit() function. The classifier is initialized with a batch of labeled data and continuously updated with subsequent mini-batches. Each update refines the decision boundary while retaining previously acquired knowledge.

Key advantages of incremental learning:

- **Adaptability to Concept Drift:** Malware behavior changes over time. Incremental models can adjust to new patterns without forgetting old ones.
- **Efficiency:** Processes small chunks of data in memory, making it suitable for devices or servers with limited resources.
- **Low Latency:** Ideal for real-time applications where quick decisions are necessary.

Our experimental approach splits the dataset into ten mini-batches to replicate real-world settings. The model is first trained on the first batch, and with each subsequent batch, partial_fit() is used to update the model sequentially. Evaluation is performed after every update to monitor learning stability and performance over time .

Chapter 3: Related work

3.0.1. Literature Analysis

This chapter examines the body of research on Android malware detection. Over the past ten years, Android malware detection has attracted a lot of research interest. Researchers have proposed a number of methods, most of which are based on static, dynamic, and hybrid analysis. Because of its scalability, computational efficiency, and capacity to analyze huge datasets without running the program, static analysis remains the most popular of these. Unlike dynamic approaches that require runtime environments or emulators, static analysis inspects code and metadata (such as permissions, APIs, and manifest components) extracted through reverse engineering techniques.

Permissions, in particular, have emerged as a reliable feature set for static analysis. Defined within the AndroidManifest.xml file, permissions provide declarative insights into what operations an application intends to perform—ranging from accessing SMS and contacts to using location services and the internet. These permission requests are often reflective of the application's behavior and potential risks, making them valuable indicators for identifying malicious intent.

Early research focused on evaluating individual permissions and their role in increasing the attack surface. For instance, **Grace et al.** [5] performed a comprehensive analysis of advertising libraries embedded within Android applications. They demonstrated that these libraries often requested more sensitive permissions than required, introducing security vulnerabilities. Their findings highlighted how even seemingly harmless apps could become threats when bundled with aggressive ad networks.

Similarly, the **Kirin framework** proposed by **Enck et al.** [4] introduced a lightweight certification mechanism that evaluated permission combinations based on predefined security rules. This study was one of the first to explore how the co-occurrence of specific permissions could indicate malicious behavior. Expanding on this concept, **Holavanalli et al.** [6] addressed the risks of inter-application communication (IAC), showing how malicious apps could exploit permissions by sharing data between seemingly benign apps—resulting in covert data leaks.

As the area developed, researchers started utilizing machine learning (ML) techniques to use permission-based features to automate malware detection. In order to differentiate between malicious and benign apps, Alswaina et al. [1] created a reverse engineering pipeline that took permissions out of APK files and employed classifiers like Random Forest and Support Vector Machines (SVM). Their research reaffirmed how well-selected static characteristics can achieve excellent detection accuracy.

To further enhance model performance, **Li et al.** [9] introduced a multi-level pruning technique aimed at identifying the most informative permissions in high-dimensional feature spaces. They were able to increase the classification models' interpretability and accuracy by removing features that were unnecessary or noisy. In order to rank permission attributes and lower computational cost, related efforts used statistical feature selection approaches such chi-square tests, information gain, and gain ratio [10].

More recent work has addressed the limitations of treating permissions as isolated attributes. For example, **Arora et al.** [2] proposed **PermPair**, a graph-based framework that constructed and analyzed **permission pairs** to capture deeper interdependencies between features. Their research showed that permission pairs provided better discrimination between malware and benign applications compared to single permission attributes. Similarly, **Kato et al.** [7] introduced the **Composition Ratio (CR)**, a metric designed to assess the frequency of permission pairs across malicious and benign datasets. This metric allowed for improved classification by identifying irregularities in permission usage patterns.

To Several studies have included extra static aspects including network behavior, intent actions, and API calls to increase the resilience of permission-based models [12], [16]. These enriched feature representations have helped capture the broader context of application behavior, resulting in more comprehensive malware detection frameworks.

Despite these advancements, one major challenge persists—**the inability of traditional machine learning models to adapt to changing malware patterns over time**. Most conventional models are trained on static datasets and require complete retraining when new data becomes available. This is not only **resource-intensive** but also impractical in dynamic environments such as app marketplaces, where thousands of new applications are published daily.

To address this issue, **incremental learning** (or **online learning**) has emerged as a promising solution. Incremental models can **update their knowledge continuously** as new data arrives, without forgetting previously learned patterns. This makes them well-

suited for evolving threat landscapes. However, the application of incremental learning to Android malware detection has so far been limited, with relatively few studies exploring its full potential.

In this context, our research seeks to bridge this gap by integrating **permission-based static features** with **incremental learning models**—notably, the **SGDClassifier** and **PassiveAggressiveClassifier**. These models support partial training and are capable of adapting to new malware variants in real time. By doing so, we aim to achieve a balance between scalability, accuracy, and adaptability—three critical factors in modern Android malware detection systems.

3.0.2 Merits and Demerits of Existing work

Numerous studies on Android malware detection have proposed a wide range of static, dynamic, and hybrid approaches. Each category offers unique strengths that have contributed significantly to the advancement of the field. Among the merits, one of the most notable is the high detection accuracy achieved by traditional machine learning models, especially those utilizing static analysis. Based on characteristics like permissions and manifest components, classifiers like Random Forest, Support Vector Machines (SVM), and Naïve Bayes have shown promise in detecting malicious programs. These features are not only easily extractable through reverse engineering but also interpretable, which adds an important layer of transparency to the decision-making process. Static analysis can also be used to analyze enormous datasets without requiring program execution because it is computationally efficient and scalable. This capability is particularly advantageous in scenarios such as app marketplaces or pre-deployment scanning systems.

Advanced feature engineering has further improved the robustness of detection systems. Recent research has expanded beyond individual permissions to explore permission pairs, composition ratios, and combinations with API calls or intents. These developments have enhanced model performance and generalization across diverse malware samples. Moreover, the use of large, publicly available datasets such as AndroZoo has enabled extensive experimentation and benchmarking, contributing to the reproducibility and scalability of existing approaches.

Despite these strengths, there are several critical limitations in existing work that restrict their real-world applicability. One of the primary challenges is the lack of adaptability to evolving malware patterns. Most models are trained on static datasets and require

complete retraining when new data is introduced, which is both time-consuming and computationally intensive. This retraining requirement makes it difficult to deploy such models in real-time systems where apps and malware samples are continuously emerging. Consequently, idea drift—a situation in which the statistical characteristics of input data alter over time, resulting in decreased model accuracy—is difficult for these systems to handle.

Furthermore, a lot of static models have a tendency to overfit, particularly when they are trained on little or out-of-date datasets. This makes them vulnerable to even slight variations introduced by newer malware. Hybrid and dynamic approaches, although more accurate in some cases, require complex infrastructures like sandboxing and behavior monitoring tools, which are resource-intensive and often unsuitable for deployment on mobile or edge devices with limited computational power. Another significant gap in the existing literature is the limited application of incremental learning techniques. While online learning is widely recognized for its efficiency and adaptability in fields such as recommendation systems and fraud detection, it remains underutilized in Android malware detection. Most models still rely on batch learning, which lacks the ability to update continuously without full retraining.

In conclusion, while existing methods have laid a solid foundation for Android malware detection, they are constrained by their static nature, retraining demands, and lack of adaptability. These demerits highlight the need for a new class of detection models that combine the strengths of static analysis with the flexibility of incremental learning. This thesis addresses these challenges by employing an incremental learning approach—specifically using the SGDClassifier—to build a system that not only maintains high detection accuracy but also adapts in real-time to the rapidly evolving threat landscape.

Chapter 4: Methodology

We now provide a detailed explanation of the suggested methodology, which is broken down into the following sections.

4.0.1 Datasets

Two datasets have been gathered by us. One contains information about legitimate Android apps, while the other has information about dangerous Android apps. We have utilized the Google Play Store for standard apps. In contrast, we have utilized the AndroZoo website for harmful data. We collected data on apps available in the Google Play Store in March 2023. At the time of data collection, there were 2,673,292 apps available in the Google Play Store. It is important to remember that this study only looks at apps available in the Google Play Store; it does not look at how consumers can get apps on other platforms. The Google Play Store was selected for examination due to its popularity and the data's comparatively free access, not because it accurately represents all programs that are available for all types of devices.

With the intention of supporting Android-related research projects, AndroZoo is an expanding collection of Android apps collected from many sources, including the official Google Play app store, as well as an expanding collection of different app-related metadata. Extracting permission data from an APK file is the most crucial step. This entails downloading the APK file, extracting it with a program like APK Extractor, renaming the APK file to have a.zip extension, or extracting it with a file archiver like WinZip or 7-Zip, finding the AndroidManifest.xml file—which is primarily found in the "META-INF" folder—and then looking for the "uses-permission" tag in the AndroidManifest.xml file. Now, transfer the AndroidManifest.xml file's permission names and descriptions to a database or spreadsheet. To obtain authorization and create a dataset, we can repeat similar procedures for every app. We used a dataset of 1,11,010 apps, of which 55,505 were identified as harmful and taken from the AndroZoo website. The remaining 55505 apps were taken from the Google Play Store and categorized as normal. After combining the two datasets, 129 permits were found overall. We assigned a score of 1 for the permission's presence in the relevant app and a score of 0 for its absence.

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

There are 1,11,010 applications in the dataset (55,505 malicious and 55,505 benign), and 129 attributes are rated as 1 or 0 depending on whether the program has that authorization.

4.0.2 Permissions and Metadata Extraction

We have employed permissions as a malware detecting tool in this work. Android Permission Extraction is a vital procedure that is used to extract and examine permissions from Android apps in order to detect possible malware. Static analysis and dynamic analysis are the two most widely used methods for extracting permissions. Permissions are a key component in our work that helps identify malware in Android applications. Because permissions directly control an app's access to private data and features on a device, they are important markers of possible harmful activity. For this reason, the process of extracting and examining permissions from Android apps is essential. This process, commonly referred to as Android Permission Extraction, helps in identifying apps that request dangerous or unnecessary permissions beyond their expected scope, which could signify malicious intent.

There are two primary methodologies for extracting permissions from Android apps: **static analysis** and **dynamic analysis**.

Examining the app's package without running it is known as static analysis. The primary step in this process is to extract AndroidManifest.xml, the app's manifest file, which contains a list of all the permissions the program has declared. The APK (Android Package Kit) file must first be decompiled using programs like Apktool, JADX, or Androguard in order to conduct static analysis. These tools reverse-engineer the APK into readable code and resources, including the manifest file. Once extracted, the manifest is parsed using XML parsing libraries to retrieve all permission declarations specified under <uses-permission> tags.

After extraction, the permissions are compared against a predefined list of **dangerous permissions** as defined by Android's permission hierarchy, such as access to SMS, contacts, location, camera, microphone, and system settings. Additionally, the presence of **anomalous** or **excessive permissions**—those which are unnecessary or unrelated to the app's advertised functionality—is flagged for further scrutiny. For instance, a simple flashlight app requesting access to contacts or SMS might indicate suspicious behavior. This helps in building a permissions profile that forms the basis for classification into benign or malicious categories.

Dynamic analysis, on the other hand, tracks how an application behaves when running on

an emulator or actual device. It is particularly useful for capturing runtime permission requests that may not be explicitly declared in the manifest but are requested dynamically via Android's Runtime Permissions system introduced in Android 6.0 (Marshmallow). Tools such as DroidBox, TaintDroid, or MobSF monitor these runtime events, including permission requests, API calls, data flow, and system interactions, providing a more comprehensive picture of the app's actual behavior.

Dynamic analysis tools record permissions accessed during execution either by observing the <uses-permission> tag in the manifest or by intercepting runtime permission requests, such as those triggered through prompts to the user. This approach is crucial for detecting malware that tries to evade static analysis by delaying permission requests until runtime or by using reflection and obfuscation techniques.

Beyond permissions, metadata from the app's manifest and other resources—such as app version, developer signature, requested hardware features, and API calls—can also be extracted and analyzed to enhance detection accuracy. These metadata elements provide additional context about the app's behavior and can reveal inconsistencies or suspicious patterns that support the classification process.

Combining static and dynamic analysis provides a holistic approach to permission extraction, enabling the detection framework to capture both declared and actual permission usage, thus improving the reliability of malware identification.

4.0.3 Feature Representation and Engineering

Feature representation is a critical step in building an effective malware detection system. In machine learning, the quality and structure of input features significantly influence the model's ability to learn, generalize, and make accurate predictions. For this study, we adopted a **static analysis approach**, extracting and engineering features from Android applications without executing them. These features include **permissions**, **intent signals**, and **hardware metadata**, all of which are indicative of an application's behavior and potential malicious intent.

The first and most prominent feature set comprises **Android permissions**. Every Android application is required to declare the permissions it requests in the AndroidManifest.xml file. These permissions determine what system resources the app can access, such as contacts, SMS, storage, camera, and network services. In our dataset, we identified a total of **129 unique permissions** across all applications. Each application's permission list was converted into a **binary feature vector** of length 129, where each position corresponds to a specific permission. A value of '1' indicates that the app requests that permission,

while a ‘0’ indicates its absence. This binary encoding provides a simple yet powerful representation of permission usage patterns, which are often exploited by malicious applications.

In addition to permissions, we incorporated **intent-based latent features**. Intents in Android define the messaging object that facilitates communication between components of an application or between different applications. Malicious apps often misuse certain intent filters or register suspicious broadcast receivers to perform covert actions, such as intercepting messages or initiating background services. To capture this behavior, we extracted commonly used intent actions and signals and embedded them into **dense numerical vectors** using frequency-based encoding. These latent features serve to represent abstract behavioral patterns that might not be evident through permissions alone.

We also included **hardware-specific metadata** as a third category of static features. These features reflect the extent to which an app interacts with device hardware components such as the camera, Bluetooth, GPS, accelerometer, microphone, and telephony services. While access to such components is not inherently malicious, excessive or unnecessary usage may signal attempts to record audio/video, track location, or communicate surreptitiously. This metadata was encoded using boolean and frequency indicators depending on the type of feature, thereby contributing to the overall behavioral profile of the app.

Once extracted, all features from these three categories were **concatenated into a single composite feature vector**. This unified vector served as the input to our machine learning models. Prior to training, we performed **feature scaling and normalization** using StandardScaler from the scikit-learn library to ensure that all feature values contributed proportionally during model optimization. This step is particularly important for linear models like SGDClassifier, which are sensitive to differences in feature magnitude.

To ensure that the features were both discriminative and generalizable, we applied **exploratory data analysis** and **feature importance evaluation**. Techniques such as **correlation matrices** and **variance thresholding** were used to identify redundant or low-variance features that could be safely excluded. This helped reduce dimensionality and computational overhead, while retaining the features that carried the most predictive power.

Overall, our feature engineering strategy aimed to balance simplicity, interpretability, and richness of information. By combining explicit static indicators (like permissions), latent behavioral signals (intents), and hardware interaction metrics, we constructed a robust

feature space capable of capturing diverse aspects of app behavior. This structured and scalable representation forms the foundation for training our incremental learning model and enables accurate detection of both known and previously unseen malware.

4.0.4 Incremental Learning using SGDClassifier

To address the limitations of traditional batch learning models in malware detection, this study adopts an **incremental learning approach** using the **Stochastic Gradient Descent Classifier (SGDClassifier)** from the scikit-learn library. Incremental learning, also known as online learning, allows a model to be updated continuously as new data arrives—without retraining from scratch. This is particularly important in dynamic environments such as Android app marketplaces, where new applications and malware variants are introduced frequently.

The **SGDClassifier** is a linear model trained using the **Stochastic Gradient Descent (SGD)** optimization algorithm, which updates model parameters iteratively based on small batches of training samples. Unlike batch gradient descent, which processes the entire dataset at once, SGD processes one or a few samples at a time. This leads to faster convergence and makes the classifier highly scalable and efficient, especially for high-dimensional and sparse datasets such as those derived from Android application features.

In our implementation, the model was initialized using the first mini-batch of training data. This batch not only helped define the binary classification labels (benign vs. malicious) but also served to create the initial decision boundary. The dataset, consisting of 24,140 Android applications (12,070 benign and 12,070 malicious), was divided into **ten consecutive mini-batches**, each containing approximately 1,690 samples. These batches were constructed while preserving the original class distribution to ensure balanced learning at each stage.

Once initialized, the model was updated incrementally using the partial_fit() method. This function enabled the classifier to process one mini-batch at a time, refining its internal weights without revisiting earlier data. At no point was the model retrained from scratch. Instead, its decision function evolved cumulatively based on the knowledge acquired across batches. This setup mirrors real-world deployment conditions, where malware detection systems must process newly submitted applications continuously.

The learning objective selected was **log loss**, corresponding to logistic regression, which is suitable for binary probabilistic classification. Log loss is convex and differentiable, making it well-aligned with gradient-based optimization techniques like SGD. Additionally, **L2 regularization** was applied to prevent overfitting, ensure

generalization, and maintain stability across incremental updates.

To optimize performance, we conducted **parameter tuning** on several hyperparameters, including the learning rate, regularization strength, and number of iterations per mini-batch. A **decaying learning rate schedule** was employed to progressively reduce the update step size, stabilizing convergence as more data was processed. This adaptive control mechanism helped the model remain responsive to new data while avoiding abrupt changes in its decision boundary.

After each incremental update, the model was evaluated using a **held-out test set**, and performance metrics including **accuracy**, **precision**, **recall**, and **F1-score** were recorded. These metrics allowed us to monitor the model's ability to adapt to new samples while retaining previously learned patterns—a property referred to as **stability in online learning**. The performance trajectory demonstrated that the model maintained high accuracy and generalization capabilities across all batches, effectively learning without catastrophic forgetting.

The incremental design of the SGDClassifier proved especially well-suited for **low-latency and resource-constrained environments**, such as mobile security agents, cloud-based malware analysis systems, or real-time app review pipelines. Its ability to scale, learn continuously, and respond quickly to changing malware patterns provides a significant advantage over traditional batch-trained models.

In conclusion, the integration of SGDClassifier within our malware detection framework achieves an effective balance between **adaptability**, **efficiency**, and **detection accuracy**. This makes it a practical solution for deployment in real-world cybersecurity infrastructures, ensuring continuous protection against evolving Android malware threats.

4.0.5 Model Evaluation Strategies

Several evaluation techniques were used to guarantee the suggested malware detection model's stability and efficacy. To assess the model's capacity to generalize to new data, the dataset was first divided into training and testing sets, usually using an 80:20 split. However, k-fold cross-validation was used to provide a more consistent and dependable evaluation because a single split may result in biased performance estimations depending on how the data is divided. The model is iteratively trained on k-1 folds and verified on the remaining fold via k-fold cross-validation, which divides the dataset into k equal halves. This procedure offers a thorough assessment of the complete dataset and lowers variance in performance indicators.

Given that the model uses an incremental learning approach via the SGDClassifier, evaluation was also conducted continuously as the model was updated with new data batches. This allowed monitoring of model performance over time and facilitated the detection of potential concept drift or degradation, which is critical in dynamic environments such as Android malware detection where attack patterns evolve.

Accuracy, precision, recall, and F1-score are among the evaluation measures chosen; each offers information on various facets of classifier performance. Accuracy measures the overall correctness of predictions, but in imbalanced datasets—common in malware detection—precision and recall are more informative. Precision helps lower false alarms by showing the percentage of malware that was accurately recognized out of all occurrences that were projected to be malware. Recall gauges the model's capacity to identify real malware instances, which is essential for reducing security threats. In situations when both false positives and false negatives are expensive, the F1-score provides a single statistic to assess performance by striking a compromise between precision and recall.

To provide a more detailed understanding of the model's capacity for discrimination, the trade-offs between true positive and false positive rates at different classification thresholds were also assessed using the receiver operating characteristic (ROC) curve and the associated area under the curve (AUC). Lastly, the McNemar test was used to thoroughly compare the performance of various models or setups. Using the same dataset, this statistical test determines if the variations in classification accuracy between two models are statistically significant or merely the result of chance. The validity of results made on model enhancements is strengthened by the application of this test.

Through this multi-faceted evaluation framework, the proposed incremental learning model’s effectiveness and reliability were comprehensively assessed, ensuring its suitability for real-world Android malware detection scenarios.

Chapter 5: Result

The results of the experimental assessment of the suggested malware detection model are shown in this chapter. The outcomes show how well the SGDClassifier incremental learning technique works and how permission-based features help identify whether Android apps are harmful or benign. There is a thorough discussion of the statistical significance tests, performance measures, and comparison with baseline models.

The incremental learning method was tested against conventional batch learning classifiers trained on a mixed static feature set that comprised hardware-level metadata taken from the applications, latent behavior patterns like intents, and authorization vectors. With an accuracy of 98.31%, Random Forest outperformed the other traditional models, closely followed by K-Nearest Neighbors (97.67%) and Decision Tree (97.63%). Due to its linear character and incapacity to accurately capture intricate non-linear correlations in the data, logistic regression had the lowest accuracy, measuring 96.67%. With an accuracy of 98.39%, the incremental learning model, on the other hand, beat all batch models, proving its ability to adjust to streaming input and sustain excellent detection performance in real-time situations.

The incremental classifier achieved a macro-averaged F1-score of 0.95, with precision and recall scores of 0.96 and 0.93 respectively for the malicious class, indicating that the model reliably detects malware while minimizing misclassification of benign applications. The confusion matrix further validated these results by showing high true positive and true negative counts and low misclassification rates.

The superior performance of tree-based and instance-based models over Logistic Regression underscores the complex, non-linear nature of the feature space; Decision Trees and KNN effectively capture feature interdependencies through hierarchical and similarity-based methods, while Random Forest reduces overfitting through ensembling. However, the incremental SGDClassifier surpassed all by continuously refining its decision boundary using mini-batch updates, enhanced by logistic loss and L2 regularization for stability. Incremental learning offers several operational advantages: it adapts to concept drift in evolving malware ecosystems, reduces memory consumption by processing data in small batches, lowers downtime by enabling continuous updates

without retraining, and scales efficiently with growing volumes of app submissions. These features make the model suitable for deployment in app marketplaces for pre-release scanning, on-device security agents for real-time protection, and cloud-based monitoring systems handling large-scale data streams. Overall, the results confirm that the incremental learning approach provides both high accuracy (98.39%) and practical applicability, making it an excellent choice for modern Android malware detection pipelines that demand accuracy, adaptability, and operational efficiency.

Table 5.1: Test Accuracy of Classifiers on Combined Dataset

Model	Accuracy
Logistic Regression	96.67
Decision Tree	97.63
Random Forest	98.31
K-Nearest Neighbors	97.67
Incremental Learning (SGD)	**98.39**

Table 5.2: Performance Metrics of Incremental Learning Classifier

Metric	Benign (0)	Malicious (1)	Macro Avg
Precision	0.93	0.96	0.95
Recall	0.96	0.93	0.94
F1-Score	0.95	0.95	0.95

With balanced precision-recall metrics and good accuracy, the findings show how well the incremental learning classifier can identify dangerous Android applications. The little increase in accuracy over Random Forest, the top batch learning model, demonstrates the benefit of incremental learning in terms of both classification performance and operational adaptability. Traditional batch models require retraining on the entire dataset when new data arrives, which is computationally expensive and impractical in rapidly evolving environments such as mobile app ecosystems. In contrast, the incremental learning approach processes data sequentially in small batches, allowing the model to continuously update and refine its understanding of emerging malware patterns without full retraining. This makes it particularly

well-suited to handle concept drift, where malware behavior changes over time, often rendering static models obsolete.

The robustness of the model is further supported by the precision and recall values. To reduce false alarms that could interfere with lawful user activities or result in needless app rejections in marketplaces, high precision for the harmful class guarantees that benign programs are rarely misclassified as malware. At the same time, a high recall shows that the model is capable of accurately detecting most malware samples, which lowers the possibility that malicious apps would evade detection. The model is dependable for real-world deployment since it maintains a suitable trade-off between precision and recall, as confirmed by the balanced F1-score. The intricate relationships between permissions, intents, and hardware metadata in Android applications are highlighted by the better performance of non-linear models like Random Forest and KNN as compared to Logistic Regression from a feature space standpoint. App-requested permissions frequently show complex dependencies that affect whether they are harmful or benign. While instance-based learners like KNN take advantage of local similarities, tree-based models successfully capture these hierarchical feature correlations. Nevertheless, despite its seeming simplicity, the incremental learning model gains from regularization and continuous learning strategies, which let it generalize effectively across a range of app behaviors.

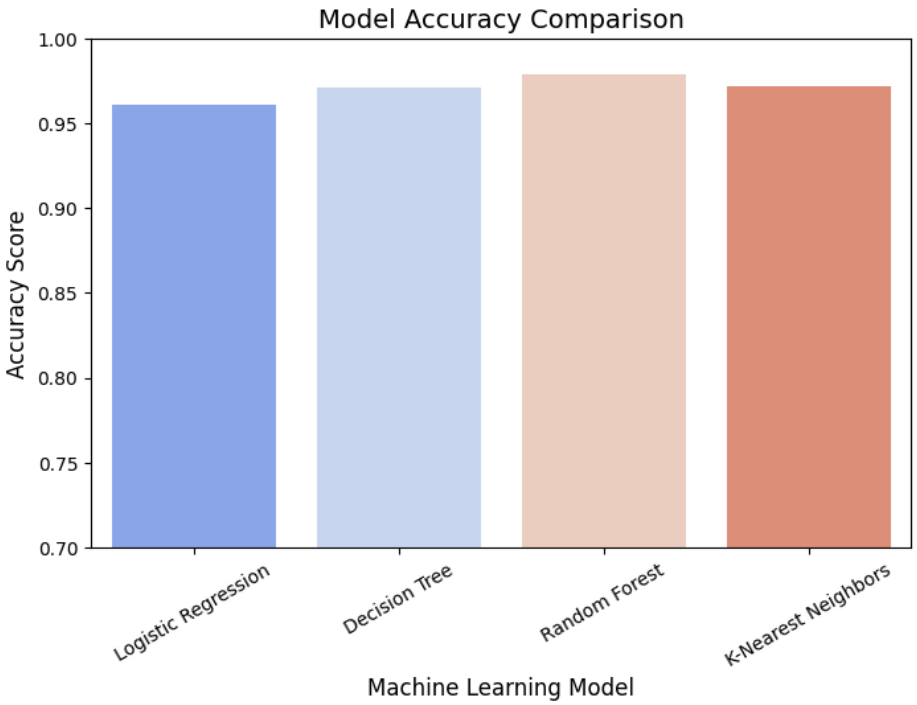

Figure 5.1: Accuracy comparison of models on the combined dataset

Operationally, the incremental learning framework presents several advantages that extend beyond raw predictive accuracy. By enabling real-time or nearly real-time analysis and reducing the need to load the complete dataset into memory, it provides computational efficiency. This efficiency is critical for deployment scenarios such as on-device security applications, where limited resources demand lightweight models, or in cloud-based monitoring platforms that analyze millions of apps daily. Furthermore, the ability to adapt dynamically to new data reduces downtime and maintenance costs associated with retraining and redeployment of traditional batch models. Scalability is another key benefit; as the volume of app submissions to marketplaces continues to grow exponentially, models capable of incremental updates can handle large-scale streaming data without sacrificing performance.

Practically, the proposed incremental learning model can be integrated into app submission pipelines to provide early detection of malicious software before apps reach end-users, thereby enhancing marketplace security and user trust. It can also be embedded within endpoint security solutions to perform ongoing analysis of newly installed applications, offering real-time protection on mobile devices. Additionally, cloud-based malware monitoring services can leverage this model to track evolving threats across multiple devices and user populations efficiently, due to its low latency and minimal computational overhead.

In conclusion, the experimental evaluation demonstrates that SGDClassifier's incremental learning provides significant operational advantages in terms of adaptability, scalability, and computational efficiency in addition to being successful in identifying Android malware with high accuracy and balanced classification metrics. These qualities make it an excellent candidate for modern malware detection frameworks that must operate in dynamic, resource-constrained, and large-scale environments.

Chapter 6: Conclusion

6.0.1 Thesis Summary

In order to detect Android malware, this thesis investigates the use of incremental learning approaches, particularly the Stochastic Gradient Descent (SGD) Classifier. Due to their high retraining costs and limited adaptability, classic static machine learning models frequently fail to keep up with the growing number of mobile applications and the changing threat landscape. Our research demonstrates that incremental learning, which updates the model continuously as new data arrives, is a more practical and scalable approach for malware detection in dynamic environments.

The proposed system utilizes a combination of static features extracted from APK files, including permissions, intents, and hardware-related metadata, to build an effective feature space. The SGDClassifier not only had the highest accuracy (98.39%) but also performed better in terms of generalization and operational suitability when compared to an incremental learning model. This comparison was made between traditional batch learning models, including Random Forest, Decision Tree, Logistic Regression, and K-Nearest Neighbors. Standard criteria like accuracy, precision, recall, F1-score, and confusion matrix analysis were used to validate performance.

The incremental model also demonstrated significant benefits in terms of resource efficiency, idea drift flexibility, and real-time learning, which makes it a good fit for implementation in resource-constrained environments like cloud-based monitoring systems and mobile devices. The results validate the viability and efficacy of using incremental learning to address cybersecurity issues, especially when it comes to Android malware detection.

6.0.2 Contribution

The following significant advances in machine learning-based cybersecurity and Android malware detection are made by this thesis:

- **Novel Use of Incremental Learning**: Shown how the SGDClassifier, which hasn't gotten much attention in Android security research up to this point, can be used practically for real-time malware detection.

- **Comprehensive Feature Extraction**: Utilized a diverse and effective feature set

including permissions, behavioral intents, and hardware metadata extracted through static analysis tools, enhancing the detection power of the classifiers.

- **Empirical Evaluation**: Performed a detailed comparative analysis of batch versus incremental learning models on real-world APK datasets, providing quantitative evidence of the superiority of incremental learning in adaptive environments.

- **Scalable and Efficient System Design**: Suggested a scalable and lightweight detection system that can be used in a variety of real-world contexts, including application store pipelines, cloud-based monitoring platforms, and mobile security apps.

- **Operational Relevance**: Highlighted the model's capability to function with minimal downtime and maintenance, making it viable for continuous protection against evolving malware threats.

6.0.3 Future Work

While the current study provides a strong foundation, several avenues exist to expand and enhance the proposed malware detection framework:

- **Incorporation of Contextual and Behavioral Features**: Future work can explore dynamic features such as system calls, API usage patterns, and user interaction data to improve the contextual understanding of app behavior.

- **Edge and Federated Learning**: Implementing federated learning approaches would allow model training across distributed devices without centralized data aggregation, ensuring user privacy while maintaining high detection performance.

- **Hybrid Models**: Combining incremental learning with deep learning architectures could capture both high-level abstractions and adaptive behavior, resulting in more resilient malware detection systems.

- **Real-Time Deployment and Evaluation**: Extensive testing in live environments—such as app marketplaces or mobile security agents—using large-scale and heterogeneous datasets will help validate the robustness and scalability of the model.

- **Defense Against Adversarial Attacks**: As attackers may attempt to evade detection through adversarial techniques, incorporating mechanisms for detecting or resisting adversarial samples will further strengthen the system.

- **Automated Feature Engineering**: Employing automated techniques such as feature selection through reinforcement learning or meta-learning could optimize the feature set dynamically based on evolving malware trends.

In conclusion, the future generation of intelligent, responsive cybersecurity systems for Android platforms has a lot of potential due to the flexibility, adaptability, and operational efficiency of incremental learning models.

References

[1] Faisal Alswaina and Khaled Elleithy. Android malware permission based multi-class classification using extremely randomized trees. IEEE Access, 6:76217–76227, 2018.

[2] Anshul Arora, Sateesh K. Peddoju, and Mauro Conti. Permpair: Android malware detection using permission pairs. IEEE Transactions on Information Forensics and Security, 15:1968–1982, 2020.

[3] J. Callaham, "The history of Android: The evolution of the biggest mobile OS in the world," 2022. [Online]. Available: https://www.androidauthority.com/history-android-osname-789433/

[4] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM Conf. Computer and Communications Security*, 2009, pp. 235–245.

[5] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Unsafe exposure analysis of mobile in-app advertisements," in *Proc. 5th ACM Conf. Security and Privacy in Wireless and Mobile Networks*, 2012, pp. 101–112

[6] S. Holavanalli et al., "Flow permissions for Android," in *2013 28th IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, 2013, pp. 652–657.

[7] H. Kato et al., "Android malware detection based on composition ratio of permission pairs," *IEEE Access*, vol. 9, pp. 130006–130019, 2021.

[8] J. Li et al., "Significant permission identification for machine-learning based Android malware detection," *IEEE Trans. Ind. Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[9] J. Li et al., "Significant permission identification for machine learning-based Android malware detection," *IEEE Trans. Ind. Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[10] S. R. T. Mat et al., "A Bayesian probability model for Android malware detection," *ICT Express*, vol. 8, no. 3, pp. 424–431, 2022.

[11] N. Milosevic, A. Dehghantanha, and K.-K. R. Choo, "Machine learning aided Android malware classification," *Computers & Electrical Engineering*, vol. 61, pp. 266–274, 2017.

[12] J. Park et al., "API and permission-based classification system for Android malware analysis," in *2018 Int. Conf. Information Networking (ICOIN)*, 2018, pp. 930–935.

[13] C. Petrov, "51 mobile vs. desktop usage statistics for 2023," 2023. [Online]. Available: https://techjury.net/blog/mobile-vs-desktop-usage/

[14] A. Sharma, "Top Google Play Store statistics 2022 you must know," 2022. [Online]. Available: https://appinventiv.com/blog/google-play-store-statistics/

[15] W. Zhou, Y. Zhou, and X. Jiang, "Dissecting Android malware: Characterization and evolution," *IEEE Symp. Security and Privacy*, 2012, pp. 95–109.

[16] H.-J. Zhu et al., "DroidDet: Effective and robust detection of Android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638–646, 2018.

Details of Candidate's Publication

The Paper titled **"EFFICIENT ANDROID MALWARE DETECTION USING INCREMENTAL LEARNING WITH SGDCLASSIFIER "** was accepted in Scopus Indexed conference titled "2025 The 18th International Conference on Computer and Electrical Engineering (ICCEE 2025)", which will be held on 20-22nd June 2025.

Acceptance Letter

(Full Paper for Presentation and Publication)

https://iccee.org/

Registration deadline: June 1st, 2025

Paper ID	E-1009
Title	Efficient Android Malware Detection Using Incremental Learning with SGDClassifier
Author (s)	Janhvi Gupta, Prachi Raj, Anshul Arora

To whom it may concern,

Congratulations! Now we are pleased to inform you that **your above full paper has been accepted by ICCEE 2025 for presentation and publication** after the reviewing process, monitored by the Technical Program Committee. You are cordially invited to present your paper on the conference. **Your paper will be published in conference proceedings,** when you follow the procedure to finish the registration before the set date. 2025 The 18th International Conference on Computer and Electrical Engineering (ICCEE 2025) will be held in Singapore on June 20-22, 2025. It provides opportunities for the different areas delegates to exchange new ideas and application experiences face to face, to establish business or research relations and to find global partners for future collaboration.

Registration Procedure

1. Revise your paper according to Comments in review form.
2. Format your paper according to the Template attached carefully https://iccee.org/files/IOS/Word-Template.dotx
3. Fill and complete the Registration Form and pay for it online: https://iconf.young.ac.cn/RAUB6
4. Return your registration materials below to iccee_secretary@academic.net before the deadline.
   A. final papers (both doc and pdf version) B. payment proof

Notes:

1. We will feedback registration status in 3 working days after receiving your registration materials. If you don't receive any response, please contact us proactively.
2. For registration fee and cancellation policy, please visit https://iccee.org/registration.html to check.
3. Please do not trust any contact information other than the official website. Be aware of the various types of online fraud and be alert.

Please feel free to contact us if you have any further questions. Look forward to see you in Singapore!

Yours sincerely,
ICCEE 2025 Organizing Committee

CONTACT US
Secretary: Ms. Mila Xiao
Email: iccee_secretary@academic.net
Tel: +86-134 0855 5552
Wechat: Iconf-cs-2

9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text
- Small Matches (less than 8 words)

Match Groups

73 Not Cited or Quoted 9%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

3% Internet sources
5% Publications
5% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **73** Not Cited or Quoted 9%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

3% Internet sources
5% Publications
5% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Publication	
Garima Kumari, Anshul Arora. "Smartphone Malware Detection using Permission...		3%
2	Internet	
www.mdpi.com		<1%
3	Submitted works	
University of West London on 2025-01-16		<1%
4	Submitted works	
University of West London on 2025-01-16		<1%
5	Submitted works	
Delhi Technological University on 2024-04-25		<1%
6	Internet	
patchskill.com		<1%
7	Internet	
link.springer.com		<1%
8	Publication	
T. Mariprasath, Kumar Reddy Cheepati, Marco Rivera. "Practical Guide to Machin...		<1%
9	Submitted works	
University of Greenwich on 2025-04-23		<1%
10	Internet	
www.joiv.org		<1%