


الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
جامعة قاصدي مرباح ورقلة
كلية تكنولوجيات الإعلام والإتصال
قسم الإعلام الآلي وتكنولوجيات الإعلام

Republic of Algeria Democratic and People's
Ministry of Higher Education and Scientific Research
University of KASDI Merbah–Ouargla
Faculty of New Technologies of Information and Communication
Department of Science Computer and Information Technologies

Professional Master Thesis

Field: Mathematics and Computer Science
Sector: Computer Science
Specialty: Network Administration and Security

Theme

Privacy-Preserving Android Malware Detection Based on Federated Learning

Presented by: SAIB Melissa and ABANOU Baya

Publicly discussed: 12/06/2025

Jury members:

Dr. Kahlessenane Fares	President	UKM Ouargla
Dr. BENKADDOUR Mohammed Kamel	Supervisor	UKM Ouargla
Dr. Cheradid Abdelatif	Examiner	UKM Ouargla

Academic year: 2024/2025

Acknowledgment

First and foremost, we express our deepest gratitude to Almighty Allah, whose boundless mercy and blessings have guided and supported us throughout our academic journey. It is by His will that we were able to overcome the challenges and reach this important milestone.

We would like to extend our sincere thanks and appreciation to our supervisor, ***Dr. BENKADDOUR Mohammed Kamel****, for his continuous guidance, encouragement, and invaluable advice throughout the preparation of this thesis. His constructive feedback and insightful suggestions played a vital role in shaping our research and improving the quality of our work.*

We are also deeply grateful to our co-supervisor, ***Dr. BENDER-RADJI Chourouk****, for her support, expertise, and kind assistance. Her contribution has significantly enriched our learning experience and provided us with essential perspectives that helped us move forward in our study.*

We extend our heartfelt thanks to our beloved parents for their unwavering love, support, and prayers. Their sacrifices, encouragement, and belief in us have always been a source of strength and motivation. We owe this achievement to their endless patience and understanding.

Finally, we would like to thank all the people who supported us directly or indirectly during this journey, including our teachers, colleagues, and friends. Each one of them has contributed to our progress in meaningful ways, and we are sincerely grateful for their help.

May Allah bless all those who have guided and assisted us along the way.

Abstract

In response to the explosive growth of Android smartphones and the corresponding surge in mobile malware, traditional security solutions which rely on aggregating raw user data in a central repository pose unacceptable privacy risks. Although classical Machine Learning and Deep Learning classifiers achieve high detection rates, they typically demand unrestricted access to sensitive behavioral or system logs. To address this dilemma, we propose a privacy-preserving Android malware detection framework based on Federated Learning. In our approach, each user device independently trains a Convolutional Neural Network (CNN) on its local data and transmits only model updates to a coordinating server. The server integrates these updates (using FedAvg or FedProx) into a global model and redistributes it for successive local training rounds. We evaluate performance on the CICAndMal2020 dataset under both IID and non-IID data partitions, experimenting with 3, 5, and 10 client configurations. Our results demonstrate that the federated models not only rival but in some settings surpass the accuracy of a centrally trained counterpart while ensuring that all raw data remain on end-user devices. This work highlights Federated Learning's promise for scalable, privacy-aware Android malware defense.

Keywords : Federated learning, Android Malware, CICANDMAL2020, CNN, FedAvg, FedProx, IID, Non-IID

Résumé

En réponse à la croissance fulgurante des smartphones Android et à la hausse correspondante des malwares mobiles, les solutions de sécurité traditionnelles, qui reposent sur l'agrégation de données sensibles des utilisateurs dans un dépôt central, présentent des risques de confidentialité inacceptables. Bien que les classificateurs classiques d'apprentissage automatique et d'apprentissage profond atteignent des taux de détection élevés, ils exigent généralement un accès illimité aux journaux comportementaux ou systèmes sensibles. Pour résoudre ce dilemme, nous proposons un cadre de détection de malwares Android respectueux de la vie privée, fondé sur l'apprentissage fédéré. Dans notre approche, chaque appareil utilisateur entraîne de manière autonome un réseau de neurones convolutionnel (CNN) sur ses données locales et transmet uniquement les mises à jour du modèle à un serveur coordonnateur. Le serveur intègre ces mises à jour (en utilisant FedAvg ou FedProx) dans un modèle global, qu'il redistribue ensuite pour les tours d'entraînement locaux suivants. Nous évaluons les performances sur l'ensemble de données CICAndMal2020, dans des configurations IID et non-IID, et avec 3, 5 et 10 clients. Nos résultats montrent que les modèles fédérés non seulement égalent, mais dans certains cas surpassent la précision d'un entraînement centralisé, tout en garantissant que toutes les données brutes restent sur les appareils des utilisateurs. Ce travail met en lumière le potentiel de l'apprentissage fédéré pour une défense Android évolutive et respectueuse de la vie privée.

Mots clés : Apprentissage fédéré, Logiciels malveillants Android, CICANDMAL2020, CNN, FedAvg, FedProx, IID, Non-IID

ملخص

استجابةً للنمو السريع في عدد هواتف أندرويد والزيادة المصاحبة في برمجيات الخبيثة، تُعَدُّ حلول الأمان التقليدية التي تعتمد على تجميع بيانات المستخدمين الخام في خادم مركزي محفوفة بمخاطر كبيرة على الخصوصية. وعلى الرغم من أن المصنفات التقليدية في التعلم الآلي والتعلم العميق تحقق معدلات كشف مرتفعة، إلا أنها تتطلب عادةً الوصول غير المحدود إلى سجلات سلوكية أو نظامية حساسة. لمعالجة هذه المعضلة، نقترح إطارًا لاكتشاف برمجيات أندرويد الخبيثة يحافظ على خصوصية المستخدمين ويستند إلى التعلم الاتحادي. في منهجنا، يقوم كل جهاز مستخدم بتدريب شبكة عصبية التفافية (CNN) بشكل مستقل على بياناته المحلية ويرسل إلى الخادم المنسق تحديثات النموذج فقط، دون مشاركة البيانات الخام. يجمع الخادم هذه التحديثات باستخدام خوارزميات FedAvg أو FedProx لتكوين نموذج عام ثم يوزعه مجددًا لجولات تدريب محلية لاحقة. قمنا بتقييم الأداء على مجموعة بيانات CICAndMal2020 في سيناريوهات بيانات متطابقة التوزيع (IID) وغير متطابقة التوزيع ،(non-IID) وباستخدام 3 و5 و10 عملاء. أظهرت النتائج أن النماذج الاتحادية لا تكتفي بمضاهاة دقة التدريب المركزي فحسب، بل تتجاوزه في بعض الحالات، مع ضمان بقاء جميع البيانات الخام على أجهزة المستخدمين. يسلط هذا العمل الضوء على إمكانات التعلم الاتحادي كوسيلة دفاعية قابلة للتوسع وتحترم خصوصية مستخدمي أندرويد.

الكلمات المفتاحية : التعلم الفيدرالي, البرمجيات الخبيثة لأندرويد, FedProx, FedAvg, CNN, CICANDMAL2020, Non-IID IID,

---

Contents

List of Figures

List of Tables

Abbreviations

Malware	Malicious Software
CNN	Convolutional Neural Networks
DL	Deep Learning
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
RELU	Rectified Linear Unit
IA	Intelligence Artificielle
ML	Machine Learning
FL	Federated Learning
FedProx	Federated Proximal
FedAvg	Federated Averaging
IID	Independent and Identically Distributed
Non-IID	Non-Independent and Identically Distributed
FP	False Positive
FN	False Negative
TP	True Positive
TN	True Negative

General Introduction

In recent years, the Android operating system has become the most widely used mobile platform worldwide, powering billions of devices from smartphones and tablets to smart TVs and IoT devices [1]. This widespread use has driven a surge in Android application development and distribution, providing users with unprecedented access to information, services, and entertainment. However, this rapid growth has also introduced serious security risks, making Android devices prime targets for cybercriminals [2]. The open nature of the Android ecosystem, along with the ease of app publication, has made it particularly susceptible to various forms of malicious software, or malware, which can compromise user privacy, steal sensitive information, or harm devices.

As the volume and complexity of Android malware continue to grow, traditional security methods like signature-based detection have become less effective [3]. These methods often fail to catch new threats or variants that use obfuscation to avoid detection. As a result, there has been increasing interest in using Artificial Intelligence (AI) especially Machine Learning (ML) and Deep Learning (DL) to create smart, adaptive malware detection systems. These technologies can automatically learn from data, allowing systems to identify patterns and behaviors linked to malicious activity.

Machine Learning, with its strength in learning from labeled data, has shown great potential in classifying Android apps as either benign or malicious. Commonly used ML algorithms include Decision Trees, Random Forests, and Support Vector Machines (SVM). Deep Learning models such as Convolutional Neural Networks (CNNs) and Long Short Term Memory networks (LSTMs) have taken things further by automatically extracting complex features from raw data [4]. These techniques have achieved high accuracy in tasks such as dynamic behavior analysis, static code inspection, and permission-based detection.

Despite their strong performance, these AI-driven systems often depend on centralized data collection, where user data must be uploaded to a central server for model training. This raises significant concerns about data privacy and security and also introduces inefficiencies in mobile environments that are constrained by bandwidth, computational power, and battery life.

To address these issues, researchers have recently turned to Federated Learning (FL) a decentralized machine learning approach that allows multiple clients (such as mobile devices or institutions) to collaboratively train a global model without sharing raw data [5].Instead, only model updates or gradients are sent to a central aggregator, which helps preserve user privacy while still benefiting from the collective knowledge distributed across clients. This makes FL especially promising for privacy sensitive applications like Android malware detection, where data must remain on the user's device.

The goal of this thesis is to develop a federated learning based Android malware detection and classification system capable of identifying malware effectively.

Research Motivation

Machine Learning and Deep Learning models have achieved impressive results in detecting and classifying Android malware. However, most of the existing research still relies on centralized training, which brings up concerns about data privacy, limited device resources, and real-world deployment challenges.

What motivated this research is the need to explore whether newer, decentralized approaches like Federated Learning can address those limitations without sacrificing performance. Even though FL is gaining attention, there's still not enough work that directly compares its effectiveness with traditional ML and DL methods for Android malware detection.

By comparing centralized and federated approaches in the same setup, this thesis aims to give a clearer picture of their strengths and weaknesses. The goal is to see if FL can be a viable privacy-preserving alternative that still delivers strong results.

Research Objective

The primary objective of this research is to develop and evaluate effective models for Android malware detection and classification using both Machine Learning (ML) and Deep Learning (DL) techniques. The study aims to analyze the performance of these centralized models and then implement a Federated Learning (FL) approach to compare results in terms of accuracy, efficiency, and privacy preservation.

Specifically, this work seeks to:

- Build and train ML and DL models to classify Android applications as benign or malware.
- Implement a Federated Learning-based approach using the a deep learning model to avoid centralized data collection.
- Compare the performance of centralized and federated approaches to understand the trade-offs in accuracy, privacy, and resource efficiency.
- Investigate whether FL can be a practical alternative to traditional methods for Android malware detection, especially in privacy-sensitive environments.

Roadmap of the Thesis

The rest of this thesis is organized into four chapters, each addressing specific aspects of the study

Chapter 1: Malware Classification and Detection This chapter gives a general introduction to malware. It explains what malware is, how it has changed over time, and what it is used for. It also describes the different types of malware and how they work. Then, it focuses on Android malware how it enters Android devices, examples of famous Android malware, and how to protect against them. Finally, the chapter talks about the problems researchers face and possible future improvements in malware detection.

Chapter 2: State of the Art This chapter talks about the role of Artificial Intelligence (AI) in cybersecurity. It explains machine learning and deep learning, with examples of models like Decision Trees,MLP, and LSTMs. It also presents some of the datasets commonly used in malware research and the evaluation metrics used to measure model

performance. In the end, it gives an overview of recent studies and compares different research works in this field.

Chapter 3: Federated Learning for Android Malware Detection This chapter introduces Federated Learning (FL), a method that allows devices to train a model without sharing their data. It explains how FL works, why it is useful for Android malware detection, and what challenges it faces. It also presents different FL methods like FedAvg and FedProx, and how data is shared or split across devices. The chapter explains how FL can improve security and privacy in malware detection.

Chapter 4: Experiment, Results, and Discussion This chapter describes the tools and dataset used in the experiments. It explains how the data was prepared and how it was divided among devices. It presents the results of a traditional (centralized) deep learning model and then compares them with federated learning results. Finally, it analyzes the performance of both models and discusses the findings.

Chapter 1

Overview of Android Malware

1.1 Introduction

Malicious software, commonly referred to as *malware*, continues to pose a serious threat in today's digital landscape. It has the potential to compromise individual users, disrupt business operations, and even affect large-scale network infrastructures. The threat becomes even more significant in the context of mobile devices, particularly those running the Android operating system.

Android devices have become an essential part of daily life due to their affordability, open-source nature, and wide application ecosystem. However, these same characteristics make the Android platform a prime target for malware developers. The vast user base and open architecture allow attackers to easily deploy malicious applications, often disguising them as legitimate software.

This chapter begins by defining malware and exploring its historical background. It also provides a broad overview of the major types of malware and outlines the various approaches used for malware detection and classification. The goal is to build a solid understanding of how malware functions, how it is categorized, and how it can be detected using modern data-driven techniques.

1.2 Malware

1.2.1 Definition of Malware

Malware, short for malicious software, is a type of program made to harm people, organizations, and computer or communication systems. It can do many bad things, like block your internet access, damage your operating system, steal passwords and private information, or lock your important files and ask for money (ransom) to unlock them.

In recent years, malware has become a serious and growing problem. For example, in 2017, the number of new malware programs increased by 22.9% compared to 2016, reaching over 8.4 million new threats.

Today, malware is one of the main tools used in cyberattacks. Hackers use it to take control of computers, crash servers, send large amounts of spam, attack critical systems, and break into data centers. These kinds of attacks can cause huge damage and lead to big financial losses. Figure 1.1,shows Basic illustration of how malware can disguise itself as a normal app to infect an Android device and perform harmful activities like stealing personal data. [6]

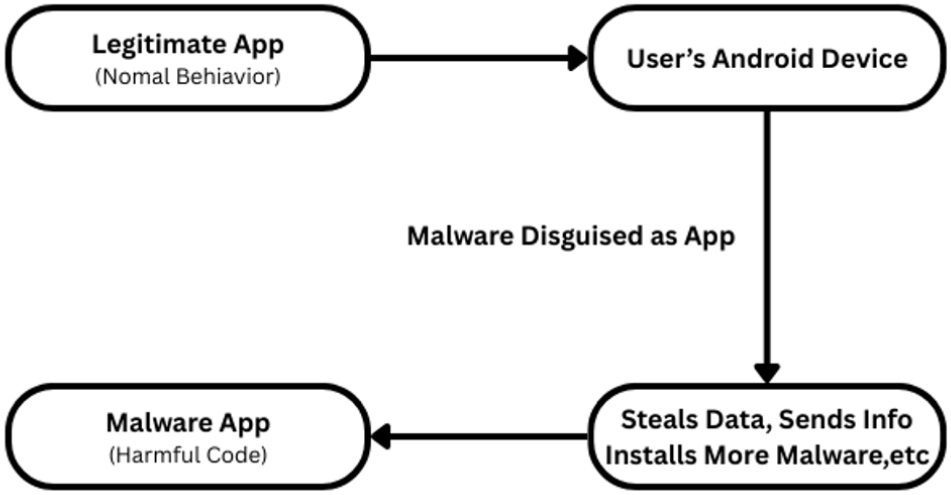

Figure 1.1: How Malware works

1.2.2 History and Evolution of Malware

The evolution of malware can be divided into five main phases. In the early stage, malware first appeared and was mainly simple in nature. The second phase saw the arrival of the first Windows-based malware and email worms, which made spreading viruses easier. With the rise of the internet in the third phase, network worms became common and started spreading rapidly. The fourth phase introduced more dangerous types like rootkits and ransomware, which are harder to detect and more harmful. In the current and fifth phase, malware has become highly advanced, with some created by government agencies for spying and cyber-espionage purposes.Figure 1.2, illustrates the timeline of the 5 phases.

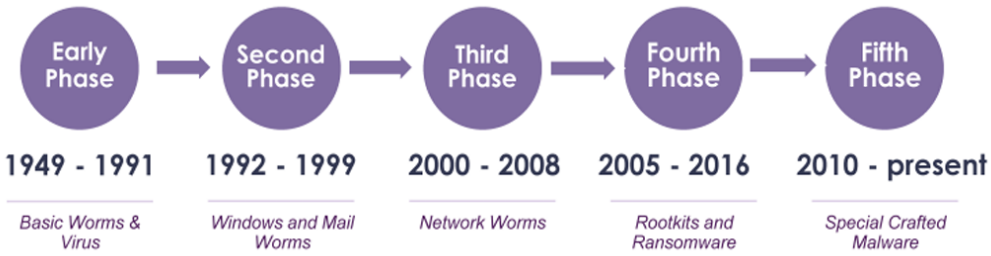

Figure 1.2: Malware evolution: timeline of the 5 stages

- **Early Years (1949s–1991s):** In the beginning, malware was mostly experimental. One of the first examples was the Creeper virus in 1971, which could copy itself. Then in 1986, the Brain virus appeared the first virus to infect personal computers, showing a shift from experiments to real harmful actions.
- **Transition Period (1992s–1999s):** During this time, viruses started spreading more easily. Some infected files and others targeted programs like Microsoft Word. As home computers and internet use grew, so did the number of infections.
- **Internet Age (2000–2008):** When the internet became common, malware spread even faster. Worms and trojans became popular tools for hackers. A well-known example

is the ILOVEYOU worm in 2000, which spread through email and caused a lot of damage.
• **Rise of Ransomware (2015-2017):** In the 2010s, ransomware became one of the biggest threats. A famous example is WannaCry in 2017, which locked people's files and asked for money to unlock them. These attacks showed how weak many systems still are.
• **Modern Malware (Late 2010s to Now):** Today, malware is much more advanced. Some types can change their code to avoid detection, while others don't leave files on the system (fileless malware). Hackers are even using AI to make malware smarter, which makes protecting systems even more difficult. [7]

1.2.3 Common Objectives of Malware

Malware is usually created with a specific goal in mind. While the intentions can vary widely, there are a few common purposes that are seen most often:

1. **Stealing Information:** One of the main reasons hackers use malware is to steal data. This can include personal information, passwords, credit card numbers, and even government or company secrets. This type of malware can cause serious financial and personal damage to the victim.

2. **Disrupting Systems:** Another goal of malware is to cause problems or damage to a system. For example, a virus might delete important system files, making a computer stop working. In more serious cases, malware can be used to take down entire networks or launch powerful attacks like DDoS (Distributed Denial of Service), which overwhelm systems and make services go offline.

3. **Demanding Money:** Some malware is made to scare or force people into paying money. For instance:

   Scareware tries to trick people by showing fake warnings (like fake virus alerts) to get them to buy unnecessary software.

   Ransomware locks or encrypts a person's files, and demands money (usually in cryptocurrency like Bitcoin) to unlock them. Some companies have even started keeping cryptocurrency ready in case they get attacked and choose to pay the ransom.[8]

1.2.4 Life Cycle of Web-based Malware

When a computer gets infected by web-based malware (malware that spreads through the internet), the malware usually doesn't stop there. It often talks to other computers or servers online to do more harm. Here are the main stages it goes through:

1. **Spreading to Other Computers:** After getting into a computer, the malware tries to find other weak computers to infect. It usually looks for nearby computers in the same network or others on the internet. It does this by scanning for open ports (doors) on the computer that hackers usually use to break in.

2. **Telling the Hacker:** Once it's installed, the malware sends a message back to the person who made it. This message might go through email, a website, or other communication methods. It tells the hacker that the malware was installed successfully and may include details like the victim's IP address, computer name, or system type.

3. **Stealing Data:** After letting the hacker know it's there, the malware often starts stealing private information. This can include saved passwords, browsing history, or even banking details. It sends this stolen information back to the hacker using email, websites, or file upload methods like FTP.

4. **Becoming Part of a Botnet:** Sometimes, the malware makes the infected computer part of a larger group of infected computers called a botnet. These computers are controlled by the hacker and can be used to send spam emails, attack websites, or spread more malware all without the owner knowing. [9]

1.3 Types of Malware

1.3.1 Viruses

A virus is a type of malware that attaches itself to another program. When the user runs that program usually without knowing it's infected the virus spreads by inserting its code into other programs on the computer. [10]

1.3.2 Worms

Worms are a kind of malware that are a lot like viruses because they can make copies of themselves. But the main difference is that worms don't need any help from the user to spread they can move from one computer to another all on their own. Viruses, on the other hand, need the user to do something, like open a file, to start spreading.[10]

1.3.3 Trojans

A Trojan Horse, or simply a Trojan, is a type of malware that pretends to be a safe or useful program. But once it gets into your system, it gives attackers the ability to do anything a normal user can like stealing files, changing data, or even deleting things. Trojans often hide inside downloads like games, apps, tools, or software updates. Many times, they trick people into installing them by using fake messages or links (like in phishing emails) that look real.

- **Trojan: Virus or Malware?** A Trojan is often called a "Trojan virus," but that's not really accurate. Unlike a virus or a worm, a Trojan can't copy itself or run on its own it needs the user to open or install it. Trojans are still a type of malware, and like other malware, they're made to cause harm. They can damage or steal files, spy on what you do, or create hidden ways for hackers to get into your system. Sometimes, they block, change, delete, or leak your data then hackers might demand money to return it, or sell it on the dark web.

- **Types of Trojan Malware:** Trojans are a very common and versatile attack vehicle for cybercriminals. Here we explore 10 examples of Trojans and how they work:

  1.Exploit Trojan: This type looks for weak spots in software and uses them to break into your system.

2.Downloader Trojan: Once it gets into your device, it quietly downloads and installs more harmful programs.

3.Ransom Trojan: Like ransomware, it locks your device or files and demands money to unlock them.

4.Backdoor Trojan: It creates a secret way for hackers to get back into your system anytime they want.

5.DDoS Attack Trojan: This one infects many devices and turns them into a botnet. The hacker then uses this network to flood a system with traffic, crashing websites or services.

6.Fake Antivirus Trojan: It pretends to be antivirus software, shows fake virus warnings, and tricks you into paying to "fix" problems that don't actually exist.

7.Rootkit Trojan: This Trojan hides deep in your system to stay unnoticed for a long time while doing damage or spying.

8.SMS Trojan: Targets mobile phones. It can send or intercept text messages, sometimes to expensive premium numbers to make money.

9.Banking Trojan: Made to steal bank info like login details, credit card numbers, or payment app data.

10.GameThief Trojan: Targets gamers, trying to steal login details for online gaming accounts.

Each of these Trojans has a different goal, but they all work by tricking users and secretly carrying out harmful actions.[11]

1.3.4 Ransomware

Ransomware is a type of malicious software that locks or encrypts your files so you can't access them. The attacker then demands money (a "ransom") to give you back access, usually by providing a special decryption key.

If you don't pay, they might either permanently block your access or threaten to leak your personal or sensitive data online.

Ransomware has become one of the most dangerous and widespread cyber threats today. It targets all kinds of organizations like governments, schools, banks, and hospitals and causes millions of dollars in losses every year.

- **How do ransomware attacks work?** No matter what kind of ransomware it is, most attacks usually follow the same basic steps:

  Step 1: Infection

  The attack usually starts with a phishing email. It might look like a normal message, but it contains a malicious link or attachment. When someone clicks it, the ransomware is secretly downloaded onto their device.

  Step 2: Encryption

  Once the ransomware is on the system, it begins scanning for important files like documents, photos, or databases and locks them by encrypting them. Some types of ransomware can even spread to other computers on the same network, causing more damage across the organization.

Step 3: Ransom demand

Once the data has been encrypted, a decryption key is required to unlock the files. In order to get the decryption key, the victim must follow the instructions left on a ransom note that outline how to pay the attacker usually in Bitcoin.[11]

1.3.5 Spyware

Spyware is a type of harmful software that secretly gets into your computer or phone and collects private information like passwords, PINs, or payment details. This stolen data is then sent to advertisers, data companies, or even cybercriminals who make money from it.

Spyware used to be more common on Windows computers, but now it can also infect Apple devices and smartphones. Since people now use their phones for banking and other personal things, spyware attacks on mobile devices have become more advanced and frequent.

It's important to know that not all tracking software is bad. For example, some websites use cookies to remember your login info or customize your experience. That kind of tracking is usually harmless.

Spyware doesn't usually go after specific people. Instead, it tries to infect as many devices as possible to gather as much information as it can. Because it runs quietly in the background, most people don't even know it's there and getting rid of it can be tricky without strong security tools.[11]

1.3.6 Adware

Adware is a type of software that shows you annoying ads like pop-ups, banners, and random redirects while you're using your device. It watches what you do online and then shows ads based on your activity usually without asking you first. Most of the time, adware affects computers, but it can also get into phones and tablets.

- **How Does Adware Get On Your Device?**

  Adware usually sneaks in when you download free apps or programs from the internet. It hides in the background and installs itself without you noticing.

  Some developers include adware in their software on purpose because they get paid every time you see or click an ad. That means even trustworthy-looking apps or programs might come with hidden adware.

  Even though it might seem like just a small annoyance, adware can slow down your device, mess with your browser, and open the door to more serious threats.[11]

1.3.7 Rootkits

A rootkit is a sneaky type of malware that gives hackers secret control over a computer or even an entire network. Once it's inside, it opens a hidden backdoor so attackers can come and go whenever they want without being noticed.

Rootkits can be very dangerous because they don't just sit there. They often bring in more harmful malware like ransomware, trojans, keyloggers, or bots.

What makes rootkits especially tricky is that they're very good at hiding. They can even turn off your antivirus or avoid being detected by security scans. That's why they can stay on a system for years without anyone knowing.[11]

1.3.8 Fileless Malware

Fileless malware is a type of cyberattack that doesn't rely on downloading files or saving anything on the computer's hard drive. Instead, the attacker takes advantage of a weak spot in a trusted program and injects harmful code straight into the computer's memory (RAM).

What makes it even more dangerous is that it often uses well-known programs like Microsoft Office, PowerShell, or Windows Management Instrumentation (WMI). These are tools that computers already trust, so the attack can run quietly in the background without being noticed making it very hard to detect or stop. [12]

1.4 Detection and Analysis of Malware

Malware analysis is an important first step in detecting malware. Before we can detect it, we need to understand how it works and why it was created. This kind of understanding helps security experts build better tools to defend against it. There are three main types of malware analysis methods, and they're grouped based on how they work and how much time they take.

1.4.1 Static Analysis

Static Analysis is a way of studying software or code without actually running it. In this method, experts look at the code itself to find out if it contains anything harmful. To do this, they often reverse engineer the code using special tools. This helps them understand how the malware is built and how it might behave.

Some of the tools used in static analysis include: - Debuggers - Disassemblers - Decompilers - Source code analyzers

Common methods used in static analysis are: - Checking the file format - Looking for hidden strings - Using fingerprinting - Scanning with antivirus tools - Disassembling the code to study its instructions

This approach helps experts figure out whether a program is safe or dangerous before it's ever run.

1.4.2 Dynamic Analysis

Dynamic Analysis is the process of studying how software behaves when it is actually running. Instead of just looking at the code, as in static analysis, this method involves executing the program to observe its actions in real time.

To do this, the suspicious software is run in a virtual environment (like a sandbox, simulator, or emulator), where it's safe to monitor its behavior without affecting the real system. Tools like RegShot and Process Explorer help track what the program does during execution.

During dynamic analysis, security researchers look at: - Which functions the program calls - How the control flow works - The instructions and parameters it uses

This method is considered more effective than static analysis because it can uncover hidden or advanced malware behaviors that only appear when the software is running. However, it takes more time and effort because a special environment needs to be created to safely execute and analyze the malware.

1.4.3 Hybrid

Hybrid Analysis is a method that combines both static and dynamic analysis techniques to take advantage of the strengths of each.

First, the software is checked using static analysis by looking at its code and scanning for known malware signatures. Then, it's executed in a virtual environment to observe how it actually behaves in real-time, just like in dynamic analysis.

By using both methods together, hybrid analysis provides a more complete and accurate understanding of how malware works.[13] The comparison between static and dynamic analysis techniques have been shown in table 1.1.

Table 1.1: Comparison between Static and Dynamic Analysis

Static Analysis	**Dynamic Analysis**
Fast and safe	Time consuming and vulnerable
Good in analyzing multipath malware	Difficult to analyze the multipath malware
Can not analyze obfuscated and polymorphic malware	Can not analyze obfuscated and polymorphic malware
Low level of false positive (Accuracy is high)	High level of false positive (Accuracy is low)

1.5 Android Malware

1.5.1 Android OS Architecture and Security Model

From architecture point of view, Android operating system is divided into four layers: the kernel layer, libraries and runtime layer, applications framework layer, and applications layer. Android kernel is a modified version of Linux2.6 kernel, which is updated from time to time with different versions of Android. The libraries pro vide support for graphics, media capabilities, and data storage. Android runtime, embedded in libraries layer, contains the Dalvik virtual machine to power the applications .As a replacement of Dalvik, Android introduced its new Android RunTime (ART) with ahead-of-time (AOT) compilation, which improves performance. All applications use the applications framework API for accessing the lowest level of the architecture.Figure 1.3, represents the Architecture of Android.

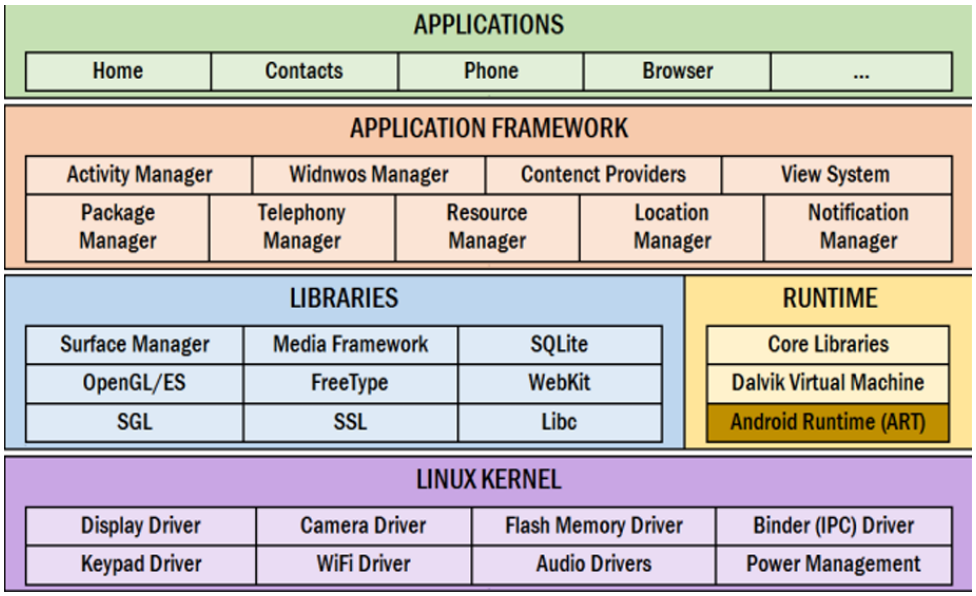

Figure 1.3: Architecture of Android Operating System

- **The Kernel Layer :** The Android Operating System's kernel is a modified version of Linux Kernel to carry out some special requirements of the platform. Linux was selected because it is open source, and has verified pathway evidence. In some cases, drivers need to be rewritten. Linux The kernel provides networking, driver, and virtualization mechanisms. al management of the power and memory. It holds all the items and works well with hardware. necessary hardware drivers. Drivers control and participate in touch with hardware. For instance, each device includes Bluetooth hardware , so the kernel must also provide a Bluetooth driver for Bluetooth hardware to communicate with. Linux kernel is used for process management, networking, memory management, security settings, etc. Android can be ported to a number of hardware, which is a comparatively easy and effortless task.

- **Native Libraries Layer :** These are written in languages like C and C++ to work closely with the device's hardware. The Android Runtime is where apps written in Java get converted and run using something called Dalvik (a special virtual machine).

- **Application Framework:** This is where Android provides services and tools that developers use to make apps like handling user activities, locations, packages, etc.

- **Application Layer (top layer):** This is where all the apps live the ones made by Google, by developers, and maybe even by you. Each app is made of parts like Activities (screens), Services (background tasks), Broadcast Receivers (to react to events), and Content Providers (to share data).

- **Android's Security Model:** Android's security is mainly enforced in two places: the Linux kernel and the application layer.

  At the kernel level, each app runs under its own user identity. This creates isolation between apps meaning one app can't access another app's data or system files unless it has permission.

At the application level, Android uses permissions to control what apps can do. Before an app can send SMS, access the internet, or read contacts, for example, it must ask for permission in its Manifest file. Users are then asked to approve or deny these permissions during installation or while using the app.

These permissions help protect the system and the user's personal data. However, there are risks: users might unknowingly allow unnecessary or harmful permissions to malicious apps, which can lead to privacy breaches.[14]

1.5.2 Common Vectors of Android Malware

Android devices are attractive targets for attackers because they're almost always online and contain valuable personal information like contacts, emails, and login credentials. Some attackers use these devices to build botnets, while others try to steal data or intercept SMS codes used for online banking.

1. **social engineering tricking:** is One common way attackers get in users into installing harmful apps. For example, they might create a fake website that looks like a real one and convince someone to download an app that seems helpful but is actually malicious.

   Unlike desktop systems, Android gives users a bit more control. Before installing an app, the system shows what permissions the app is asking for like access to contacts or the internet. This can help users spot something suspicious, like a simple drawing app asking for access to sensitive data. While this doesn't stop all attacks, it gives users a better chance to catch them before they happen.

2. **Drive-by exploitation:** is a type of attack where a hacker takes advantage of bugs in apps especially web browsers to run harmful code on a device, usually without the user even knowing. These attacks often happen when a user simply visits a malicious website that takes advantage of flaws in the browser to install malware.

   On traditional desktop systems, this type of attack has been a big issue, especially because most software is written in low-level languages like C or C++ that are prone to memory-related bugs. That's why desktop browsers now use stronger protections like sandboxing and memory safeguards.

   On Android, things are a bit different. Most Android apps are written in Java and run inside the Dalvik Virtual Machine (DVM), which adds an extra layer of protection by doing things like boundary checks to prevent these kinds of bugs.

   However, Android isn't completely safe. Some parts of apps, including the default Android browser, still use native code, which is more vulnerable. So even though the risk is reduced compared to desktops, it's not gone entirely. Also, many Android devices are still running outdated versions of the OS, which leaves them exposed to known vulnerabilities that newer updates would fix.

3. **A Phishing attack:** is used in order to gather information, especially credentials from a user by masquerading itself as an official instance. This can be done by sending faked emails or serving similar looking website e.g. for an online banking system. This kind of attack includes aspects of social engineering, as discussed in section 4.1. Due to this, there are no good technical solution for this kind of attacks and so they are still common on desktop system. On the Android platform this also

holds. But the situation here is different, because for most online services, which are used on mobile devices, there are also applications available to have a better user experience while using the service. Therefore, most users do not use the website directly, but use an application for this service. So in this case the attack vector can not be applied that easy because the application can not be tricked as easy as the user.

4. **Network services:** are provided by programs running on the local system. They are used to communicate with other systems. In order to achieve this they listen on a network interface, receiving packets and process them. An example for typical services running on most computer are Samba and NFS, two network file system services. In this case all involved system have to run such a program. Network services are always in the risk of being attacked due to their nature that they are reachable from the network. In most cases it is not necessary to interact with the user. If these services are vulnerable they can be attacked remotely. On desktop systems it is very usual to run such services especially network file systems and printing services. For smartphones this is not the case. At the default configuration there are no network services running or installed. So we have a reduced attack vector space on these devices.[15]

1.5.3 Types of Android Malware

The most common forms of Android malware

- **Worms:** These are a kind of malware that spread on their own from one device to another without needing any help from the user.
- **Trojans:** These hide behind something that looks safe or useful. But once the user opens or interacts with them, their real and harmful purpose is revealed.
- **Spyware:** This quietly collects personal information from a device and sends it elsewhere without the user ever knowing.
- **Expanders:** These are sneaky programs made to rack up extra charges on a user's phone bill, usually through things like premium messages or hidden subscriptions.
- **Backdoors:** This type of malware gives hackers secret access to someone's phone. Once inside, they can move around unnoticed and even install more harmful apps.[16]

1.6 Android Malware Detection Techniques

This section outlines the process of Android malware classification based on the features obtained from valid feature subsets selection. The Android malware detection methods can be categorized into signature, behavior, and machine learning based, among which the most mature method is signature-based detection. The following are introductions of several detection methods.

1.6.1 Signature-Based Detection

Signature-based detection works by looking for known patterns of malware. It uses a library that stores special "signatures" for each known Android malware. These signatures can be things like file names, bits of code, or data, which are found by experts or created automatically.

This method is popular because it is fast and very accurate when it comes to finding malware that is already known. But the downside is that the library needs a lot of work to keep it updated, and it can't find new types of malware that don't have a signature yet.

1.6.2 Heuristic-based detection

Heuristic detection (also called anomaly-based or behavior-based detection) is used to find new or unknown malware. Instead of looking for exact matches like signature-based detection, this method compares the behavior and features of unknown apps with known types of malware. Each malware type is described using a set of rules based on how it usually acts or looks.

These rules can include things like the app's structure, the functions it uses (API calls), the order of operations (code), and other behavior patterns. If a new app matches the rules of a known malware family, it's considered suspicious or harmful.

This method is good at discovering new malware and works by combining different ways to tell good apps from bad ones. It helps where traditional methods fail. However, the downside is that it can sometimes make mistakes, especially with brand-new (zero-day) malware, and label safe apps as dangerous.

1.6.3 Machine learning-based detection

Machine learning trains a learner by adjusting the parameters to make the best predictions. Existing research demonstrated that machine learning is an effective and promising method to detect Android malware. In recent years, many malware detection works have attempted to harness machine learning to seek a breakthrough in unknown Android malware detection. [17]

1.7 Case Studies of Android Malware

1.7.1 DroidDream malware

DroidDream is a type of Android malware that looked like a normal app but did harmful things in secret. It was found in the official Android Market (now Google Play), which made it more dangerous.

Once the user installed the infected app, DroidDream would start working at night between 11 PM and 8 AM when the user was likely sleeping. It used two known bugs in Android to break the phone's security and take full control (called "root access").

After that, it could:

- Steal personal info from the phone, like the phone's ID, model, and more.
- Install other harmful apps without the user knowing.
- Send all the data to hackers using the internet.

These bugs were already fixed in a newer Android version (Gingerbread), but most phones didn't have the update yet. That made it easy for DroidDream to infect many devices.

In short, DroidDream was dangerous because it looked safe, attacked silently at night, and used old security bugs that many phones were still not protected against.[18]

1.7.2 Joker malware

The Joker malware was discovered hiding in 24 apps on Google Play, which together have been downloaded over 472,000 times. Once installed, it operates entirely in the background, silently reading your SMS messages and harvesting your contact list and device information. Without your knowledge or consent, it then fraudulently subscribes you to premium services often costing around €6 per week by mimicking clicks on web-based subscription buttons, intercepting the confirmation codes sent by SMS, and sending those codes back to complete the signup.

Joker's operations are geographically limited: it targets users in 37 countries (including the UK, Germany, France, India, and China) but does not function in the United States or Canada. To avoid detection, Joker never displays any visible interface and, after installation, downloads additional malicious payloads from the internet. These stealth techniques make it particularly difficult for standard mobile security apps to detect or block its activity right away. .[19]

1.8 Prevention and Protection Techniques

1.8.1 Best Practices for Users

To protect your application and its users, adopt the following security best practices ,always enforce encrypted connections (HTTPS) to safeguard data in transit, and consider certificate pinning so your app only accepts known, trusted certificates. Rigorously validate and sanitize all user input to prevent injection of malicious code, and embed security checks into your development workflow review your code frequently, employ safe coding patterns, and run automated vulnerability scans. Keep your knowledge current by tracking emerging threats and recommended countermeasures, and ensure that every tool, SDK, and library you depend on is updated to the latest version, since patches routinely address newly discovered security flaws.[20]

1.8.2 Google Play Protect

Google Play Protect works behind the scenes to keep your device safe in three key ways. First, it continuously scans and verifies every app you install both at installation time and afterward and if it detects a potentially harmful app, it will alert you to remove it, disable it until you do, or even uninstall it automatically.

Second, Play Protect helps guard against malware by monitoring signals such as your internet connection, visits to dangerous websites, and details about your system and installed apps; when Google identifies a threat, it can block installation, stop the app from running, or display a warning. Although you can disable some data-collection features in your settings, Play Protect will continue to vet apps downloaded from the Play Store.

Finally, if an app is ever removed from the Play Store for misusing personal data, Play Protect will notify you and give you the option to uninstall it, ensuring you stay informed and in control of your privacy. [21]

1.8.3 Role of Antivirus and Mobile Security Apps

Antivirus and mobile security apps play a critical role in ensuring the security of Android devices. They protect sensitive user information, help businesses maintain operations by preventing cyberattacks, ensure compliance with data protection regulations, and provide automatic updates to defend against new threats. [22], Figure 1.4 shows Key Roles of Antivirus and Mobile Security Apps

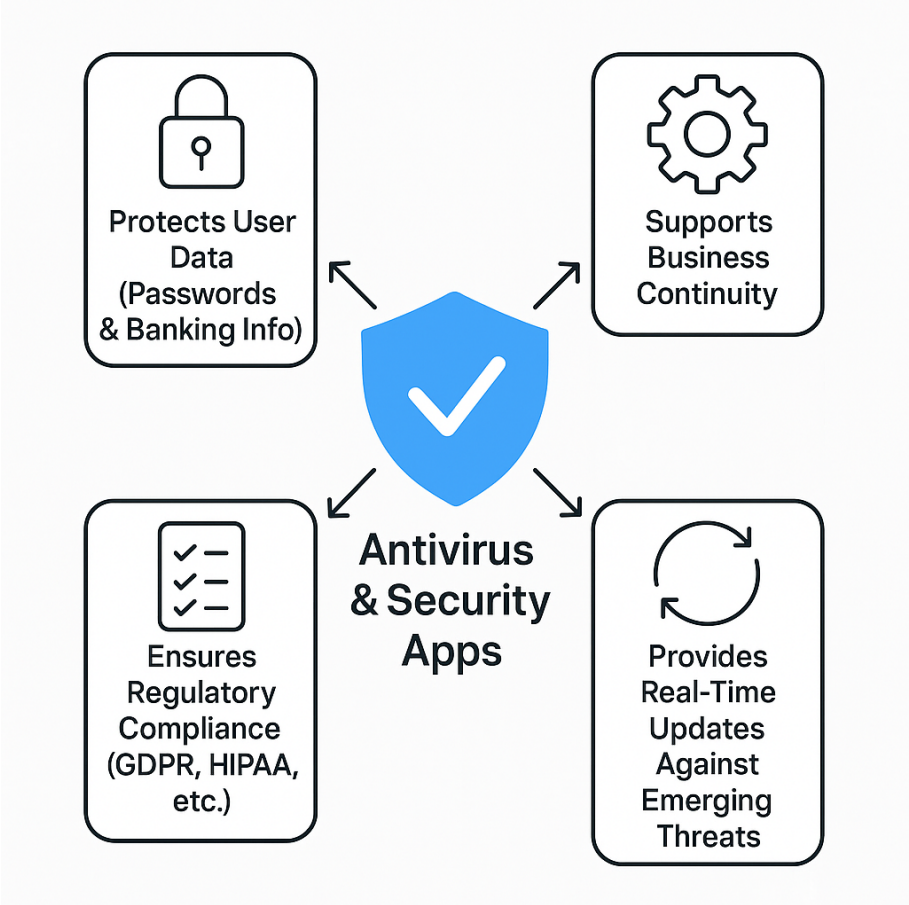

Figure 1.4: Key Roles of Antivirus and Mobile Security Apps

1.9 Challenges and Future Directions

1.9.1 Evasion Techniques Used by Malware

Modern malware employs a range of evasion techniques[23] to bypass detection systems. These methods can be grouped into five main categories, as illustrated below in figure **??**

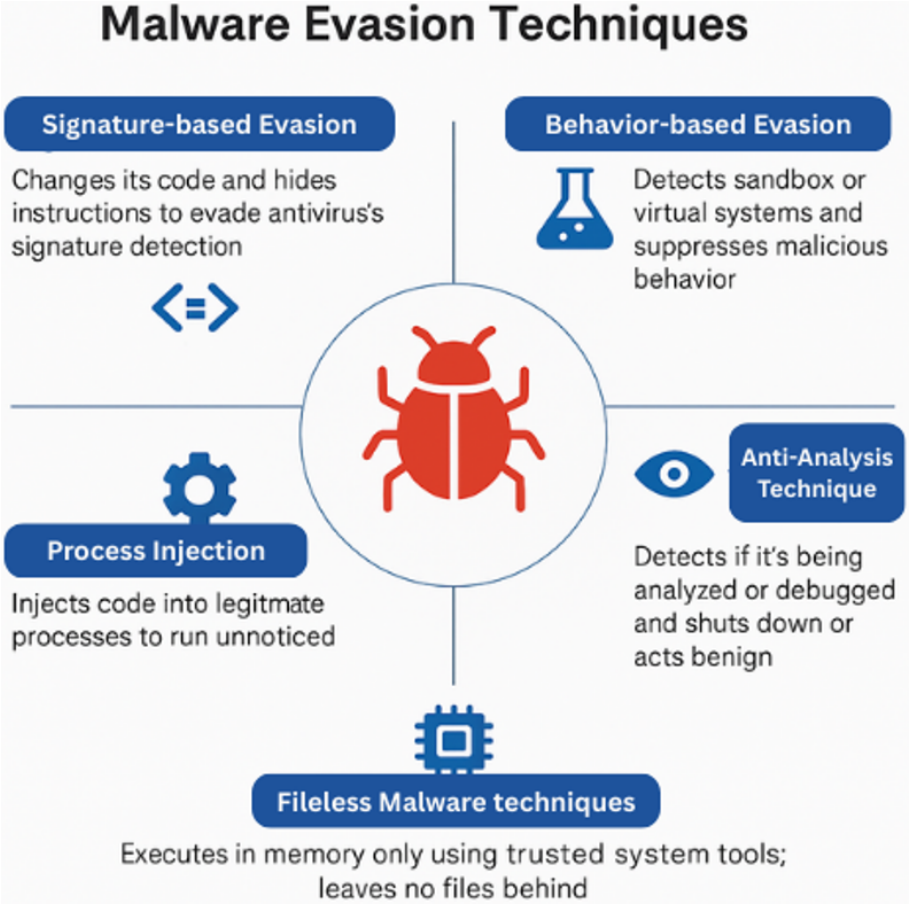

Figure 1.5: Malware Evasion Techniques

1.9.2 Limitations of Current Detection Systems

Based on the survey of the recently proposed malware detection and classification models along with reviewing the approaches and techniques that have been used, the shortcomings and usefulness of those approaches and techniques are specified. Therefore, our review could identify the challenges and open issues. The following are the most open issues and suggestions for research directions.

1. **Obfuscation Techniques** Malware creators use smart tricks to hide their code. They make it look different even though it works the same. This makes it hard for security tools to catch them.

2. **Evasion Techniques** Some malware knows when it's being watched in a testing environment and pretends to be harmless. But once it runs on a real phone or device, it starts doing bad things.

3. **Zero-Day Malware** New types of malware appear every day. Because they've never been seen before, most systems trained on older malware can't detect them.

4. **Redundancy and Irrelevant Behaviours** When the data used to train detection systems has too much extra or unrelated stuff, the system gets confused and makes mistakes.

5. **False Positive/Negative Rate** Sometimes, good apps are wrongly flagged as malware (false positives), or bad apps are missed completely (false negatives). False positives can even break a user's system if the wrong app gets blocked.

6. **Incremental Learning** Many systems are trained only on new malware. That means they might miss old threats because they haven't learned how older malware behaves.[24]

1.9.3 Future of AI in Malware Analysis

1. **Deep Learning for Advanced Threat Detection**

   Future malware detection systems will leverage deep learning algorithms, which can process vast amounts of data and identify malware with greater accuracy. Neural networks will recognize even the most complex attack patterns.

2. **AI-Powered Threat Intelligence**

   AI will collect, analyze, and interpret threat intelligence from various sources, including the dark web, hacker forums, and malware databases, providing organizations with real-time insights into emerging threats.

3. **Real-Time Threat Hunting with AI**

   Instead of waiting for an attack to occur, AI-powered threat hunting systems will actively search for hidden malware and vulnerabilities within networks, preventing attacks before they happen.

4. **AI-Driven Zero-Day Threat Detection**

   AI will play a crucial role in identifying zero-day vulnerabilities by analyzing anomalous behaviors and suspicious code execution in real-time, reducing the risk of unknown threats.

5. **Integration of AI and Blockchain for Cybersecurity**

   Blockchain's immutable records combined with AI-driven analysis will create more secure and transparent malware detection systems, making cyberattacks easier to track and mitigate.

6. **AI-Powered Self-Healing Systems**

   Future cyber security frameworks will include self-healing AI-powered systems, which can detect, respond, and repair vulnerabilities autonomously without human intervention.[25]

1.10 Conclusion

This chapter explained how Android malware works, how it's detected, and the tools used to fight it. We looked at real examples like Joker and DroidDream, and explored different detection methods, including AI.

Even though current systems are helpful, they still face challenges like hidden malware, new types of attacks, and false alarms. Using AI and machine learning can improve detection, but they need constant updates and good data.

In the end, staying safe from malware means using security tools, keeping devices updated, and always being careful with the apps we install.

Chapter 2

State of the Art

2.1 Introduction

Artificial Intelligence (AI) has advanced rapidly, bringing together methods like Machine Learning (ML) and Deep Learning (DL) to build systems that learn from data, spot complex patterns, and make decisions on their own. ML and DL use algorithms and statistical models to tackle tasks such as feature extraction, behavior analysis, and automated classification without needing explicit rules written for each scenario.

In the realm of Android security, these AI techniques have been applied to the challenge of malware classification and detection. Traditional signature-based scanners often miss new or obfuscated threats and can generate many false positives. By contrast, ML models (for example, decision trees, SVMs, and random forests) and DL architectures (like CNNs and LSTMs) learn to distinguish benign apps from malicious ones by training on large, labeled datasets. They can automatically extract features from app code, network activity, or permission usage, and adapt as malware evolves.

However, training powerful AI models typically requires aggregating sensitive app and user data in one place, raising privacy and compliance concerns. Federated Learning (FL) has emerged as a solution: devices train local models on their own data and send only model updates to a central server. This approach keeps raw data on-device, preserving privacy while still benefiting from the collective knowledge of multiple clients.

This chapter provides a comprehensive introduction to AI and its various branches. It briefly discusses the datasets used in android malware detection and classification.

2.2 Overview of Artificial Intelligence (AI)

Because AI comes in so many different forms, it's hard to put it into words. One area of agreement is that artificial intelligence is a field of scientific inquiry, rather than an end product. AI is difficult to define with any precision partially because several different groups of researchers with drastically different motivations are working in the field. M.L.'s definition may be the most accurate. Minsky, "Artificial intelligence is the science of making machines do things that would require intelligence if done by men."[26] At its core, Artificial Intelligence (AI) is about making machines that can think and act like humans. This means they can see, understand, learn, and make decisions on their own. AI has changed how we use technology and how we think about what machines can do. Since it first started in the mid-1900s, AI has grown a lot, thanks to better computers, smarter algorithms, and more access to data. One important part of AI is Machine Learning (ML), which helps machines learn from data, find patterns, and make predictions without being directly programmed. Deep Learning (DL) is a newer and more advanced area of ML. It allows machines to understand very complex data using systems called neural networks, which are inspired by how the human brain works. Deep Learning has made big progress in areas like recognizing images, understanding language, and speech recognition. Figure 2.1 illustrates that artificial intelligence is a broad discipline encompassing both machine learning and deep learning.

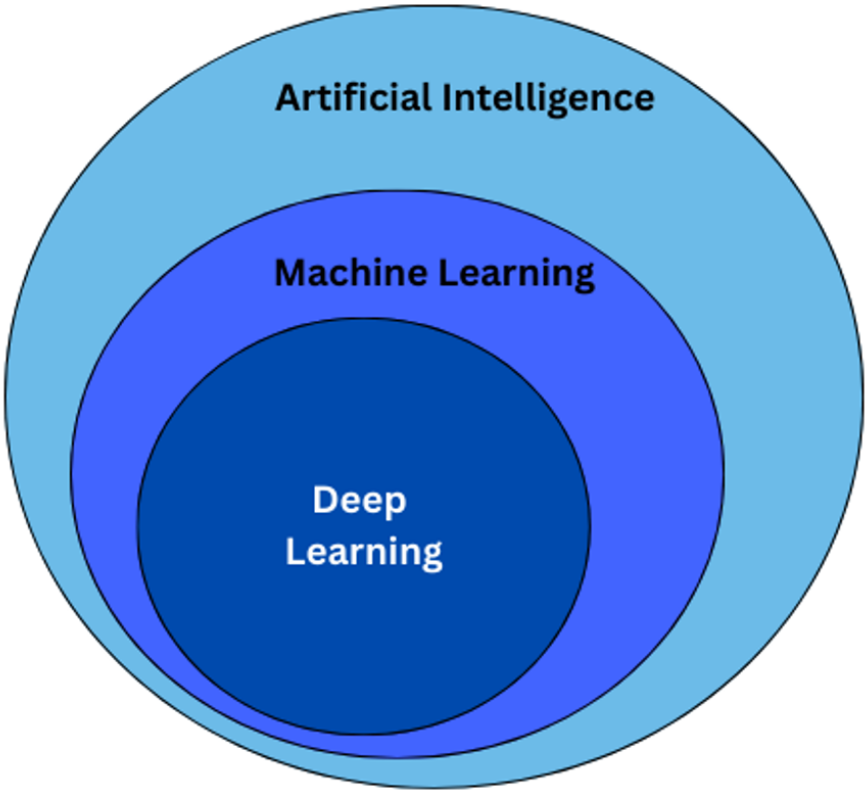

Figure 2.1: Relationship between Artificial Intelligence, Machine Learning and Deep Learning

2.2.1 Machine learning

Machine Learning is a field in computer science that allows computers to learn from data and improve their performance without being directly programmed. One of the earliest definitions was given by Arthur Samuel, who created a program that could play checkers. At first, he was better than the program, but over time, the program learned from playing games against itself and became better by recognizing good and bad moves.

A more formal definition was provided by Tom Mitchell, who said that a computer program learns from experience (E), in relation to a specific task (T) and a performance measure (P), if its performance at the task improves with more experience.

Today, Machine Learning is used in many fields to make sense of large amounts of data, using algorithms that help machines learn and make smart decisions.[27]

2.2.1.1 Machine Learning algorithms

1. **Supervised Learning:** Supervised learning is algorithms where the computer learns from examples that already have the correct answers. It looks at a set of input and output pairs and tries to figure out the relationship between them. These algorithms need some guidance because they learn from labeled data — where each input is matched with the right output.

   The data is usually split into two parts: a training set and a test set. The algorithm learns patterns from the training data, then uses what it learned to make predictions or classifications on the test data. The workflow of supervised machine learning algorithms is given in Figure 2.2.

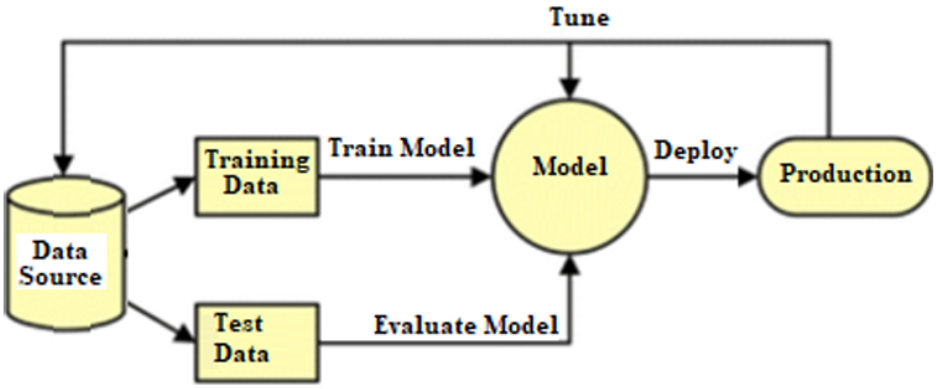

Figure 2.2: Supervised learning Workflow

2. **Unsupervised Learning :** Unsupervised learning is a type of machine learning where the system learns on its own by finding patterns in the input data—without being given any labels or correct answers. It mainly works by grouping similar data into clusters, which is why it's often called a clustering algorithm. A common example is Google News (URL news.google.com), which automatically groups related news articles from different sources into the same news story, helping users see all coverage of a topic in one place.

3. **Semi-supervised learning:** is a type of machine learning that uses both labeled and unlabeled data to learn. It sits between supervised and unsupervised learning. The main idea is to take advantage of the large amount of data that doesn't have labels, along with a smaller amount of labeled data, to build better models. This approach is especially useful when labeling data is difficult or expensive, but there's plenty of unlabeled data available.[28]

4. **Reinforcement Learning :** Reinforcement learning is a type of learning where an agent learns by interacting with its environment. The goal is to take actions that will earn the highest long-term reward. When the agent makes a good decision, it gets a reward; when it makes a bad one, it gets a penalty. Unlike supervised learning, reinforcement learning doesn't give the correct answers upfront. Instead, the agent learns from experience by trying different actions and learning which ones work best over time.

2.2.2 Key Differences Between Traditional Machine Learning and Deep Learning

- Traditional machine learning needs experts to handcraft the features from data, while deep learning learns those features automatically from the raw data.

- Classic machine learning models struggle to work with raw data like images or audio. Deep learning, on the other hand, can process this kind of data directly through multiple layers.

- Machine learning methods usually rely on shallow models with limited abstraction. Deep learning builds several layers of representation, where each layer captures increasingly complex patterns.

- As the size of data and computing power increases, deep learning keeps improving. Traditional machine learning models often hit a limit and stop getting better with more data.
- Training deep models used to be very hard, but advances like the use of GPUs and access to large datasets have made it much easier and more practical.
- Deep learning has outperformed traditional methods in many real-world tasks, including speech recognition, image classification, and natural language understanding.[29]

2.2.3 Techniques of ML

Machine learning encompasses a wide range of algorithms designed to analyze data, identify patterns, and make predictions. Among these, certain techniques have proven to be particularly effective in classification problems. In this section, we focus on two widely used methods Decision Tree and Support Vector Machine (SVM). These techniques were selected due to their relevance in malware detection tasks and their ability to handle structured datasets efficiently.

2.2.3.1 Decision tree

A decision tree is a diagram used to support decision-making by visually mapping out different choices and their possible outcomes in a branching, tree like structure. Each node in the tree represents a decision point or condition based on an attribute, and each branch indicates the result of that condition, continuing until a final decision or classification is reached. This intuitive structure breaks down complex problems into smaller, manageable steps. In data mining, decision trees are widely used to create classification systems or predictive models based on multiple input variables. The method is non-parametric, making it effective for handling large and complex datasets without assuming a specific statistical distribution. When ample data is available, it can be split into training and validation sets where the training set is used to build the model and the validation set helps determine the optimal tree size to ensure accuracy and generalizability..Figure 2.3, shows an example of how decision tree works.[30]

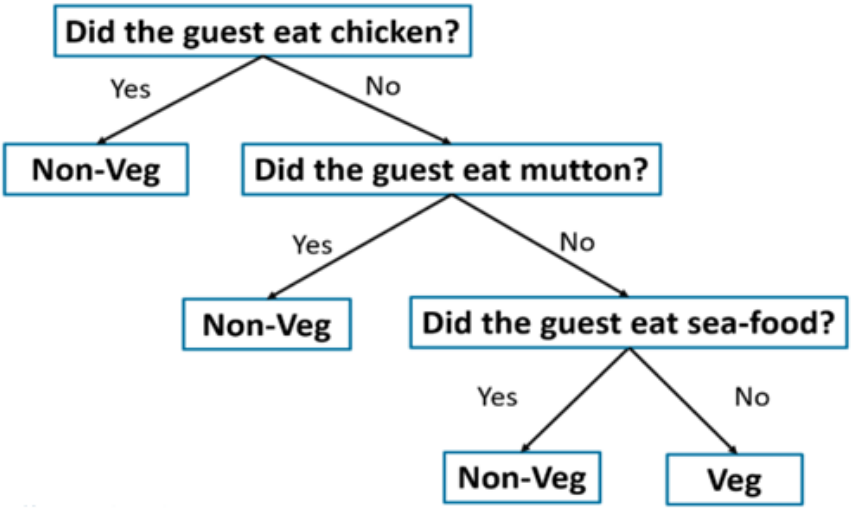

Figure 2.3: Decision Tree

- **Advantages of Decision Trees:**

  • Decision trees are very easy to understand and interpret, even for people without a technical or statistical background.

  • They require minimal data preprocessing there's usually no need for normalization or scaling.

  • They can handle both numerical and categorical data types.

  • They are capable of capturing non-linear relationships between features and outcomes.

  • Once trained, decision trees make predictions quickly and with low computational cost.

  • During the training process, they automatically select the most relevant features.

- **Limitations of Decision Trees:**

  • Decision trees are prone to overfitting, especially when they grow too deep without proper pruning, which can reduce their performance on new, unseen data.

  • They can be unstable — small changes in the input data might lead to a completely different tree structure.

  • They may be biased toward features with many categories, which can affect the accuracy.

  • Since they use a greedy approach to split nodes, they don't always result in the most optimal solution.

  • Compared to ensemble methods like Random Forests or Boosting, single decision trees often have lower predictive accuracy.[31]

2.2.3.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is another powerful and widely used machine learning method. It is a supervised learning algorithm used for both classification and regression tasks. SVM works by finding the best boundary, or margin, that separates different classes in the data. The goal is to draw this margin in a way that keeps it as far as possible from the nearest data points of each class, which helps reduce errors. Even when the data isn't linearly separable, SVM can handle it using a method called the kernel trick, which transforms the data into a higher-dimensional space to make it easier to classify.[30]

- **Strengths of SVMs**

  • Highly Effective Across Domains: Support Vector Machines consistently rank among the top-performing algorithms for both classification and regression tasks in fields ranging from bioinformatics to image recognition.

  • Better Generalization Through Margin Maximization: By seeking the widest possible gap between classes, SVMs naturally resist overfitting and tend to make more reliable predictions on new, unseen data.

  • Storage and Prediction Efficiency: Only the critical data points the support vectors are needed to define the decision boundary, which keeps model size small and makes predictions fast.

- **Limitations of SVMs**

  • Challenges with Very Large Datasets: Training standard SVMs on massive datasets can be slow and require large amounts of memory, prompting the development of specialized "large-scale" SVM variants.

  • Sensitivity to Class Imbalance: When one class heavily outweighs another, SVM performance can suffer unless specific strategies or modified algorithms for unbalanced data are employed.

  • Complex Parameter and Kernel Tuning: Getting the best results often means carefully choosing and tuning kernel functions and regularization parameters, which can be a time-consuming and nuanced process.[32]

2.2.4 Deep learning

Deep learning is a type of machine learning that helps computers learn from experience and understand the world by organizing knowledge in layers of concepts. Since the computer learns on its own from data, there's no need for a human to manually program every detail. It figures things out by building complex ideas from simpler ones, step by step. These layers of understanding form a deep structure kind of like stacking building blocks to reach more advanced knowledge.[33]

2.2.5 Deep learning approaches

2.2.5.1 Multilayer perceptron(MLP):

The multilayer perceptron is a commonly used neural network . MLP is composed of multiple layers, including an input layer, hidden layers, and an output layer, where each layer contains a set of perception elements known as neurons. Figure 2.4 illustrates an MLP with two hidden layers, an input and output layer. In interactions, each node displays a certain amount of bias.[34]

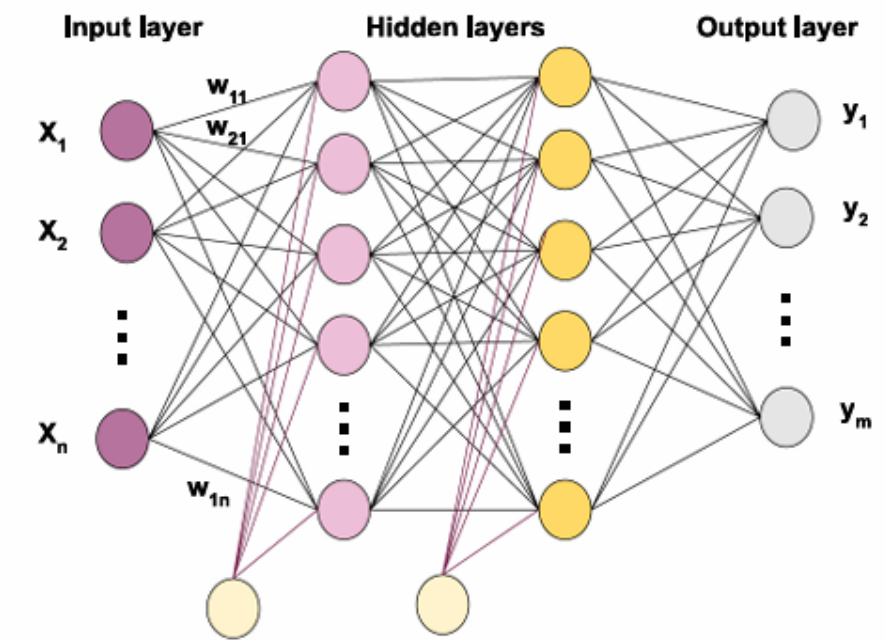

Figure 2.4: MLP topology.

2.2.5.2 Recurrent neural networks (RNNs)

Recurrent Neural Networks (RNNs) are a core deep learning architecture designed for sequential data. By adding loops in their structure, RNNs can maintain a hidden state that acts like memory, allowing them to carry information from earlier in the sequence and handle long-range dependencies unlike standard feedforward networks .

2.2.5.3 Long Short-Term Memory(LSTM)

LSTMs (Long Short-Term Memory networks) were developed to solve the vanishing gradient problem found in traditional RNNs (Recurrent Neural Networks). While the structure of an LSTM is similar to that of an RNN, its core building blocks the LSTM cells are specially designed to remember information over long periods.

Like RNN cells, LSTM cells take the previous output into account, but they also maintain a separate internal state known as the cell state, which serves as long-term memory. This means LSTMs can access both short-term memory (through the hidden state) and long-term memory (through the cell state), which is how they get their name.

You can think of the cell state as a conveyor belt that carries important information along the sequence without much interference. Inside each LSTM cell, there are special components called *gates* that control the flow of information deciding what to keep, what to forget, and what to add. These gates ensure that only the most relevant information is passed through, allowing the network to retain and learn from important patterns across time.

The key purpose of the LSTM cell is to manage how the long term memory is updated and maintained, so that information and gradients can flow smoothly during training [35]. To carefully regulate the cell-state, the LSTM uses the three yellow gates in Figure 2.5

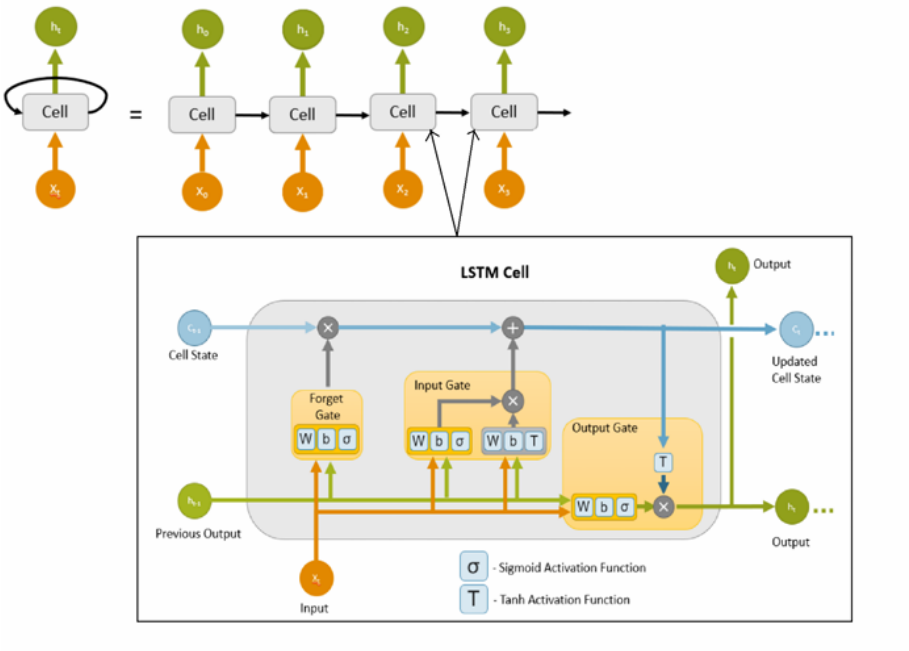

Figure 2.5: A schematic of Long Short-Term Memory(LSTM)

2.3 Android malware Datasets

- **KronoDroid dataset:**

  The original KronoDroid dataset includes malware samples collected between 2008 and 2020, making it well-suited for detecting evolving threats and addressing concept drift. One of its strengths is that dynamic features are extracted by running the malware on a real device, which helps effectively detect anti-dynamic malware that tries to avoid detection in simulated environments.

  To improve the dataset, they created a refined subset by adding malware category labels using information from various online repositories. They then trained several supervised machine learning models and used the ExtraTrees classifier to select the top 50 most important features. Among the models tested, the Random Forest (RF) achieved the best performance reaching 98.03% accuracy for detecting malware and 87.56% accuracy for classifying malware into 15 different categories.[36]

- **AndroZoo dataset:**

  AndroZoo is a large and continuously expanding dataset of Android applications gathered from multiple sources, including the official Google Play Store. It's designed to support research related to Android security and analysis.

  Currently, AndroZoo contains over 15 million unique APK files, and each app is analyzed by multiple antivirus engines to determine whether it is identified as malware. In addition, the dataset includes more than 20 different types of metadata for each app, such as VirusTotal reports, to provide researchers with rich contextual information for deeper analysis.[37]

- **Drebin dataset:**

To support Android malware research and allow fair comparisons between different detection techniques, the creators of the DREBIN project have made their dataset publicly available.

The dataset includes 5,560 malware samples spanning 179 malware families, collected between August 2010 and October 2012 with help from the MobileSandbox project.

DREBIN itself is a lightweight malware detection method designed to work directly on smartphones. Since mobile devices have limited resources and can't always monitor apps in real time, DREBIN relies on static analysis—it extracts a wide range of features from the application without running it. These features are mapped into a shared vector space, allowing the model to identify common patterns associated with malware. This also helps explain why a specific app was flagged as malicious.

In a large-scale evaluation using 123,453 applications, including the 5,560 malware samples, DREBIN achieved 94% detection accuracy with minimal false positives. The system also provides explanations for its decisions, highlighting the characteristics that led to detection. On average, the analysis takes just 10 seconds per app, making DREBIN fast and practical for real-time malware checks on actual smartphones.[38]

- **CIC-AndMal2017 dataset:** A dataset named CICAndMal2017 was introduced to support research in Android malware detection. In this work, the researchers executed both malware and benign applications on real smartphones instead of using emulators. This choice was made to avoid behavior changes that often occur with advanced malware when it detects it's running in a virtual environment.

  The dataset includes 10,854 samples in total, with 4,354 malware and 6,500 benign apps. The benign applications were collected from the Google Play Store and span across the years 2015 to 2017.

  From the full dataset, 5,000 samples (426 malware and 5,065 benign) were actually installed on real devices for analysis. The malware samples were categorized into four main types:

  Adware

  Ransomware

  Scareware

  SMS Malware

  In total, the malware comes from 42 different families, making the dataset diverse and suitable for evaluating malware detection methods.[39]

2.4 Evaluation Metrics in Security Systems

2.4.1 Confusion Matrix

Compared to abstract measurements, the confusion matrix [40] provides a more detailed understanding of categorization outcomes. Four scenarios comprise the outputs:

TP, or true positive: The assault has been recognized successfully.

FP, or false positive: They viewed sound traffic as a danger.

TN, or true negative: The right traffic is recognized.
FN, or false negative: danger disregarded and judged safe.

The significance of analysis False alerts caused by high FP are unpleasant. In the worst situation, threats that have a high FN remain unnoticed.

An example in numbers: TP=80, TN=900, FP=50, FN=20
Accuracy = 980/1050 ≈ 0.933
Precision = 80/130 ≈ 0.615
Recall = 80/100 ≈ 0.80
F1 ≈ 0.694

2.4.2 Accuracy, Precision, Recall, and F1-score

Accuracy: Performance indicators, including accuracy, predictive accuracy, recall, and F1-score, are essential for assessing how well malicious threat detection systems work in cybersecurity systems. These evaluations are more significant when the data is unbalanced that is, when there are fewer assaults than healthy traffic and are used to assess how well algorithms distinguish data between real threats and healthy traffic. Accuracy: denotes the proportion of accurate predictions whether positive or negative—to all the samples. provides a summary of the model's functionality.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

It is advised against depending only on this statistic in situations with imbalanced data, though, as the model may exhibit high accuracy even in the absence of threat detection.

Predictive accuracy, or precision: shows the percentage of samples that were properly identified as positive (TP) relative to all samples that were classified as positive (TP + FP).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.2}$$

demonstrates the accuracy with which the model can detect threats without raising false alarms.

Recall: : Evaluates how well the model can identify all positive genuine samples, or actual assaults.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.3}$$

Since ignoring a threat is riskier than raising a false alert, it is regarded as a crucial cybersecurity statistic.

F1-score : is used to assess models where sensitivity and accuracy must be balanced. It is the harmonic mean of precision and recall.[41]

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

Because it lessens the detrimental effects of biased results, it is perfect for assessing model performance in situations with imbalanced data.

2.4.3 ROC Curve and AUC

The association between sensitivity (TPR) and inverse type (FPR) with threshold variation is displayed by the Receiver Operating Characteristic (ROC) curve. The association between sensitivity (TPR) and inverse type (FPR) with threshold variation is displayed by the Receiver Operating Characteristic (ROC) curve.[42]

True Positive Rate (TPR):

$$\text{TPR} = \frac{TP}{TP + FN} \tag{2.5}$$

False Positive Rate (FPR):

$$\text{FPR} = \frac{FP}{FP + TN} \tag{2.6}$$

AUC (Area Under Curve) is the main indicator, where: AUC $\approx$ 1: The model is perfect AUC $\approx$ 0.5: The form is random

2.4.4 Other Metrics (e.g., FAR, FRR)

FAR and FRR have applications in intrusion detection systems and are used to assess recognition and verification systems.[43]

FAR (False Acceptance Rate):

$$\text{FAR} = \frac{FP}{FP + TN} \tag{2.7}$$

FAR (False Rejection Rate):

$$\text{FRR} = \frac{FN}{TP + FN} \tag{2.8}$$

2.5 Related Work

Recent studies on Android malware detection have focused on both centralized and decentralized approaches. While centralized methods achieve high accuracy, they raise privacy concerns due to data collection. Federated learning offers a decentralized alternative that enhances privacy by training models locally. This section highlights key works using both approaches for Android malware classification.

2.5.1 Centralized Learning Approaches

In centralized machine learning, data from all devices is collected and processed at a single location. While this setup often yields high accuracy due to access to a comprehensive dataset, it may introduce privacy risks and data transfer overheads.

Kaur and Laxmi [44] proposed a hybrid deep learning framework combining Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks for Android malware detection. CNN is used for spatial feature extraction, while LSTM models temporal dependencies. The proposed model was evaluated on a Kaggle dataset

and achieved a malware prediction accuracy of 98%, outperforming traditional models in precision and recall.

Chowdhury et al [45] conducted a comprehensive review of Android malware detection techniques, categorizing them into traditional machine learning (ML) and deep learning (DL) approaches. Their survey analyzed various models, including Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), Random Forests, Decision Trees, as well as more advanced architectures such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Deep Belief Networks (DBNs). The review emphasized that deep learning models, particularly CNNs and RNNs, consistently outperform classical ML algorithms in terms of accuracy, scalability, and feature abstraction capabilities. For example, CNNs are highly effective in automatically extracting hierarchical features from static and dynamic Android application data. The authors reported that CNN-based models achieved detection accuracies ranging from 94% to 98%, depending on the feature representation (e.g., permissions, API calls, opcodes) and the datasets used in the referenced studies. While the paper itself did not conduct experimental evaluations, it highlighted frequently used datasets in the field, such as Drebin and CICAndMal2017, which are commonly employed to benchmark model performance. The authors also noted critical challenges in the field, including dataset imbalance, overfitting, and a lack of standardized benchmarks, and suggested future work focus on developing robust, adaptive models and more diverse, real-world datasets.

Wang et al [46] proposed GDroid, a graph-based deep learning framework for Android malware detection and familial classification. GDroid models applications as heterogeneous graphs, capturing both app-to-API invocations and API-to-API usage patterns. These graphs are processed using Graph Convolutional Networks (GCNs) to learn meaningful representations. Evaluated on the AMD and Drebin datasets, GDroid achieved a malware detection accuracy of 98.99% and a familial classification accuracy of around 97%, significantly outperforming traditional methods. It also demonstrated strong robustness against code obfuscation and repackaged malware in real-world scenarios.

Liu et al [47] proposed a graph embedding-based deep learning framework for Android malware detection using static analysis. In their approach, Android applications are represented as program dependency graphs (PDGs), where nodes include semantic features from Smali code and permissions. These graphs are embedded into vector representations and fed into a deep neural network for classification. Evaluated on the Drebin dataset, their model achieved a detection accuracy of 96.7%, demonstrating strong generalization performance and explainability, especially in distinguishing malicious patterns based on structural and contextual code information.

Kim et al [48] introduced a multimodal deep learning framework for Android malware classification that integrates binary image representations and function call graphs. The binary images were analyzed using Convolutional Neural Networks (CNNs), while Graph Neural Networks (GNNs) were applied to the function call graphs to capture structural relationships. The outputs from both modalities were fused through a Multilayer Perceptron (MLP) for final classification. This multimodal ensemble approach demonstrated improved robustness and detection capabilities. Evaluated on a large malware dataset, their model achieved a high accuracy of 96.4%, outperforming single-modality models and traditional machine learning baselines.

2.5.2 Decentralized (Federated Learning) Approaches

To address privacy concerns and data sharing limitations inherent in centralized learning, federated learning (FL) has emerged as a promising approach for Android malware detection and classification. FL allows decentralized model training on user devices, where only model updates not raw data are shared and aggregated, preserving user privacy and reducing communication overhead.

Fang et al [49] introduced FEDroid, a comprehensive Android malware detection method built upon a federated learning architecture, with a strong focus on detecting not only existing malware but also evolving malware variants. The authors identified two major limitations in existing malware detection systems: first, a lack of adaptability to malware evolution, and second, the impracticality of centralized training due to data privacy issues and the contagious nature of malware samples. To overcome these, FEDroid incorporates a genetic evolution strategy that simulates malware evolution by generating malware variants from known samples. This approach enhances the model's ability to generalize and detect unseen or mutated threats.

The detection model in FEDroid is based on a residual neural network (ResNet), which is known for its deep feature learning and ability to avoid gradient vanishing. This model architecture enables high accuracy in distinguishing between benign and malicious applications. For privacy preservation and collaborative learning, FEDroid implements a federated learning framework that allows multiple Android security organizations to train a shared detection model without exchanging raw data. This decentralized training process protects sensitive user data and ensures compliance with privacy regulations.

The authors evaluated FEDroid on three benchmark datasets: CIC, Drebin, and Contagio, covering both static and dynamic features. The experiments were conducted under both local and federated settings. Results demonstrated that FEDroid's local model outperformed all baseline classifiers, while in the federated setting, FEDroid achieved superior performance compared to other state of the art methods. Notably, in cross-dataset evaluations, FEDroid attained an F1-score of 98.53%, indicating strong generalization. Moreover, through genetic evolution experiments using the Drebin dataset, the study confirmed that FEDroid is capable of detecting newly evolved malware, highlighting its effectiveness in real-world, dynamic threat landscapes.

This work stands out by addressing the dual challenge of malware evolution and privacy-preserving collaborative learning, making it one of the most advanced frameworks for scalable and secure Android malware detection to date.

Gálvez et al [50] proposed LiM (Less is More), a privacy-respecting Android malware classification framework based on federated learning. Unlike traditional approaches, LiM avoids sharing raw user data by keeping information about installed apps on-device. To address the lack of labeled data on clients, the framework uses a semi-supervised ensemble method that combines local data with a baseline classifier trained in the cloud. Experiments conducted on 50K apps with 200 users over 50 rounds showed a 95% F1-score at the server and perfect recall on client devices, with only one false positive per 100 apps. LiM also demonstrated robustness against poisoning and inference attacks, making it a secure and practical solution for real-world deployment.

Chaudhuri et al [51] proposed a dynamic weighted federated averaging (DW-FedAvg) strategy to enhance Android malware classification using federated learning. Unlike standard FedAvg, which treats all client models equally, DW-FedAvg assigns dynamic weights to local models based on their individual performance, reducing the negative impact of poorly trained models on the global aggregation. The approach was evaluated on four

widely used Android malware datasets Malgenome, Drebin, Kronodroid, and Tuandromd and demonstrated superior performance over traditional FedAvg in terms of accuracy, F1-score, AUC, and false positive rate. The results highlight the method's ability to provide privacy preservation, scalability, and improved classification accuracy in real-world, distributed mobile environments.

Jiang et al [52] proposed FedHGCDroid, a privacy-preserving Android malware classification framework based on federated learning. The approach integrates a hybrid model combining convolutional neural networks (CNNs) and graph neural networks (GNNs) to capture multi-dimensional malicious behavior features. To address the challenges of non-IID data across clients, the authors introduced FedAdapt, an adaptive training mechanism that assigns weights based on each client's contribution. Experiments on the Androzoo dataset under various non-IID settings showed that FedHGCDroid outperforms state-of-the-art models in terms of classification accuracy and adaptability, while preserving user privacy.

Hamouda et al [53] introduced a cost-effective Android malware detection model based on federated learning (FL) and network behavior analysis. Their approach leverages transfer learning to collaboratively train a global CNN model across distributed devices without sharing raw data, thus ensuring privacy. The model was evaluated using the AAGM-2017 benchmark dataset under multiple FL settings and compared to centralized training. Results demonstrated improved accuracy, precision, detection rate, and lower computational cost, confirming the efficacy of the FL approach in preserving privacy while effectively detecting large-scale malware threats.

Table 2.1: Summary of Android Malware Detection Studies using Centralized and Federated Learning

Reference	Dataset(s)	Approach	Key Contribution
Kaur and Laxmi[44]	Kaggle	CNN + LSTM hybrid model	Static analysis, high accuracy (98%)
Chowdhury et al[45]	Drebin, CICAndMal2017	ML/DL methods: SVM, k-NN, RF, CNN, RNN	DL models outperform ML in accuracy and scalability
Wang et al[46]	Drebin, AMD	GDroid framework using GCNs	Heterogeneous graphs, 98.99% detection accuracy
Liu et al[47]	Drebin	PDG-based graph embedding + DNN	Static analysis, 96.7% accuracy
Kim et al[48]	Large malware dataset	CNN + GNN fused via MLP	Binary images + function call graphs, 96.4% accuracy
Fang et al[49]	CIC, Drebin, Contagio	ResNet with genetic evolution in FL	Strong generalization, 98.53% F1-score, privacy-preserving
Gálvez et al[50]	50K apps (200 users)	Semi-supervised FL with local + baseline classifiers	95% F1-score, perfect recall on clients
Chaudhuri et al[51]	Malgenome, Drebin, Kronodroid, Tuandromd	DW-FedAvg strategy	Dynamic weighting improves accuracy, F1, AUC; reduces false positives
Jiang et al[52]	Androzoo	CNN + GNN hybrid with FedAdapt	Enhanced adaptability in non-IID settings, better accuracy
Hamouda et al[53]	AAGM-2017	CNN with transfer learning in FL	Efficient detection, low cost, improved accuracy and privacy

2.6 Conclusion

Federal learning is a change of paradigm for building resilient, distributed security systems, especially in environments like Android with a requirement to preserve user privacy and limit data transfer. Federated learning is a reasonable alternative to centralized models because models can be trained locally and knowledge shared securely. Through the observation of learning from evaluative scales, measuring performance encompasses looking at multiple scales (e.g., F1-score, AUC) and not just accuracy. The choice of

the clustering algorithm and data distribution also directly influences the efficiency of the model. According to previous studies, improvement in applied experiments, improvement of customer selection mechanisms, and testing of more advanced privacy protection techniques such as differential privacy or homomorphic encryption need to be explored in federal contexts. This is the future of intelligent information security and calls for greater collaboration between developers and researchers in order to create more efficient and sustainable models.

Chapter 3

Federated Learning for Android Malware Detection

3.1 Introduction

In the world of Artificial Intelligence (AI) and Machine Learning (ML), data availability plays a crucial role in building accurate and high-performing models. However, the way this data is stored, processed, and utilized varies significantly depending on the underlying infrastructure. Two primary approaches have emerged: centralized and decentralized learning. Centralized learning involves collecting all training data in a single location, which can raise serious privacy and security concerns. In contrast, decentralized models distribute the learning process across multiple devices or locations, aiming to mitigate these issues.

As smart systems evolve rapidly, inconsistencies in data handling and varied learning approaches have raised important questions: How are models trained? Where is the training performed? And why is there a growing interest in alternatives to traditional centralized techniques? The need for more accurate models without compromising user privacy has sparked new research directions.

Among these, Federated Learning (FL) has emerged as a promising paradigm. It offers a balanced solution by allowing collaborative training across devices while ensuring that sensitive data remains local. This innovative approach is paving the way for intelligent, privacy-preserving technologies that meet both performance and ethical requirements.

3.2 Centralized learning

Centralized learning is one of the traditional models utilized in machine learning systems, wherein training data coming from different sources are collected in one place, most often a master server, where all of the training and testing phases of the model are run. In this model, data is passed from terminals or peripheral devices to a central server that has sufficient processing capacity to process the data efficiently and build one model based on the general distribution of the collected information[54] .This model has several advantages, including Availability of a homogeneous and complete database that facilitates training accurate models Ease of handling and updating forms through a single central point More support for classic tools and advancements in machine learning.But this approach is not challenge-free. The greatest of these challenges are security and privacy issues, since transmitting raw data from the parties to the central server can lead to data leakage or hacking threats. Much of the data used in training operations may have personal or sensitive data, which must be secured tightly during transmission and storage [55].Additionally, total reliance on the central server creates what is known as a ”single point of failure” because any fault or attack on the server will slow down the entire system. The infrastructure also faces the risk of network bottleneck problems with the transportation of large amounts of data, especially in the case of large-sized or multi-sourced data sets [56].Due to such limitations, in recent years certain new approaches towards machine learning have emerged that seek to reduce reliance on centralization, such as distributed learning and federated learning, to provide people with more convenient and secure alternatives.Figure 3.1 shows Centralized learning architecture

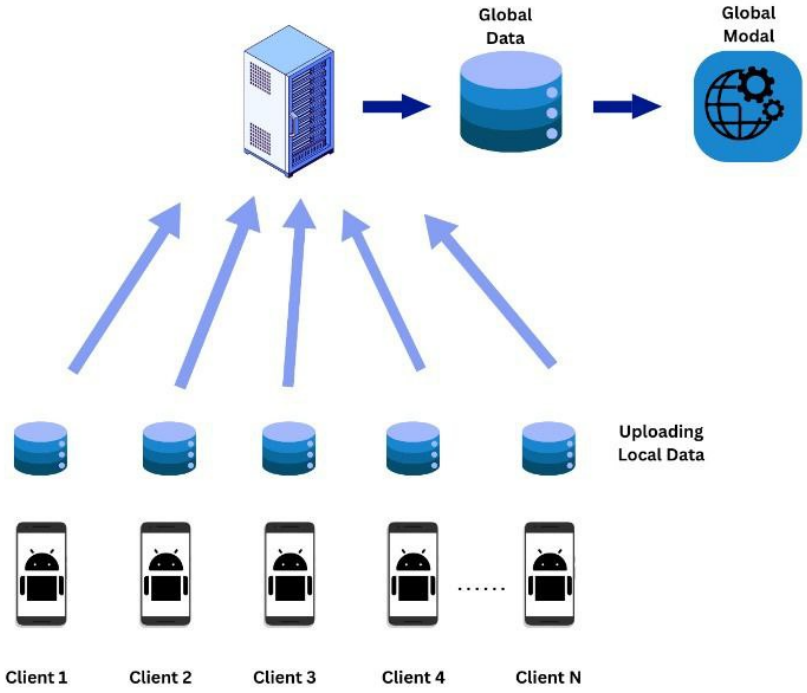

Figure 3.1: Centralized learning architecture

3.3 The approach used in this work

3.3.1 Convolutional neural network(CNN)

Convolutional Neural Networks (CNNs) are among the most powerful and widely used deep learning algorithms. They get their name from the mathematical operation called convolution, which is applied between matrices. CNNs are made up of several types of layers, including convolutional layers, non-linearity (activation) layers, pooling layers, and fully connected layers. While convolutional and fully connected layers contain learnable parameters, the pooling and non-linearity layers do not. CNNs have shown outstanding performance in many machine learning tasks, especially in fields that involve image data—such as large-scale image classification (like ImageNet), computer vision, and even natural language processing (NLP). In all these areas, CNNs have produced impressive and groundbreaking results.Figure 3.2 shows the architecture of the CNN model.[57]

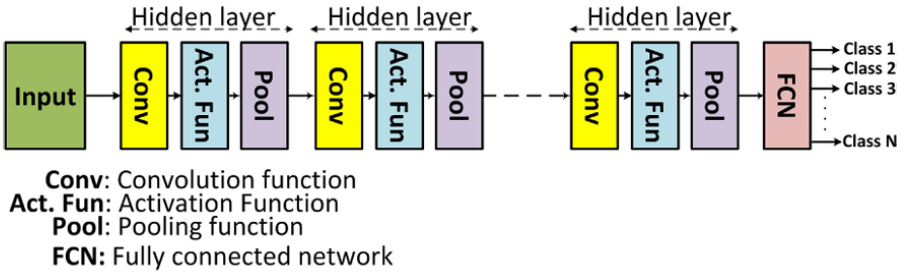

Figure 3.2: CNN architecture

3.3.1.1 Layers of convolution neural networks

In Neural Networks There is an input layer, one or many hidden layers and an output layer. All the layers have nodes and each node has a weight which is considered while processing information from one layer to the next layer

1. **Input layer:** in this layer, we supply our model with inputs. It is training information.

2. **Hidden layer:** the hidden layer receives the input from the input layer after that. Several hidden layers may exist, contingent on the amount of the data and our model. . Each layer's output is determined by matrix multiplying the output of the layer before it with its learnable weights, adding its learnable biases, and then applying an activation function to make the network non-linear

3. **Output layer:** a logistic function, such as sigmoid or softmax, receives the output from the hidden layer and uses it to translate the output of each class into a probability score for each class.

4. **Convolution layer:** A Convolutional Layer is a fundamental component of Convolutional Neural Networks (CNNs) used to automatically extract important features from input data, especially images. It works by applying small filters (also called kernels), such as 3×3 or 5×5 matrices, that slide across the input data. At each position, the filter performs an element-wise multiplication with the corresponding region of the input and sums the result into a single number. This operation is repeated as the filter moves (based on a parameter called stride), producing a new matrix known as a feature map. Padding may be used to control the output size, especially near the edges of the input. Each filter learns to detect specific features, like edges or textures, and multiple filters are typically used in a single convolutional layer to capture different patterns. As a result, the convolutional layer allows the network to identify and focus on the most relevant visual features without manual intervention.This figure 3.3 shows the Process of convolution layer

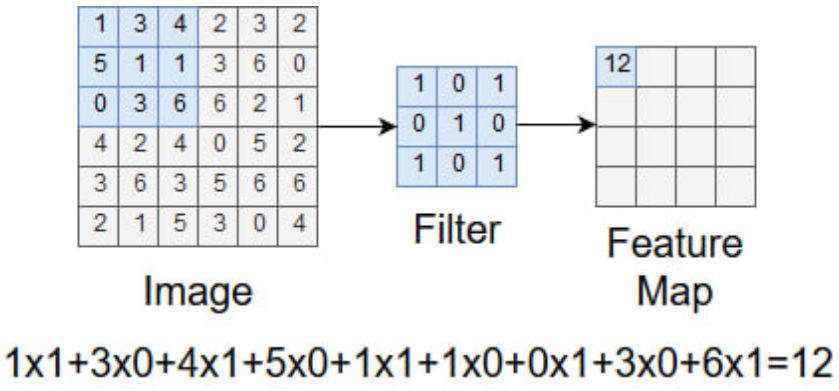

Figure 3.3: Process of convolution layer

5. **Activation layer :** ReLU stands for rectified liner unit and is a non-linear activation function which is widely used in neural network. The upper hand of using ReLU function is that all the neurons are not activated at the same time. This implies

that a neuron will be deactivated only when the output of linear transformation is zero. It can be defined mathematically as:

$$G(E) = \max(0, E) = \begin{cases} E & if\ E \geq 0 \\ 0 & else \end{cases} \tag{3.1}$$

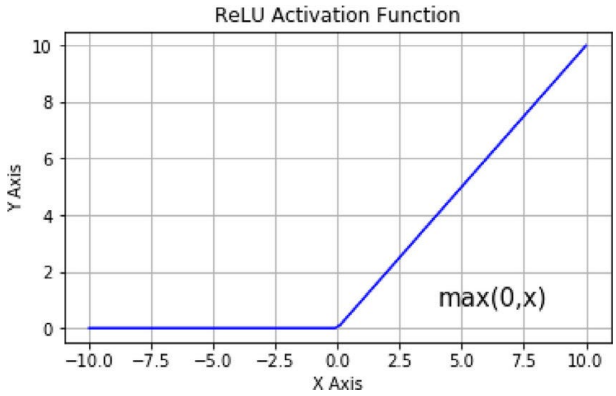

Figure 3.4: Activation Function ReLU

6. **Polling layer:** A pooling layer performs downsampling, which reduces the size of the feature maps. This helps the model become more robust to small shifts and distortions in the input and also reduces the number of parameters in the following layers. It's important to note that pooling layers don't have any learnable parameters—their behavior is controlled by hyperparameters like filter size, stride, and padding, just like in convolution layers.

   The most popular form of pooling operation is **max pooling**, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other patches. Figure 3.5 illustrates the process of max pooling.[58]

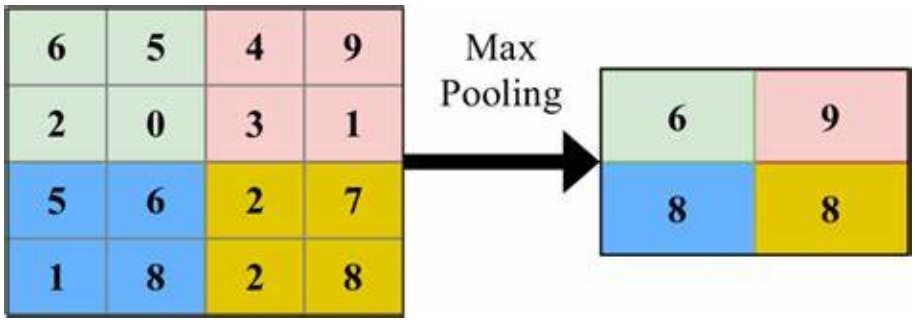

Figure 3.5: Process of max pooling layer

7. **Dropout Layer:** Dropout is a regularization technique used in deep neural networks to reduce overfitting, especially in large models with many parameters. During training, it randomly removes (drops) units and their connections, preventing neurons from becoming too dependent on each other. This process effectively trains many smaller "thinned" networks, and at test time, a single full network with scaled-down weights is used to approximate the averaged behavior of all those networks.[59]

8. **Flattening layer:** a Flattening layer converts the multi-dimensional output of convolutional and pooling layers into a one-dimensional vector. This step is necessary to connect the feature extraction part of the network to the fully connected layers used for classification or prediction. It preserves the extracted features while reshaping them for further processing.

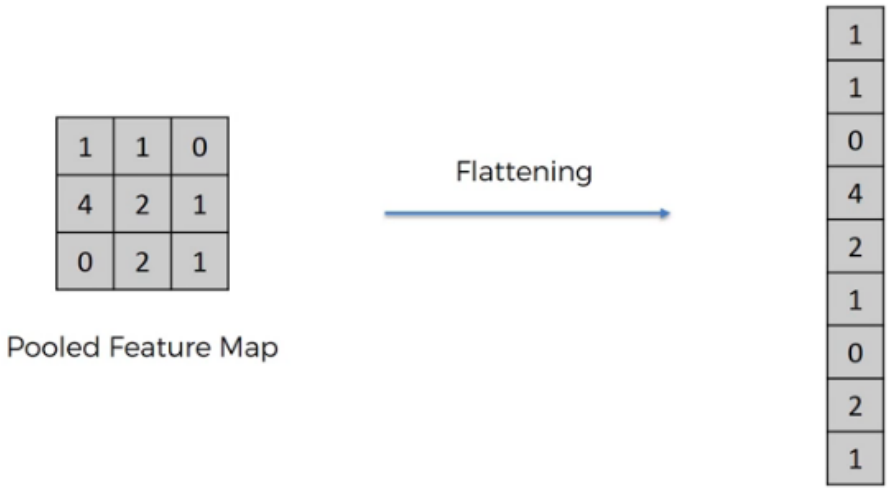

Figure 3.6: Process of Flattening layer

9. **Fully connected Layer:** A fully connected layer is usually placed at the end of a neural network and is mainly used for classification. Unlike convolution and pooling layers, which focus on local patterns, the fully connected layer performs a global operation it takes all the features extracted by the previous layers and analyzes them together. It then forms a non-linear combination of these features to make the final classification decision.[60]

3.4 Decentralized learning

A contemporary concept in machine learning called decentralized learning allows several devices or computational nodes to work together to train models without sending raw data to a central server. Using its own private dataset, each node trains locally in this method. It only communicates intermediate updates, such as model weights or gradients, with other nodes or aggregators that are part of the learning process. Because sensitive data is local and is not exposed during transmission or storage procedures, this architecture protects data privacy and improves system resilience [61]. Decentralized learning's capacity to lower bandwidth use and communication overhead is one of its main advantages. The total effectiveness of data handling is much increased because the training data does not need to be sent over the network. Additionally, scalability is supported, and reliance on centralized infrastructure is decreased by distributing the computational burden across participating nodes [62].Figure 3.7 shows Decentralized learning architecture

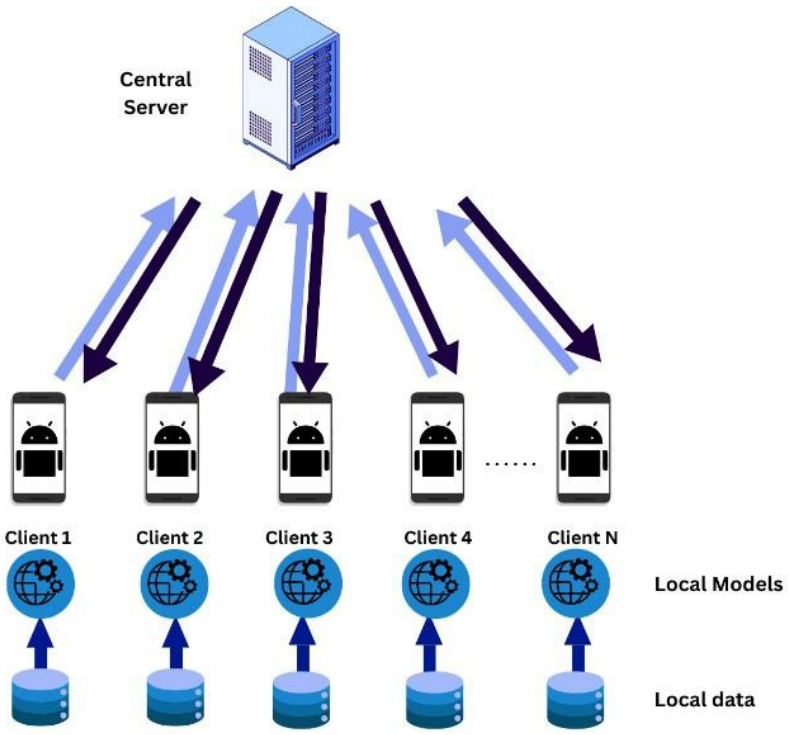

Figure 3.7: Decentralized learning architecture

3.5 What is Federated Learning

In the 2016 paper "Communication-Efficient Learning of Deep Networks from Decentralized Data" [5], Google first presented federated learning, a decentralized machine learning technique. By providing a privacy-preserving substitute for conventional centralized learning techniques, it signifies a paradigm shift in machine learning model training.

Data is gathered from multiple user devices and sent to a centralized server or data center in traditional machine learning pipelines. A global model is trained there using the aggregated data. Concerns about privacy, bandwidth usage, and data ownership are raised by this method, called centralized learning, which necessitates the constant transfer of possibly sensitive user data to external servers.

In contrast, federated learning does not require data centralization. Rather, it disperses the learning process among several devices, called clients, each of which stores local data. Only the modified model parameters (such as weights or gradients) are sent to a central server after each client trains a local model using its own data. A new global model is then created by aggregating these updates, usually with the help of algorithms like Federated Averaging (FedAvg). This model is then returned to the clients for additional training in the following round. Raw data never leaves the client device in this configuration. The user's privacy is maintained at every stage of the process by only communicating the learned model updates. Because of this, Federated Learning is especially well-suited for privacy-sensitive applications like personalized recommendations, finance, and mobile health.[5]

3.5.1 Categorization of federated learning

1. **Centralized Federated Learning**

   A central server coordinates the training process in centralized federated learning by gathering model updates from several clients, combining them, and then sending the modified global model back to the clients. Because of its ease of use and effectiveness, this architecture is widely used in mobile and Internet of Things applications. It adds a single point of failure, though, and the entire training process may be interrupted if the central server malfunctions or is hacked.[63]

2. **Decentralized Federated Learning**

   With decentralized federated learning, a central server is not required. Rather, clients exchange and aggregate model changes through direct peer-to-peer communication. While central reliance is decreased and fault tolerance is increased, this method adds complexity to peer coordination, synchronization, and communication protocols.[64]

3. **Hybrid Federated Learning**

   Both centralized and decentralized methods are combined in hybrid federated learning. While clients also carry out local aggregations or share updates with nearby clients prior to contacting the server, a central server may be present to handle coordination and partial aggregation responsibilities. Because of its scalability and flexibility, this hybrid configuration is appropriate for intricate or expansive networks.[65]

3.5.2 System Architecture

1. **Vertical Federated Learning**

   Vertical Federated Learning can be used when datasets belonging to separate parties have distinct characteristics for the same consumers. An e-commerce business and a bank, for example, can have separate data about the same clients. Without disclosing their raw data, VFL enables different organizations to work together to train models, frequently using secure computing methods to preserve privacy.[66]

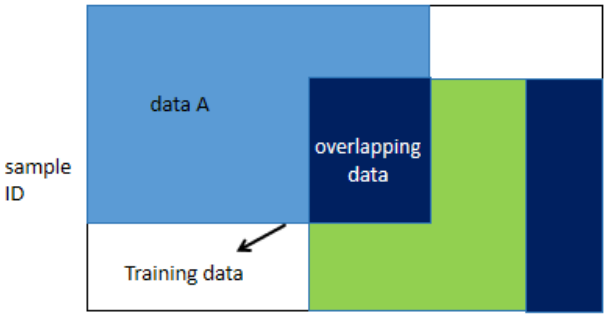

Figure 3.8: Vertical Federated Learning

2. **Horizontal Federated Learning**

Horizontal Federated Learning is used when datasets from several devices or organizations have different samples but the same feature space. For instance, two hospitals in separate places could gather the same kinds of medical information from patients in different places. Without disclosing raw data, HFL allows various organizations to work together to develop a machine learning model while maintaining privacy.[67]

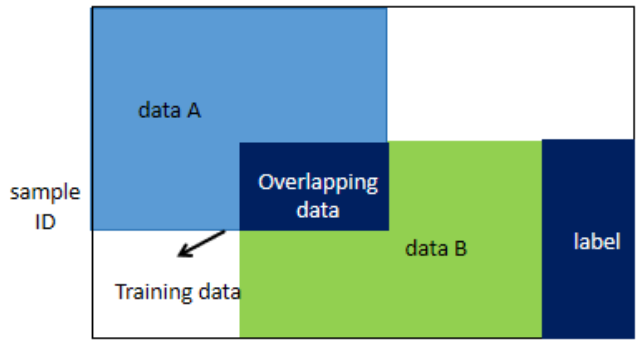

Figure 3.9: Horizontal Federated Learning

3. **Federated Transfer Learning**

   Federated Transfer Learning is applied when participant datasets have little overlap and varies in terms of both characteristics and samples. When firms from many domains want to work together, this situation frequently arises. By using transfer learning strategies, FTL makes it easier to collaborate even in diverse environments by transferring knowledge from one area to another.[68]

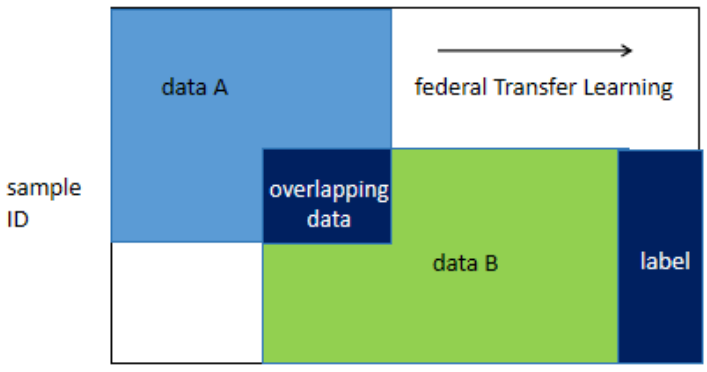

Figure 3.10: Federated Transfer Learning

3.5.3 Key Concepts

Federated learning is based on a very different model in terms of how information is structured and shared compared to traditional well-known machine learning systems, the latter of which relies on transferring data to a central server to train the model, other than machine learning that defines its structure, how tasks are distributed between the

participating devices and how the group learning process is coordinated without compromising data privacy.We will then explain the most important elements that form the core of this system, starting with the client-server structure and ending with local training and model aggregation.

1. **Client-server architecture**

   The structure of the client and server is one of the fundamental architectural models that support federal learning. This strategy necessitates the creation of a particular protocol that permits data flow between various devices (customers and servers) since it is based on the division of labor between two groups of entities : customers and servers. According to Oluwatosin (2014), this topology depends on transport layer protocols like TCP to provide dependable data transmission. Information must be planted across several levels of the reference OSI model to guarantee that it is comprehended and processed at every level. Within the communication and processing cycle, the client and server each have a distinct function that enhances the other in the client-server architecture.[69]

   - **Client :** Gets the person who initiates the connection and makes the service request ready. It is used to run applications and access forms or data that the server maintains. Client systems are easier to build and implement since they often have simpler designs than servers and don't require sophisticated system features.
   - **Server :** It is in charge of offering the services that clients need. Execute procedures like data processing, model aggregation, or result saving. The server could need high system rights because of how sensitive its activities are. In order to prevent any security issues or abuse, it is crucial to make sure that these benefits are not transferred to clients while creating a safe framework.[70]

2. **Local training and model aggregation**

   Federated learning is based on the elemental rule of actualizing training at the fringe level, instead of exchanging information to a central server. In this setting, each client carries out a nearby demonstrate preparing work out utilizing their claim information, which remains on the gadget without taking off. This plan gives a tall level of protection assurance, restricting the sharing of delicate information with third parties or transmitting it through open systems, in this manner decreasing the probability of cyberattacks or information spills. Nearby preparing takes after a timetable facilitated by the central server, so that each preparing circular starts by sending the current worldwide show to partaking gadgets, which it employments as a beginning point for overhauling the show based on its neighborhood information. After a few preparing steps (as a rule, a particular number of ages), overhauled shapes are sent to the central server and not the information itself, making this prepare compelling in terms of bandwidth-efficient consumption and congruous with protection prerequisites. Within the central server, these models are coordinated into a process known as show accumulation. The Unified Averaging (FedAvg) calculation is one of the most broadly utilized get together calculations, to begin with presented by McMahan et al. (2017). This calculation calculates the

weighted normal of each show parameter over all clients, with each upgrade given weight commensurate with the volume of nearby information prepared. In this way, each circular comes about in a new worldwide show that gathers the encounters picked up from diverse clients and is redistributed once more within another circular. This strategy accomplishes the rule of decentralized collaborative learning, which combines the utilization of the extraordinary diversity of users' information with regard for their security. In addition, this approach permits made strides by and large execution of the demonstrate without relinquishing information security, which makes it reasonable for delicate applications such as healthcare, managing an account administrations, and smartphone applications.[5]

3.5.4 Federated Learning Process

The federated learning process consists of a sequence of iterative and structured steps designed to train a global machine learning model without sharing raw data, preserving privacy and enhancing communication efficiency.

Step 1: Global Model Initialization

At the start of each training session, a central server initializes a shared learning model, which is distributed to all clients. This model can either be randomly initialized or based on a pre-trained one.[71]

Step 2: Client Selection

The server selects a subset of active clients according to predefined criteria such as device availability, battery level, network connectivity, or geographic distribution.[5]

Step 3: Local Training

Each client trains the received model on its local data using a specified algorithm, commonly Stochastic Gradient Descent (SGD), without sharing any raw data with the server.[72]

Step 4: Model Update Communication

After local training, clients send only their model updates (e.g., weight changes) back to the server instead of raw data, minimizing privacy risk and network load.[4]

Step 5: Aggregation

The server aggregates the updates received from clients using an algorithm such as Federated Averaging (FedAvg), where contributions are typically weighted based on local dataset size.[5]

Step 6: Model Update and Re-distribution

The updated global model is then redistributed to the selected clients, and the process is repeated for several rounds until convergence is achieved and the model reaches optimal performance.[71]

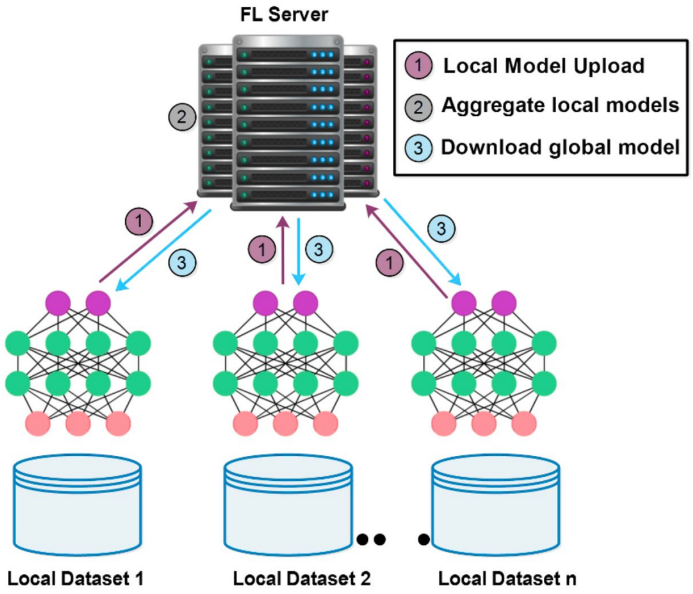

Figure 3.11: Federated Learning architecture

3.5.5 Advantages

Given the rising concern about data privacy and adherence to laws like GDPR in Europe and HIPAA in America, federated learning signifies a significant change in the way AI models are developed and trained. Although there have been difficulties, this method has shown promise.

1. **Data privacy preservation**

   One of the fundamental motivations behind the development of Federated Learning (FL) is the protection of user privacy. With the widespread use of smart devices—such as smartphones, smartwatches, and IoT systems—vast amounts of personal data are generated daily. Unlike traditional centralized learning models that require collecting all user data on a single server, FL enables each device to train locally and only send model updates (like weights and biases), not raw data. Privacy is preserved because this sensitive data remains on the device. Technologies such as Secure Aggregation, Differential Privacy, and Secure Multi-Party Computation (SMPC) further enhance privacy. Secure Aggregation ensures that servers can only see the aggregate of encrypted updates. Differential Privacy adds statistical noise to prevent data reconstruction, and SMPC allows computations across multiple devices without exposing individual data. Using algorithms like FedAvg, the server combines these updates into a global model. This approach is especially important in sensitive fields like healthcare, finance, and privacy-focused applications such as Google Gboard and smart home devices.[5]

2. **Reduced need for data transfer**

   Federated Learning distinguishes itself from traditional machine learning by eliminating the need to transfer raw data to a central server. Instead of sending user data such as images, text, or behavioral logs, each client device keeps the data locally and only transmits model updates (e.g., weight changes) after local training. This significantly reduces network strain. Techniques like epoch-wise aggregation allow

clients to perform multiple local training cycles before sending updates, minimizing communication frequency and enhancing efficiency. Since training occurs locally, devices can continue learning even during poor or lost connectivity, making federated learning ideal for environments with limited infrastructure, such as rural areas. It is also well-suited for low-power IoT devices, drones, and autonomous vehicles. Moreover, the reduced data transmission leads to lower operational costs, especially in industrial applications dealing with large-scale sensory data. A practical example of this is Google's Gboard app, which uses federated learning to improve word prediction without sending users' messages to central servers, thus preserving privacy and saving bandwidth.[54]

3.5.6 Challenges of distribution and client heterogeneity

Federated learning has several advantages, such as lowering data transport and protecting privacy, but its broad use in intelligent systems is hampered by a number of algorithmic and architectural issues. We go over the most notable of these difficulties below:

1. **Client Heterogeneity** The fact that the clients in federated learning are actual devices dispersed throughout many contexts, including computers, cellphones, Internet of Things devices, and others, is one of the main issues. Due to this substantial variability, there are notable variations in :

   computational capability (some devices can't finish a single training cycle, while others have strong CPUs).

   Network connectivity (for instance, while certain devices are constantly connected, others disconnect often).

   Power availability or battery capacity.

   Because of this, certain clients might not take part in all training sessions or they might cause updates to be delayed, which would make the training process unstable and the unified model less effective.[54]

2. **Non-IID Data Distribution:** Data variance is the non-independent and uniform distribution of data. The great majority of centralized learning systems use equally dispersed (IID) data to train the model. Devices differ greatly in the data they utilize for federated learning. The data does not precisely represent the worldwide distribution because every client gathers their own data. Biased data may be produced by certain clients. The accuracy of the unified model decreases when it is used on fresh data. This obstacle is regarded as one of the most difficult problems, as maintaining user data is frequently tough. In order to identify suggested alternatives like FedProx or rebalancing enhancements, research and development is currently continuing.[73]

3. **Intermittent connection and client dropout in federated learning:** One of the key technical challenges in federated learning is client dropout, which occurs when certain devices fail to consistently participate in training rounds. This issue arises due to various reasons, such as battery depletion in power-limited devices like smartphones or sensors, unstable internet connections—especially in remote or underserved regions—and interruptions caused by system crashes or users closing the application. Additionally, devices often prioritize user-initiated tasks, such

as calls or app usage, over background training processes. These disruptions can significantly affect the synchronization of the federated learning process. If many clients are absent in a given round, the resulting model updates may not reflect the overall data distribution, potentially leading to biased or inaccurate outcomes. Furthermore, dropout can result in inefficient use of computational resources and delays in model convergence. In time-sensitive applications such as healthcare or autonomous systems, the lack of timely updates may hinder decision-making and compromise system reliability.[74]

4. **Risks to Privacy and Security :** Federated learning is marketed as a method that enhances privacy protection by storing data locally, however, total security is not assured. This is due to the possibility that implicit information in updates transmitted to the central server, such as weights or gradients, might be utilized to get user data.[75]

3.6 Importance of Federated Learning in Android Environments

Android has become the most popular mobile operating system globally due to the exponential rise in the use of mobile devices. However, because of its widespread use, it has become a prime target for cyberthreats, especially malware assaults that jeopardize device integrity and user security. Scalability, latency, and above all privacy issues are some of the main issues with traditional malware detection methods, which usually depend on centralized data collecting and processing. Federated Learning (FL), a potent, privacy-preserving method that permits cooperative machine learning across several Android devices without sending raw data to a central server, has surfaced in response to these constraints.

Each Android device trains locally on its own private data in a federated environment. Only the learned model updates such as weights or gradients are sent to a central server, which combines them to enhance a global model. The system can adjust to the varied and changing nature of malware across various devices and user behaviors thanks to its decentralized design, which also protects data privacy.[76]

The benefits of on-device learning and real-time adaptation, the main reasons for utilizing FL in Android-based malware detection, and the safeguards that guarantee privacy-preserving malware identification are all covered in the sections that follow.

3.6.1 Motivation for Using Federated Learning with Malware

Due to the growing global dependence on Android smartphones, malware authors have turned these platforms into their top targets. Securing mobile applications has become essential as they still handle private user data, including location, financial information, and private messages. Conventional malware detection methods frequently depend on centralized structures in which distant computers gather and process user data. However, this strategy frequently fails to scale well with the rapidly increasing amount of data on Android users and poses serious privacy problems.

A paradigm shift in the training of machine learning models is brought about by federated learning (FL). FL enables individual devices (clients) to train models locally

using their own data, rather than sending raw user data to a central server. Without ever having access to the sensitive data below, these models are then combined on a single server.

FL is used in Android-based malware detection for the following reasons:

- Data privacy : guarantees that personal information, including browser history, app activity, and communications, never leaves the user's device.
- Scalability: Because Android is so widely used, centralized data collecting is not feasible. Without putting undue strain on central infrastructure, FL enables training to expand across millions of devices.
- Diversity of Data: The localized nature of FL enables models to learn from a broad range of attack vectors, since malware variants frequently take advantage of device-specific or user-specific behaviors.

  Studies conducted in the real world, like McMahan et al. (2017), showed how FL can be effectively used for keyboard prediction use cases (like Google Gboard), providing information on the viability of applying comparable methods for malware detection.[5]

3.6.2 On-device learning and real-time adaptation

The capability of FL to conduct on-device learning is among its most promising features in Android contexts. With the growing number of capable processors in Android devices, lightweight models like MLPs, decision trees, or even quantized neural networks may now be trained locally.

On-device learning has a number of benefits are:

Personalized Detection: Depending on the unique user behaviors and environments of each device, malware detection models can be customized. A user who often installs programs from third-party sources, for example, can run into different malware dangers than someone who exclusively utilizes official app stores. FL enables local models to learn and change based on such unique behavioral patterns, boosting the accuracy of malware detection.

Android systems are able to adapt to the unique risk profile of their users thanks to this personalization. Detection methods that prioritize heuristic and behavior-based analysis may be used to users who participate in dangerous activities (such as sideloading programs or visiting unprotected websites). However, lightweight signature-based detection could be advantageous for users in highly regulated settings. By learning from environmental cues such as the frequency of user interactions, permission usage, or unusual patterns in app activity, the locally trained models gradually improve in efficacy.

Reduced Latency: Because data must be delivered, analyzed, and returned to the device remotely, traditional cloud-based malware detection solutions sometimes have significant delay. FL ensures a quicker reaction to harmful activities by making decisions immediately on the device.

In order to combat fast-acting malware, such as ransomware, which can do irreparable harm within seconds of execution, this detection time reduction is essential. Threats can be detected before harmful acts, such encrypting user files or transmitting sensitive information to command-and-control sites, are carried out thanks to on-device detection.

FL-based systems can also operate offline or with intermittent connectivity, making them robust in real-world environments where network access may be limited.

Real-Time Updates: Rapidly evolving malware can affect Android devices that operate in dynamic contexts. Without waiting for centralized retraining or model updates, on-device learning enables models to instantly adjust to new threats. The malware detection system's resilience is increased by this ongoing cycle of adaptation.

FL-based systems enable devices to continually update their own defenses, as contrast to traditional systems that depend on recurring signature updates sent from the cloud. For instance, new behavioral indications may be employed right away to improve local models as they are found, such as unusual system calls, suspicious background activity, or illegal data transmissions. These updates improve the global model and are distributed to all clients after being combined and verified via federated learning, allowing for a group immune response to new threats.

Example: Suppose an Android device encounters a previously unknown Trojan malware that bypasses the existing global detection model. With on-device learning, the local model on the device can quickly adapt by analyzing behavioral patterns, such as abnormal permissions requests, unusual battery drain, or suspicious network activity. This updated local model can then send encrypted updates to the central federated server. As a result, the global model is refined using this new information and redistributed to all other devices, thus strengthening the system against emerging threats in near real-time.[14] Furthermore, Android's software development frameworks (like TensorFlow Lite and PyTorch Mobile) have increasingly supported on-device training and inference, making it technically feasible to implement such learning pipelines directly on smartphones without overtaxing their resources.[72]

3.6.3 Privacy-preserving malware detection

The capacity to protect user privacy is a primary driver behind the usage of federated learning in Android malware detection. Conventional cloud-based solutions include sending raw dataincluding private data like location history, app activity, and file contents to distant servers for analysis. This raises ethical and legal questions about user permission and data governance in addition to increasing the likelihood of data breaches. By guaranteeing that raw data never leaves the user's device, federated learning (FL) solves these problems. Rather, only model changes are sent to a central server, including gradient information. Because these changes are securely aggregated, there is less chance that private user information might be reverse-engineered. In addition to complying with data privacy laws like the General Data Privacy Regulation (GDPR), this approach drastically lowers the attack surface for malevolent actors.

- **Data Minimization:** FL complies with the data minimization concept, which is a fundamental element of contemporary data protection regulations, by sending just the most important model parameters rather than raw data. This restricts exposure and guarantees adherence to global privacy frameworks[71]

- **Anonymity and Differential Privacy:** By utilizing strategies like safe aggregation, which makes sure the server cannot view individual client updates, and differential privacy, which introduces stochastic noise to model updates, FL implementations can further improve privacy. Even in the presence of adversaries, these

measures reduce the danger of re-identification by making it very difficult to link updates to a particular user or device[77]

- **Trust and User Acceptance:** Trust is crucial when it comes to malware detection. Users are more inclined to choose security measures when they are aware that their personal information is still on their device. By broadening the data environment, this increased involvement enhances the FL ecosystem and boosts the global model's resilience and generalization [71]. As an example, suppose an Android device discovers a new spyware program that records keystrokes and transmits the information to a distant server. Based on system call patterns and app behavior, the device can train a local model to identify this behavior instead of sending comprehensive activity logs to a centralized server for analysis. Only model changes that are encrypted and anonymized are transmitted to the central server. These changes are added to other clients' updates via safe aggregation, which improves a common global model. This cooperative procedure improves detection accuracy without ever gaining access to or disclosing the user's private input data. FL offers a convincing foundation for creating morally sound and safe malware detection systems for Android by keeping sensitive data decentralized and protecting user privacy with cutting-edge cryptographic and algorithmic approaches.

3.7 Federated Learning Strategies in Malware Classification

As federated learning develops further, a number of methods and approaches have been developed to increase its efficacy in the field of malware categorization, particularly in Android ecosystems. With an emphasis on aggregation methods, data distribution scenarios, and client selection processes, this section describes the main tactics utilized in federated learning. These tactics aid in resolving the fundamental issues of adversarial robustness, device heterogeneity, and non-IID data.

3.7.1 Model Aggregation Strategies

Federated learning depends on a number of optimization strategies that determine how client-trained local models are aggregated into a global model. Three of the most popular methods for classifying malware are FedAvg, FedProx, and FedMedian:

1. **FedAvg**

   Developed by McMahan et al., FedAvg computes a weighted average, with each client contributing proportionately to the dataset size, in order to aggregate local model weights. It is easy to use, effective at communicating, and frequently utilized in real-world FL applications.

   - **Advantages:** High scalability and low communication costs.
   - **Cons:** Sensitive to non-IID data heterogeneity.
   - **Mathematical Formulation:** Let be the local model weights of client at round and be the number of samples on client k . The global model update is computed as:[5]

optimizes the global objective:

$$w^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^t \quad \text{where} \quad n = \sum_{k=1}^{K} n_k \tag{3.2}$$

where:

- w^{t+1}: global model after round $t+1$,
- w_k^t: local model of client k at round t,
- n_k: number of data points on client k,
- n: total number of data points across all clients.

Use Case in Malware Detection: FedAvg works well in settings where client data is not heavily skewed and is somewhat balanced. It performs best during the early stages of training or when customers have comparable use patterns.[17]

2. **FedProx**

   A FedAvg extension that adds a proximal term to the local optimization goal. By penalizing significant departures from the global model, this term strengthens the algorithm's resistance to heterogeneous inputs.

   - **Advantages:** Enhanced stability in a variety of environments.
   - **Cons:** The proximal regularization parameter has to be adjusted.
   - **Mathematical Formulation:** Mathematical Formulation: Clients solve the following optimization problem locally:[4]

     extends FedAvg by including a proximal term to handle data heterogeneity. Each client solves:

$$\min_w F_k(w) + \frac{\mu}{2} \|w - w^{(t)}\|^2 \tag{3.3}$$

     where $\mu > 0$ is a regularization parameter, and $w^{(t)}$ is the global model from the previous round and

     - $f_k(w)$: Local loss function on client k
     - w^t: Current global model
     - μ: Proximal term coefficient

     This proximal term $\frac{\mu}{2}\|w - w^{(t)}\|^2$ restricts the local updates from deviating too far from the global model, which is particularly helpful in handling system and data heterogeneity across clients.

Where is the local loss function, is the current global model, and is the proximal coefficient. Practical Advice: FedProx is frequently used in situations when customers have a large amount of non-IID data, such Android malware detection situations where user exposure to threats and app use differ greatly. By preventing local updates from deviating from the global learning trajectory, the proximal term aids in constraining them.[18]

3. **FedMedian**

   FedMedian computes the median of model weights to aggregate local models rather than average them. As a result, it is more resistant to hostile customers or disruptive changes.

   - **Advantages:** Sturdy against Byzantine clients and outliers.
   - **Cons:** It could converge more slowly than average techniques.
   - **Mathematical Formulation:** For each model parameter dimension , the global update is:[78]

     is a robust aggregation method that computes the median of each model parameter across clients:

     $$w_j^{(t+1)} = \text{median}(w_{1,j}^{(t+1)}, w_{2,j}^{(t+1)}, \ldots, w_{K,j}^{(t+1)}) \tag{3.4}$$

     where:

     – $w_{k,j}^t$: The j-th coordinate of the model from client k
     – d: Number of model parameters

     This approach improves fault tolerance against adversarial or noisy updates.

   Relevance to Security: FedMedian is especially helpful in hostile settings, including those where hacked or malware-infected Android devices could deliver altered updates. This approach improves model resilience by removing excessive or distorted results by depending on the median rather than the mean.

   Convergence speed, robustness, and performance under different data distribution situations are trade-offs for each of these techniques. The particulars of the client demographic and the malware detection task at hand will determine which approach is best.

3.7.2 Data distribution

In federated learning, the performance of the global model is significantly influenced by how data is distributed among clients. Two key aspects of this distribution are data heterogeneity (IID vs. Non-IID) and data balance (balanced vs. unbalanced).

3.7.2.1 Data Heterogeneity (IID vs. Non-IID)

- In IID (Independent and Identically Distributed) settings, each client holds data that is representative of the overall distribution. This uniformity typically leads to faster convergence and more stable training outcomes, especially when using standard aggregation methods such as FedAvg.[5]

- In contrast, Non-IID distributions are more reflective of real-world scenarios, especially in Android malware detection, where user behavior, geography, and app usage vary greatly. This means certain clients may only observe specific types of malware, causing inconsistency in local models and degrading the performance of the global model. Addressing this heterogeneity often requires specialized strategies like client clustering or adaptive aggregation.[71]

3.7.2.2 Data Balance (Balanced vs. Unbalanced)

- A balanced dataset ensures that each client has roughly the same amount of data, which supports fair contribution during model aggregation.
- However, in practice, data is often unbalanced, with some clients holding large datasets while others have very little. This can lead to biased global models dominated by data-rich clients. To mitigate this, techniques like SMOTE (Synthetic Minority Oversampling Technique) are employed to oversample minority classes or smaller datasets. SMOTE generates synthetic examples, improving class balance and helping reduce bias in training, particularly when detecting rare malware types.[4]

3.8 Challenges and Future Directions of Federated Learning

3.8.1 Addressing Client Heterogeneity

One of the biggest problems with federated learning environments is client heterogeneity. Federated learning uses a large number of devices with widely disparate capabilities and data distributions, in contrast to centralized machine learning, where all data is collected and processed in a homogenous infrastructure. Hardware characteristics (CPU, memory, battery life), network bandwidth, and—above all—the kind and distribution of local data, which is frequently non-IID (not independent and identically distributed), are some examples of these variations. These disparities result in problems including unstable global aggregation, biased local models, and stragglers (slow devices). Numerous tactics have been put forth to lessen these issues. By adding a proximal element to the local goal function, FedProx, for example, limits updates to stay near the global model and lessens the divergence brought on by non-IID data. Alternative approaches include grouping clients according to shared traits or dynamically modifying the training load in response to device performance. For FL systems to be deployed in the real world in a way that is reliable, equitable, and scalable, handling client heterogeneity is crucial.[4]

3.8.2 Enhancing Privacy and Security

Federated learning's ability to improve data privacy by guaranteeing that raw data never leaves the user's device is one of its main driving forces. But even with its decentralized structure, FL is not impervious to security and privacy threats. It is still possible for adversaries to try to contaminate the model by adding malicious gradients or to deduce private information from shared model updates. To address these challenges, researchers have proposed a variety of privacy-preserving techniques. To stop individual data points from leaking, Differential Privacy (DP) adds statistical noise to local updates prior to aggregation. The central server may merge encrypted updates using Secure Aggregation without having access to the underlying values. During computing, data is further secured using homomorphic encryption and trusted execution environments (TEEs). Byzantine-resilient aggregation techniques further shield the system against clients that provide tainted updates or act maliciously. In industries like healthcare, banking, and education,

where data sensitivity is high and regulatory compliance (e.g., GDPR, HIPAA) is required, privacy-enhancing solutions are essential for implementing FL.[77]

3.8.3 Federated Reinforcement Learning (FRL)

Federated Reinforcement Learning (FRL) is an emerging paradigm that combines the decision-making power of reinforcement learning (RL) with the privacy-preserving architecture of federated learning. In traditional RL, an agent interacts with an environment to learn optimal actions through rewards and penalties. In FRL, multiple agents (clients) independently interact with their local environments and share learned policies or Q-functions with a central server or peer clients without revealing raw interaction data. This is particularly useful in domains where direct data sharing is infeasible, such as autonomous driving, industrial robotics, and personalized healthcare. FRL faces unique challenges due to the asynchronous nature of agent-environment interactions and the non-stationarity of shared updates. Moreover, ensuring convergence and maintaining exploration across distributed environments require sophisticated coordination mechanisms. Researchers are exploring hybrid algorithms that combine policy gradient methods with federated averaging, and communication-efficient techniques to reduce the overhead of frequent policy sharing. FRL holds great promise in creating intelligent systems that learn collaboratively while respecting privacy and operational autonomy.[79]

3.9 Conclusion

Federated learning has become a key approach in the field of Android malware detection through support of decentralized, privacy-protecting model training on distributed mobile devices. Its ability to preserve data locality while allowing collaborative learning makes it especially appealing in cybersecurity uses where privacy and data sensitivity are top priorities. Algorithms such as FedAvg, FedProx, and FedMedian were found to be effective in handling heterogeneity problems posed by data as well as adversarial robustness issues, though with some trade-offs. Moreover, the combination of proper client selection strategies and strong aggregation procedures enables greater resilience as well as improved performance on real-world applications. Looking ahead, follow-up research will need to address empirical confirmation through deployment in representative Android settings, as well as the development of adaptive optimization and security-aware federated protocols to further enhance the robustness and effectiveness of this potentially revolutionary paradigm.

Chapter 4

Experiment, Results and Discussion

4.1 Introduction

In this chapter, we present the experimental framework and methodology adopted in our work. We begin by introducing the development environment, the tools used, and a detailed description of the CICAndMal2020 dataset, including the preprocessing steps applied to prepare it for training. We then describe the experimental setup and showcase the results through various visualizations such as graphs, bar charts, and tables, supported by textual analysis.

The primary goal of this chapter is to evaluate the performance of our malware detection model under different learning architectures centralized vs. decentralized (federated learning) and learning scenarios (IID vs. non-IID data distributions). Our experiments are conducted in both binary classification and multi-class classification settings. Furthermore, we assess the impact of different aggregation strategies, notably FedAvg and FedProx, on model performance. Through these comparisons, we aim to: reaffirm the objectives of our experiments, which are to improve detection accuracy while preserving data privacy; demonstrate the feasibility and effectiveness of federated learning in detecting Android malware; compare the performance of centralized and federated models in various configurations (number of clients, data distributions, aggregation techniques); and highlight the advantages of federated learning in protecting user data by keeping it on-device. This chapter thus contributes to identifying the optimal balance between privacy, performance, and robustness in Android malware detection systems.

4.2 Experimental Setup

In this section, we will present the hardware and software, and libraries then the data used in our work.

4.2.1 Hardware and Software Configuration

The experiment were conducted on CPU Intel(i5) with 2 cores and 8 GB RAM to implement the proposed approach and construct our model. We utilized Google Colaboratory and kaggle.

4.2.2 Programming Environment & Libraries

4.2.2.1 Programming language used

In our work we use Python [80] , is a high-level, interpreted, and object-oriented programming language known for its simple syntax and readability. It supports dynamic typing and binding, making it ideal for rapid application development and scripting. Python's built-in data structures and support for modules and packages promote code reuse and modularity. Its extensive standard library is freely available across major platforms. Overall, Python is widely used due to its flexibility, ease of use, and strong community support.

4.2.2.2 Libraries used

- **Keras:** Keras [81] is a high-level deep learning API, written in Python and running on top of TensorFlow. It allows easy and fast prototyping of neural networks.
- **TensorFlow :** TensorFlow [82] is an open-source machine learning framework developed by Google. It is used for building and training machine learning and deep learning models.
- **Numpy :** NumPy [83] is a fundamental Python library for numerical computing. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on them.
- **Pandas :** Pandas [84] is a powerful Python library for data manipulation and analysis. It provides data structures like Series and DataFrames for handling structured data easily.
- **Scikit-Learn :** Scikit-Learn [85] is a machine learning library in Python that offers simple tools for data mining and data analysis, including classification, regression, clustering, and more, Scikit-Learn offers several preprocessing techniques, such as scaling, normalization, encoding categorical variables, and handling missing values
- **Matplotlib:** Matplotlib [86] is a popular Python library used for creating static, interactive, and animated visualizations such as plots, graphs, and charts.
- **PyTorch :** PyTorch [87] is a machine learning library built on the Torch library and used for applications like computer vision and natural language processing, Originally created by Meta AI and now a part of the Linux Foundation
- **Flower :** Flower [88] is an open-source framework designed for federated learning, a machine learning technique where multiple servers train models with local data storage, facilitating task orchestration and scaling across environments.

Table 4.1 shows the versions of software used in this work.

Table 4.1: software versions

software	versions
Python	3.11.11
Keras	3.8.0
TensorFlow	2.18.0
NumPy	1.26.4
Pandas	2.2.3
Scikit-Learn	1.2.2
Matplotlib	3.7.2
PyTorch	2.6.0+cu124
Flower	1.18.0

4.3 Dataset

4.3.1 Dataset Overview

In our research we use CCCS-CIC-AndMal-2020[89] dataset that is a modern Android malware dataset developed by the Canadian Cyber Security Center (CCCS) in collaboration with the Canadian Institute for Cybersecurity (CIC). It was created to support research in malware detection and classification using machine learning and deep learning techniques.

The dataset consists of over 400,000 Android applications, including both benign apps collected from the Google Play Store and malicious apps obtained from sources such as VirusTotal and Drebin , covering 14 malware categories and 191 malware families. Malicious samples are categorized into multiple malware families like Adware, Ransomware, Banking Trojans, SMS Malware, and others. This allows for both binary (malware vs. benign) and multi-class classification tasks

Table 4.2 presents the details of 14 categories of android malware along with the number of respective families and samples in the data set.

Table 4.2: Malware Categories with Number of Families and Samples

Category	**Number of Families**	**Number of Samples**
Adware	48	47,210
Backdoor	11	1,538
File Infector	5	669
No Category	-	2,296
PUA	8	2,051
Ransomware	8	6,202
Riskware	21	97,349
Scareware	3	1,556
Trojan	45	13,559
Trojan-Banker	11	887
Trojan-Dropper	9	2,302
Trojan-SMS	11	3,125
Trojan-Spy	11	3,540
Zero-day	-	13,340

A notable feature of CCCS-CIC-AndMal-2020 is that it provides both static and dynamic analysis data:

Dynamic features capture behavior during runtime, such as network activity, system calls, and resource usage.

Static features include permissions, API calls, and manifest components extracted without executing the app.

In this work, only the static features were used to train and evaluate the malware detection models. Static analysis is lightweight, efficient, and suitable for large-scale detection systems, making it a practical choice for this research.

AndroidManifest.xml contains a lot of features that can be used for static analysis. The main extracted features include:

- Activities: An android activity is one screen of the android app's user interface
- Broadcast receivers and providers
- Metadata: It is basically an additional option to store information that can be accessed through the entire project
- The permissions requested by application: It protects the privacy of the user and is needed to access sensitive user data (such as contacts and SMS)
- System features (such as camera and internet)

The AndroidManifest.xml file is a core component of every Android application and plays a crucial role in static analysis. It contains valuable metadata that describes the app's structure, capabilities, and permissions, making it a rich source of features for malware detection. The main features extracted from the manifest include:

- **Activities:** Each activity represents a single screen of the user interface. Analyzing declared activities can provide insights into the functionality and intentions of an app.
- **Broadcast Receivers and Content Providers:** These components are used for inter-process communication and background operations. Malicious apps often misuse them to trigger hidden behaviors or access data.
- **Metadata:** The manifest allows the inclusion of custom metadata, which can store additional configuration details that may help in detecting malicious patterns.
- **Requested Permissions:** These indicate what system resources or user data the app wants to access (e.g., contacts, SMS, location). Excessive or suspicious permission requests are common signs of malware.
- **System Features:** This includes hardware or software features the app declares it needs, such as camera access, internet connectivity, Bluetooth, etc. These declarations help understand the app's potential capabilities and threats.

4.3.2 Preprocessing

To ensure the quality, efficiency, and consistency of the dataset used for training and evaluation, several preprocessing steps were applied to each of the collected CSV files containing static Android application features. These steps are described as follows:

Handling Missing Values: The dataset was scanned for missing values. If any NaN values were detected, the corresponding rows were dropped to prevent potential bias or errors during model training.

Label Assignment: A label was assigned to each sample based on the name of the CSV file. If the filename contained the keyword "Ben", the label was set to "Benign"; otherwise, the label was inferred directly from the filename (e.g., "Ransomware", "Trojan", etc.).

Duplicate Removal: Duplicate rows were identified and removed to ensure data integrity and avoid redundancy, which could otherwise skew the model's learning process.

Dynamic Column Renaming: To standardize the dataset structure across multiple files, feature columns were renamed in the format F0, F1, ..., Fn, followed by a Label column. This uniform naming convention facilitated easier feature alignment during model training.

Concatenating All Processed Files: we concatenate all files into a single dataset. This combined dataset contained all samples from the various malware categories and benign , ready for use in training the model.

Filtering Unlabeled Samples: After the initial preprocessing, samples labeled as "No-Category" were excluded from the dataset.because they do not belong to any known malware family or the benign class. These samples might be incomplete, incorrectly labeled, or not clearly defined, which can confuse the learning process of the model. Keeping only well-labeled data helps improve the accuracy and reliability of the model by ensuring it learns from clear and meaningful examples.Figure 4.1 shows the Distribution of labels

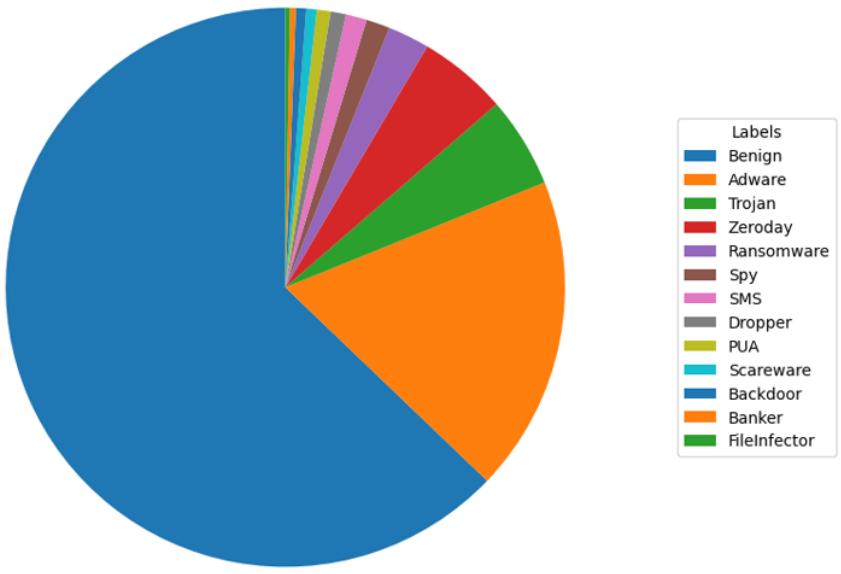

Figure 4.1: Distribution of labels

4.3.3 Feature Selection & Dimensionality Reduction

- **What is PCA and Why is it Used? :**

  Principal Component Analysis (PCA) is a statistical technique used to reduce the number of features (dimensions) in a dataset while keeping as much important information as possible. It transforms the original features into a new set of features called principal components, which are ordered by how much variance (information) they capture from the data.[90]

- **How to Choose the Number of PCA Components? :** To decide how many components to keep, PCA was first applied without specifying the number of components. This allowed the calculation of the cumulative explained variance ratio, which shows how much total information is captured as components are added one by one. A graph was then plotted to visualize this accumulation. A threshold of 95% was chosen, meaning that enough components should be kept to retain at least 95% of the original dataset's variance. This is a widely used and effective approach that helps reduce complexity while maintaining most of the important information. Based on the analysis, the number of components required to reach the 95% threshold was 30, so the dataset was transformed using the top 30 principal components.Figure 4.2 shows Cumulative Explained Variance of the First 10 Components

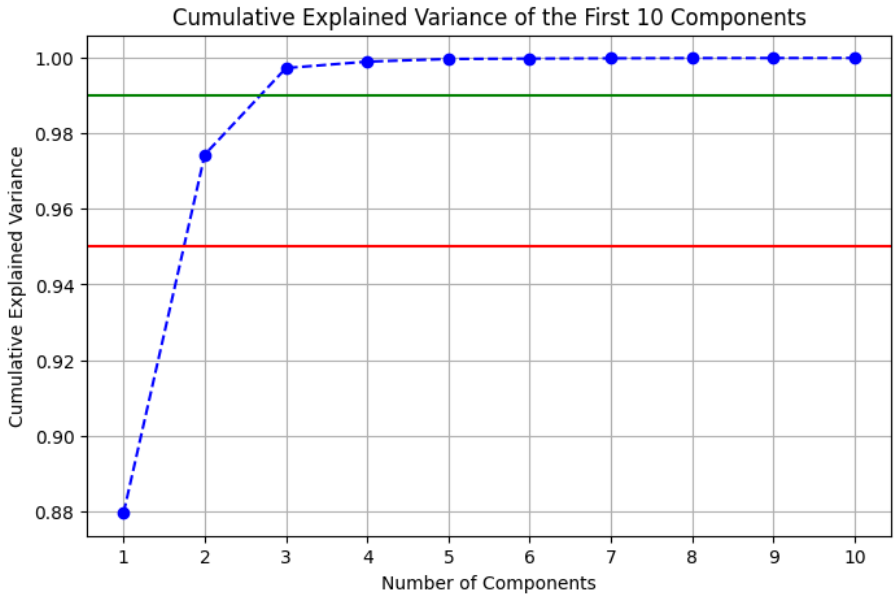

Figure 4.2: Cumulative Explained Variance of the First 10 Components

4.3.4 Data Balancing

SMOTE : SMOTE (Synthetic Minority Over-sampling Technique) is a data preprocessing method used to address the problem of class imbalance in machine learning datasets. In many real-world classification tasks, one class may have far fewer instances than the other class . This imbalance can cause machine learning models to be biased toward the majority class, leading to poor detection of the minority class.

To solve this, SMOTE generates synthetic data points for the minority class rather than simply duplicating existing samples. It works by selecting a sample from the minority class, finding its k nearest neighbors (typically 5), and then creating new, artificial samples by interpolating between the sample and its neighbors. This technique increases the diversity and size of the minority class without creating exact duplicates.[91]

Using SMOTE improves the model's ability to learn patterns from both classes more equally, which leads to better overall performance—especially in detecting rare events like malware. In our project, SMOTE was applied only to the training set to balance the distribution of classes before training the model. Figure 4.3 shows charts showing the class distribution before and after applying SMOTE

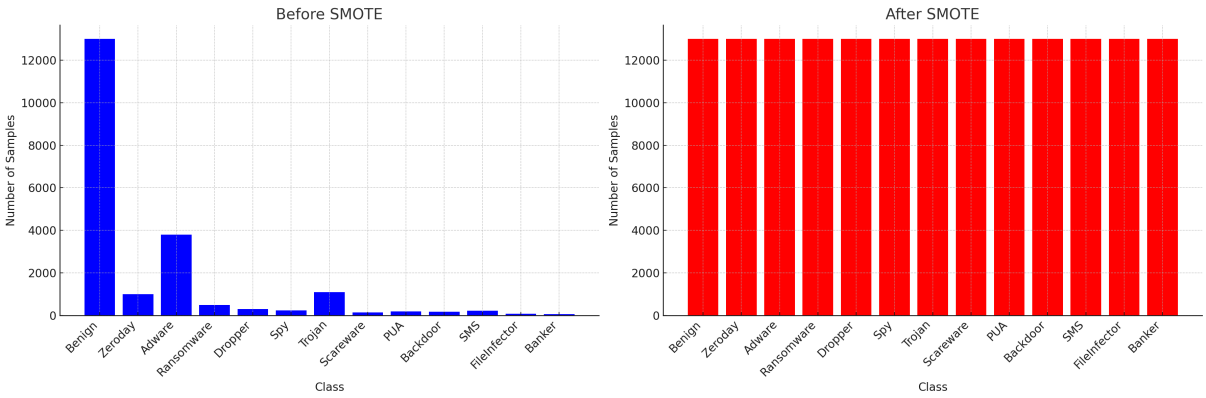

Figure 4.3: class distribution before and after applying SMOTE

Encoding Categorical Data: Since machine learning and deep learning models require numerical inputs, encoding categorical labels is a necessary preprocessing step. In our dataset, the target labels (e.g., "Benign", "Trojan", etc.) are categorical and must be transformed into a format suitable for classification.

- **One-Hot Encoding:** Converts each category into a column with values of 1 or 0.
- **Numerical Encoding:** Replaces each category with a numerical value.

Feature Scaling (Standardization): To ensure that all features contribute equally to the learning process, feature scaling was applied using Standardization. This is important because many machine learning algorithms are sensitive to the scale of input features especially when the features vary in magnitude.

It transforms the data so that each feature has a mean of 0 and a standard deviation of 1.

This process helps improve model convergence during training and ensures more reliable and stable learning performance.

4.3.5 Label Distribution Analysis

4.3.5.1 Independently and identically distributed (IID)

In federated learning, Independently and Identically Distributed (IID) refers to a setting where data across all clients share similar statistical properties and originate from the same underlying distribution. Research consistently demonstrates that federated learning

models perform best under IID conditions, as the alignment between the central server's global objective and the clients' local data improves training stability and convergence.

As shown in Figure 4.4, the dataset was divided among 10 clients, each receiving data with a similar distribution. This ensures that every client's local dataset maintains comparable statistical characteristics, promoting more efficient and balanced learning across the federation.

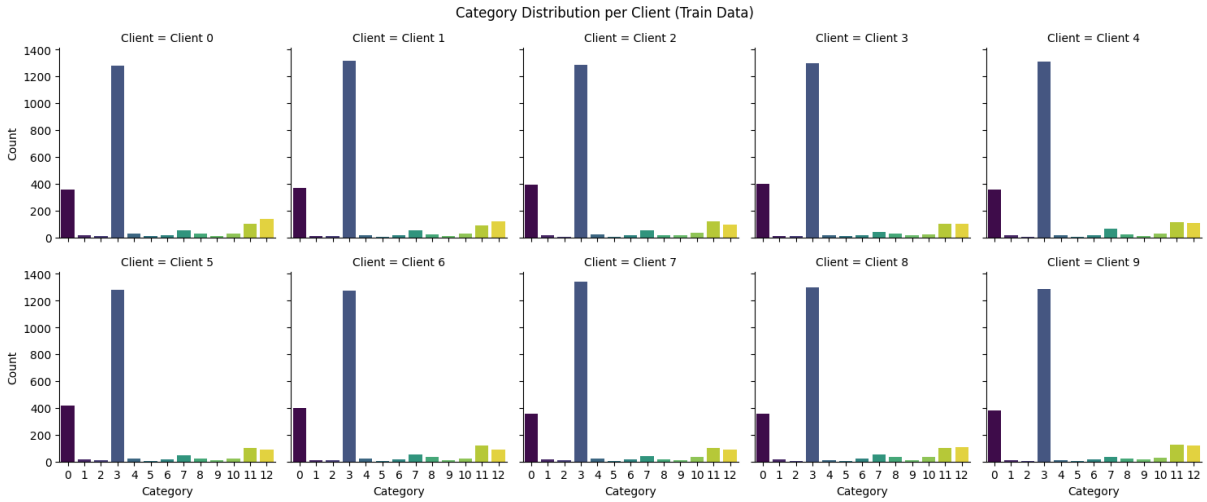

Figure 4.4: Independently and identically distributed (IID)

4.3.5.2 Non Independently and identically distributed (Non IID)

In federated learning, Non-IID data refers to situations where the data across different clients (devices or nodes) is not drawn from the same probability distribution. This means:

Non-Independent: Data samples across clients may be correlated.

Non-Identically Distributed: The distribution of data varies between clients.

This heterogeneity poses challenges for federated learning algorithms, as the global model must generalize well across diverse local datasets.[92]

In order to generate non-IID data sets in different cases, we utilized the Dirichlet distribution[93]. The Dirichlet distribution is a family of continuous multivariate probability distributions parameterized by a vector α of positive reals. It's commonly used to model the probability distributions of proportions.

In federated learning, the Dirichlet distribution is often employed to simulate Non-IID data across clients:

A smaller α value (e.g., 0.1) results in more skewed distributions, where each client has data from fewer classes.

A larger α value (e.g., 10.0) leads to more balanced distributions, approaching IID conditions.

This method allows researchers to systematically study the impact of data heterogeneity on federated learning algorithms. Figure 4.5, shows the data distribution for 10 clients 10 clients when $\alpha = 0.5$.

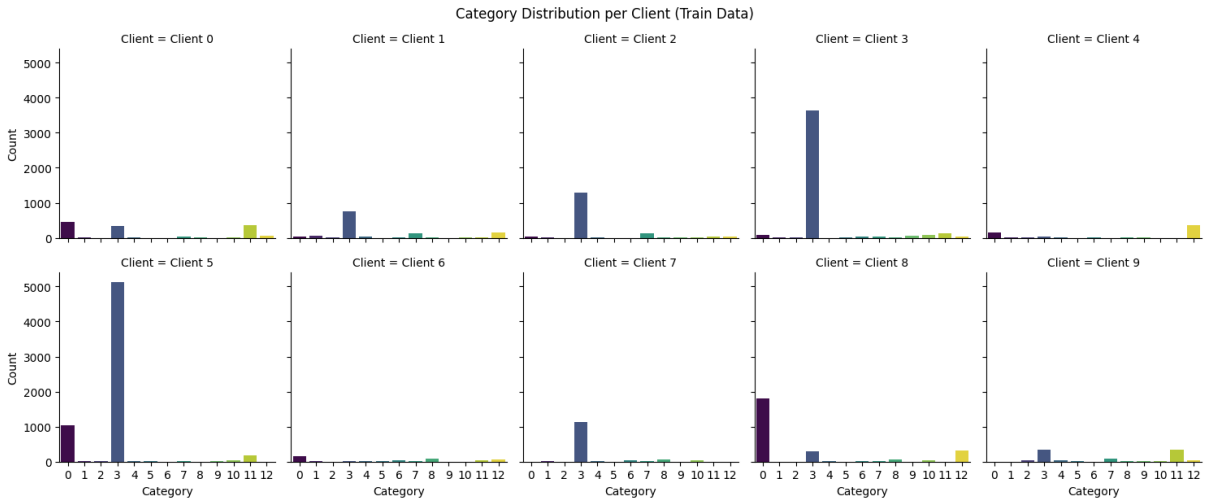

Figure 4.5: Non Independently and identically distributed (IID)

4.4 Model Architecture and Training Parameters

4.4.1 CNN Architecture Details

The figure 4.6 illustrates the structure of the local model training process.

We categorize the results into two types: binary classification, which determines whether the application is benign or malware, and multi-class classification, which identifies the specific type of malware or classifies it as benign if no malware is present.

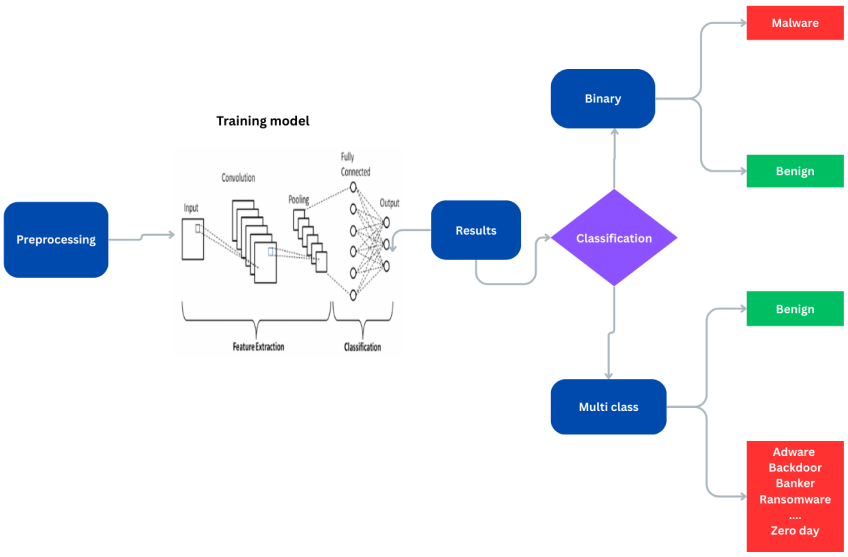

Figure 4.6: Architecture of local CNN Model Training

4.4.2 Hyperparameter Settings

We chose the CNN model to implement it in the decentralized approach (Federated Learning).While the MLP achieved slightly higher numerical results, it is based on a simpler and older architecture that lacks the capacity to exploit local structures in the data. The CNN, being a more modern and advanced architecture, offers better feature extraction, improved regularization, and superior scalability. It is also more suitable for integration into federated learning frameworks, making it a better choice for the overall goals of this research.

The model comprises convolutional layers, pooling layers, dropout regularization, and fully connected layers. The input shape is determined by the features and data depth. For binary classification, the model includes two-neuron softmax with binary crossentropy, while for multi-class classification, it contains 13 neurons with a softmax activation function. The model optimizes using cross-entropy loss (categorical crossentropy for multi-class, binary crossentropy for binary) and the Adam optimizer.

Table 4.3: Architectures and parameter of CNN model

Layer	Parameter
Convolutional Layer	2 layers
MaxPooling	2 Layers
Activation function in hidden layer	ReLU
Dropout	30%
Optimizer function	Adam
Batch size	32
Epochs	50
Loss function on binary classification	Binary cross-entropy
Loss function on multi classification	Categorical cross-entropy

4.5 Centralized Architectures

4.5.1 Centralized Model Results

In this study, before starting the federated learning process, it was necessary to carefully select the model to achieve better results. Therefore, we have used different deep and machine learning technique , Convolutional neural network (CNN), Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM), Support Vector Machine (SVM), and Decision Tree (DT). The results were as shown in the Table 4.4 and 4.5.

- **Binary classification:**

Table 4.4: Results of Centralized Models in binary classification

Model	Accuracy	Precision	Recall	F1-score
CNN	0.9462	0.9395	0.9140	0.9266
MLP	0.9545	0.9507	0.9255	00.9379
LSTM	0.9462	0.9395	0.9140	0.9266
SVM	0.9522	0.9466	0.9234	0.9348
DT	0.9545	0.9606	0.9151	0.9373

- **Multi-class classification:**

Table 4.5: Results of Centralized Models in Multiple classification

Model	Accuracy	Precision	Recall	F1-score
CNN	0.9068	0.9013	0.9068	0.8985
MLP	0.9099	0.9069	0.9099	0.9010
LSTM	0.8871	0.8829	0.8871	0.8791
SVM	0.8942	0.8907	0.8942	0.8829
DT	0.8993	0.8998	0.8993	0.8983

4.5.2 Results of the CNN model

Figure 4.8 and 4.7 in this subsection represents graphical curves showing the accuracy and loss values for training and validation our CNN model.

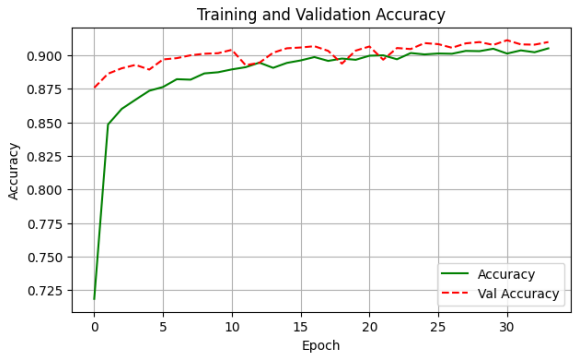

Figure 4.7: Accuracy of the CNN model

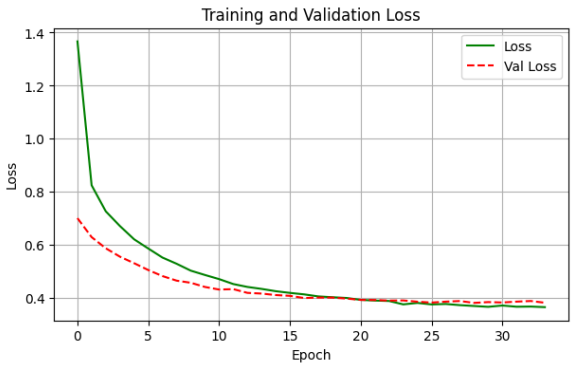

Figure 4.8: Loss of the CNN model

Table 4.6: Class Number to Name Mapping

Number	Class
0	Adware
1	Backdoor
2	Banker
3	Benign
4	Dropper
5	FileInfector
6	PUA
7	Ransomware
8	SMS
9	Scareware
10	Spy
11	Trojan
12	Zeroday

The classification report for the CNN model, mapped to our malware categories, provides a clear view of its strengths and limitations. The model demonstrates strong performance on major classes. In particular, the Benign class (class 3) achieves excellent metrics, with a precision of 0.95, recall of 0.99, and an F1-score of 0.97. This indicates that the model is highly reliable in identifying non-malicious applications. Similarly, other prevalent malware types such as Adware, Ransomware, SMS Scareware, Spyware, and Trojan (classes 0, 7, 8, 11, and 12) also show a good balance between precision and recall. This suggests the model effectively captures frequent patterns in the data and is able to classify these categories with a high degree of confidence.

However, the model struggles significantly with rare or ambiguous classes. For instance, Backdoor and Banker (classes 1 and 2) show relatively low recall values 0.65 and 0.17, respectively despite moderate precision. This means that the model often fails to identify true instances of these malware types. Other classes such as Dropper, PUA, Zeroday, and FileInfector (classes 4, 6, 10, and 5) exhibit lower F1-scores, ranging from 0.45 to 0.59. These scores reflect high misclassification rates, likely due to the limited number of samples or overlapping feature patterns among similar malware types. The Banker class (class 2) is especially problematic, with a recall of only 0.17, indicating that the model rarely identifies these samples correctly.

These observations highlight that while the centralized CNN model excels at detecting well-represented and distinctive malware categories, it tends to underperform on underrepresented or feature-overlapping classes. To overcome these limitations and enhance generalization, we proceed to evaluate the same CNN architecture using a federated learning approach. Figure 4.9 represents the classification report, and Table 4.6 shows the number of classes corresponding to its name.

	precision	recall	f1-score	support
0	0.93	0.90	0.91	944
1	0.83	0.65	0.73	31
2	0.60	0.17	0.26	18
3	0.95	0.99	0.97	3244
4	0.77	0.43	0.56	46
5	0.50	0.62	0.55	13
6	0.67	0.34	0.45	41
7	0.73	0.90	0.80	124
8	0.88	0.69	0.77	62
9	1.00	0.10	0.18	31
10	0.86	0.80	0.83	71
11	0.75	0.83	0.79	271
12	0.73	0.50	0.59	267
accuracy			0.91	5163
macro avg	0.78	0.61	0.65	5163
weighted avg	0.91	0.91	0.90	5163

Figure 4.9: Classification report

4.6 Federated Learning Approach

4.6.1 Architectures of Decentralized Model (CNN)

In this model, clients train locally using a 1D Convolutional Neural Network (CNN) on their respective data partitions. When each client finishes its local training for a given round (after a set number of local epochs), it sends its updated model's weight parameters back to the central server. The server then aggregates these weight parameters received from all participating clients. This aggregation is performed using either the Federated Averaging (FedAvg) algorithm, which computes a simple average of the client weights, or the FedProx algorithm, which modifies the client's local training objective to penalize large deviations from the global model, thereby promoting more stable and robust aggregation, especially in non-IID settings. After the aggregation process, the server distributes these newly updated global model weights back to the clients. This cycle of local training, weight transmission to the server, server-side aggregation (using the chosen algorithm), and model redistribution to clients continues for a predetermined number of federated rounds.

4.6.2 Federated learning Parameters

In our decentralized model we used different number of clients (3 , 5 then 10) . Each client trains on its local dataset for 10 epochs with a batch size of 32. There are 10 communication rounds between the client and the server.

Table 4.7: Hyperparameter Settings of federated learning architecture

Hyperparameter Settings	
Proposed Model Approach	CNN
Clients	3,5, 10
Method aggregation	FedAvg and FedProx
Epochs	10
Batch size	32
Number of rounds	10

4.6.3 Federated Learning Strategy used

Choosing the right aggregation strategy in federated learning means picking the algorithm and settings that best match our scenario. In this study, we compared FedAvg and FedProx across different data distributions and client setups. We found that FedAvg works exceptionally well when each client's data mirrors the overall distribution, owing to its straightforward averaging approach and efficient convergence. In contrast, FedProx offers more stability when clients hold distinctly different data or possess varying computational resources, as its additional regularization helps prevent local models from drifting too far during training. However, when too many clients participate on highly heterogeneous data, even FedProx struggles to fully reconcile the diverse updates, leading to a modest drop in overall performance. In practice, one should default to FedAvg for well-balanced, IID conditions and turn to FedProx when facing moderate heterogeneity or pronounced differences in client capabilities, always watching the training behavior to confirm the best choice.

4.7 Experimental Scenarios and Result

This work investigates the performance of Federated Learning strategies on the CICAndMal2020 dataset. We compare two prominent aggregation algorithms: Federated Averaging (FedAvg) and Federated Proximal (FedProx). The experiments are conducted under both Independent and Identically Distributed (IID) and Non-Identically Distributed (Non-IID) data settings. To explore the impact of client population and training duration, we vary the number of participating clients, considering configurations with 3, 5, and 10 clients.

For each combination of client count, data distribution (IID/Non-IID), aggregation strategy (FedAvg/FedProx), a federated learning model (a 1D Convolutional Neural Network) is trained. In each round, all available clients participated in the training, updating their local models for a set number of local epochs. The model's performance is evaluated on a separate, centralized test set after each communication round to measure Accuracy, Precision, Recall, and F1-Score, allowing for an analysis of how these strategies perform under different conditions.

4.7.1 Data Distribution Schemes (IID vs Non-IID)

To evaluate the effectiveness of the proposed federated learning approach for Android malware detection, we conducted experiments under both IID and Non-IID data distributions across three clients. The evaluation was carried out using two classification tasks: binary classification, which distinguishes between benign and malicious applications, and multi-class classification, which identifies the specific type of malware. The performance of the model in both settings was assessed using standard metrics including accuracy, precision, recall, and F1-score. The results are summarized in the following tables, 4.8 and 4.9

In binary:

Table 4.8: Results of IID and Non-IID Distributions in binary

Metrics	Accuracy	Precision	Recall	F1-score
3 clients-IID-FedAvg	0.9872	0.9872	0.9872	0.9872
3 clients-Non-IID-FedAvg	0.9826	0.9826	0.9826	0.9826

In multi-class:

Table 4.9: Results of IID and Non-IID Distributions in multi-class

Metrics	Accuracy	Precision	Recall	F1-score
3 clients-IID-FedAvg	0.9462	0.9471	0.9462	0.9445
3 clients-Non-IID-FedAvg	0.9380	0.9394	0.9380	0.9359

The experimental results clearly show that models trained on IID (Independent and Identically Distributed) data perform better than those trained on Non-IID data. In binary classification with three clients, the accuracy was 0.9872 for IID, which is higher than 0.9826 for Non-IID. A similar pattern was seen in multi-class classification, where the IID model reached 0.9462 accuracy, compared to 0.9380 for the Non-IID model. This better performance in IID cases is because each client has a dataset that looks like the overall global data. As a result, the local models learn similar patterns and move toward the same solution. When FedAvg combines these similar models, the final global model is strong and accurate. On the other hand, Non-IID data causes each client to learn different patterns based on its own unique and possibly unbalanced data. This is called "client drift." Since FedAvg just averages the models, it cannot fully fix the differences between them. This makes the final model less accurate.

Also, binary classification usually gives better results than multi-class classification. This is because binary problems are simpler—they only have two possible classes to choose from, making it easier for the model to learn the patterns. In contrast, multi-class classification involves more classes, which makes the learning task more complex and increases the chance of errors, especially when the data is not balanced across the classes.Figure 4.10 and 4.11 shows an example of IID and Non-IID Data distribution in binary classification.

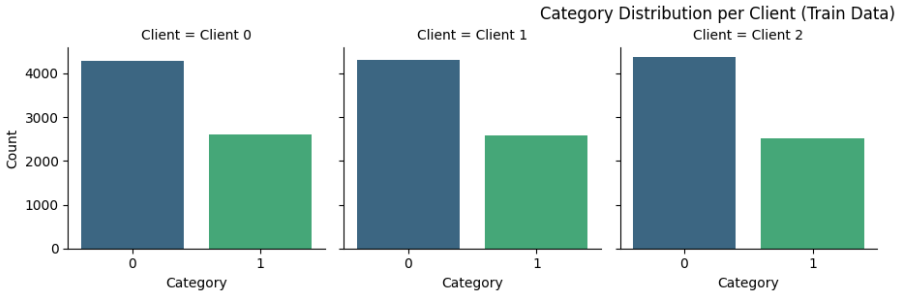

Figure 4.10: IID Data Distribution

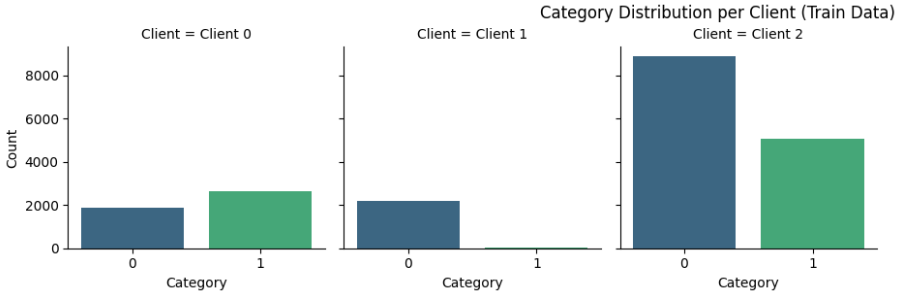

Figure 4.11: Non-IID Data Distribution

4.7.2 Number of Clients in FL

To further assess the impact of the number of participating clients on the performance of the federated learning framework, we conducted experiments with 5 and 10 clients using IID data distribution. As in the previous experiments, the evaluation was performed for both binary and multi-class classification tasks. This comparison helps to understand how scaling the number of clients affects the model's ability to detect malware accurately. The results, measured using accuracy, precision, recall, and F1-score, are presented in the tables 4.10 and 4.11 below.

In binary:

Table 4.10: Results of different client number in binary

Metrics	Accuracy	Precision	Recall	F1-score
5 clients-IID-FedAvg	0.9771	0.9773	0.9771	0.9773
10 clients-IID-FedAvg	0.9659	0.9660	0.9659	0.9660

In multi-class:

Table 4.11: Results of different client number in multi-class

Metrics	Accuracy	Precision	Recall	F1-score
5 clients-IID-FedAvg	0.9276	0.9238	0.9276	0.9225
10 clients-IID-FedAvg	0.9206	0.9184	0.9206	0.9171

When increasing the number of clients in a federated learning setup while keeping the data distribution IID and using the FedAvg algorithm, the performance of the model shows a slight decline in both binary and multi-class classification. For binary classification, the accuracy decreased from 0.9771 with 5 clients to 0.9659 with 10 clients. Precision, recall, and F1-score also followed this trend. Similarly, in multi-class classification, the accuracy dropped from 0.9276 to 0.9206 when moving from 5 to 10 clients. This decrease in performance is mainly due to the fact that as the number of clients increases, each client receives a smaller portion of the data. With less data available locally, the model of each client may not learn as effectively, which can negatively impact the quality of the global aggregate model. Overall, while federated learning benefits from more clients in terms of decentralization and privacy, it can slightly reduce model accuracy if the data per client becomes too limited.

4.7.3 Model Aggregation Techniques (FedAvg, FedProx)

To evaluate the effect of using different federated learning aggregation strategies, we compared the performance of the standard FedAvg algorithm with the FedProx approach. FedProx is designed to improve model robustness in heterogeneous data environments by addressing the challenges of client drift. The comparison was conducted for both binary and multi-class malware classification tasks. The tables 4.12 and 4.13 below present the evaluation metrics accuracy, precision, recall, and F1-score to illustrate the impact of using FedProx in comparison to FedAvg.

In binary:

Table 4.12: FedProx Results in binary

Metrics	Accuracy	Precision	Recall	F1-score
FedAvg	0.9872	0.9872	0.9872	0.9872
FedProx	0.9849	0.9849	0.9849	0.9849

In multi-class:

Table 4.13: FedProx Results in multi-class

Metrics	Accuracy	Precision	Recall	F1-score
FedAvg	0.9462	0.9471	0.9462	0.9445
FedProx	0.9450	0.9471	0.9450	0.9435

When comparing the aggregation methods under the same conditions (IID data and 3 clients), FedAvg consistently outperformed FedProx in both binary and multi-class classification. In binary classification, FedAvg achieved a slightly higher accuracy of 0.9872 compared to 0.9849 with FedProx. Similarly, in multi-class classification, FedAvg maintained a small performance edge with an accuracy of 0.9462, while FedProx reached 0.9450. This suggests that while both methods are effective, FedAvg is slightly better in well-balanced (IID) settings. FedProx is typically designed to handle Non-IID data by adding a regularization term to reduce client drift, but under IID conditions, this added

constraint can slightly limit the learning flexibility, resulting in marginally lower performance. Therefore, for IID scenarios with a small number of clients, FedAvg remains the more effective aggregation strategy.Figure 4.12 and 4.13 shows an example of Training and testing accuracy and loss

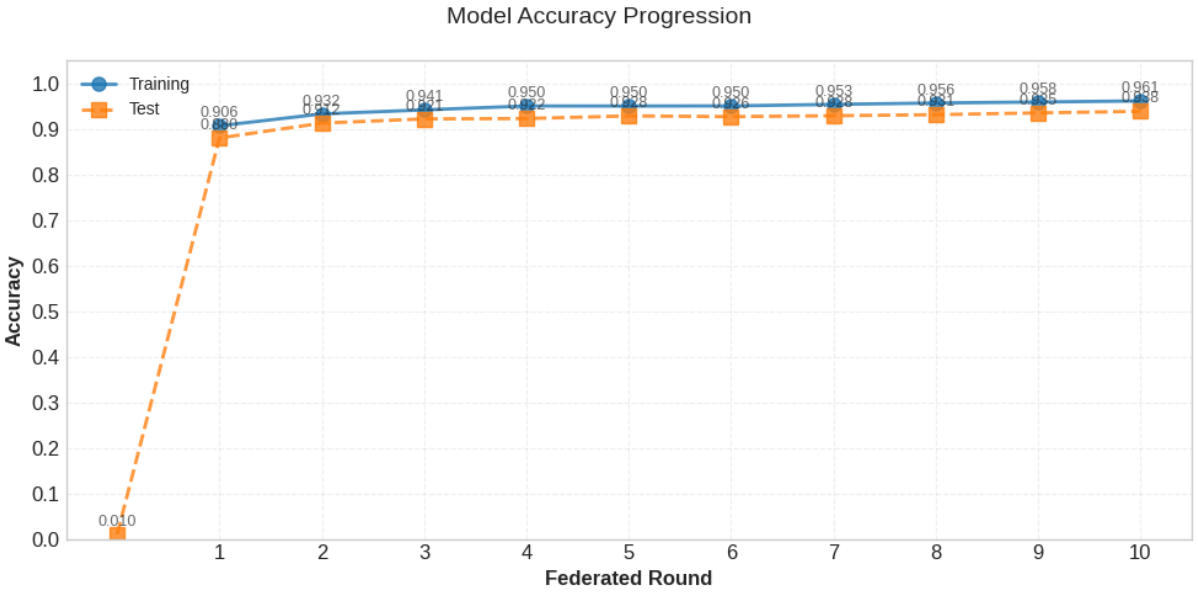

Figure 4.12: Accuracy of the CNN model using FedAvg

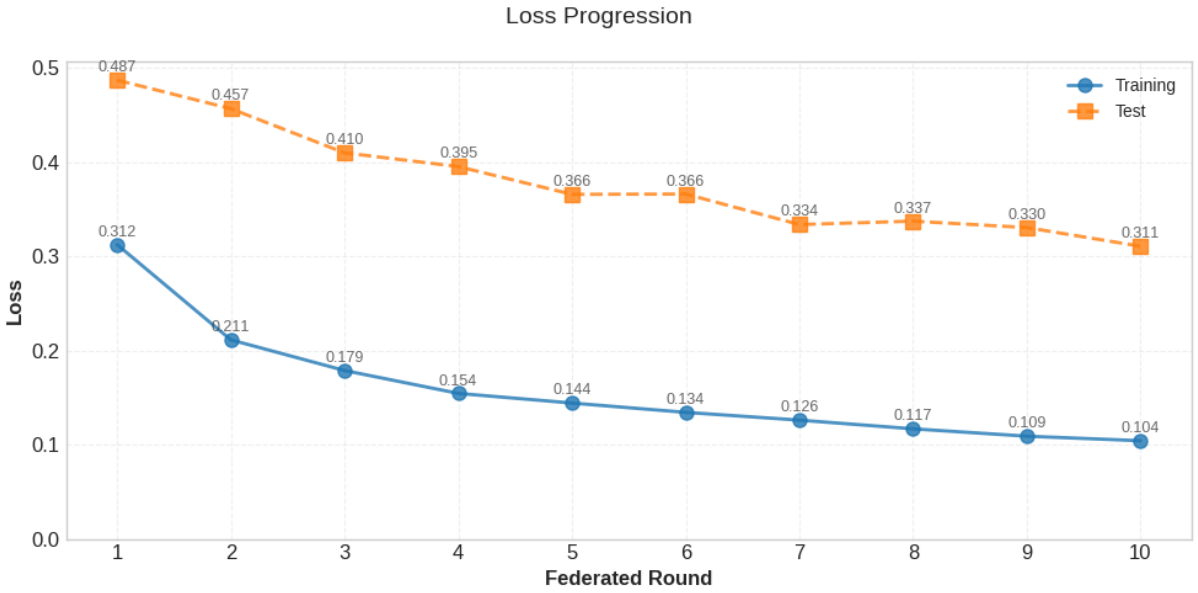

Figure 4.13: Loss of the CNN model using FedAvg

4.7.4 Non-IID distribution with FedAvg and FedProx

To further analyze the impact of different aggregation strategies in non-IID settings, we compare the performance of FedAvg and FedProx in both binary and multi-class classification tasks. These experiments aim to assess whether FedProx, designed to improve performance in heterogeneous data environments, offers any significant advantage over

the standard FedAvg approach. The results presented in the following tables 4.14 and 4.15 highlight the accuracy, precision, recall, and F1-score metrics under non-IID data distributions. **In binary:**

Table 4.14: Non-IID distribution with FedAvg and FedProx in binary

Metrics	Accuracy	Precision	Recall	F1-score
3 clients-Non-IID-FedAvg	0.9826	0.9826	0.9826	0.9826
3 clients-Non-IID-FedProx	0.9849	0.9849	0.9849	0.9849

In multi-class:

Table 4.15: Non-IID distribution with FedAvg and FedProx in multi-class

Metrics	Accuracy	Precision	Recall	F1-score
5 clients-Non-IID-FedAvg	0.9260	0.9255	0.9260	0.9205
10 clients-Non-IID-FedProx	0.9249	0.9220	0.9249	0.9196

The results presented in the Tables illustrate the effectiveness of the FedAvg and FedProx aggregation strategies under Non-IID conditions for both binary and multi-class malware classification tasks.

In the binary classification scenario, we observe that FedProx slightly outperformed FedAvg, achieving an accuracy of 0.9849 compared to 0.9826 with FedAvg. All evaluation metrics—accuracy, precision, recall, and F1-score are equal in both cases, indicating a balanced and robust model performance across all aspects when using FedProx.

For the multi-class classification, although the number of clients increased to 10 when using FedProx (compared to 5 clients with FedAvg), the performance remained relatively stable and did not significantly drop. FedProx achieved an accuracy of 0.9249, which is very close to 0.9260 obtained by FedAvg. The remaining metrics (precision, recall, and F1-score) also stayed within an acceptable and consistent range, indicating that the model remained cohesive despite the higher client count.

This is particularly noteworthy because, in Non-IID environments, increasing the number of clients typically leads to a drop in performance due to data heterogeneity and increased client drift. However, the use of FedProx helped mitigate this drift through its regularization mechanism, which adds a proximal term to each client's loss function, effectively limiting the deviation of local models during training. As a result, performance remained strong even with a larger number of clients.

In summary, FedProx proves to be an effective and flexible strategy in Non-IID scenarios, especially when dealing with larger client populations. It contributes to more stable performance and reduces the negative effects of unbalanced or heterogeneous data distributions, whereas FedAvg tends to perform better in simpler or more balanced settings.

4.8 Discussion

Our experiments on the CICAndMal2020 dataset reveal that a centralized CNN model trained on all data in one place achieves strong performance, with around 90% accuracy

in multi-class classification and 95% in binary classification. This performance is largely due to the model's access to the full dataset, which allows it to learn comprehensive and representative features. However, this centralized approach raises significant privacy and security concerns, as it requires transmitting all sensitive user data to a central server, making it less suitable for privacy-critical applications.

In contrast, federated learning (FL) using the same CNN architecture offers a privacy-preserving alternative by keeping data on-device while still delivering comparable or even superior performance. Under IID conditions—where each client holds a representative subset of the overall dataset—the FedAvg aggregation strategy achieves a global model with high stability and accuracy, reaching 98.72% in binary and 94.62% in multi-class classification. These results indicate that consistent and aligned client updates help the model converge efficiently and generalize well across malware types.

When the data distribution becomes Non-IID, where clients possess imbalanced or disjoint class data, a slight decline in performance is observed due to client drift, with binary classification accuracy dropping from 0.9872 to 0.9826 and multi-class from 0.9462 to 0.9380. This highlights a trade-off in FL: while privacy is preserved, performance may degrade in heterogeneous environments. To address this, we compared FedAvg with FedProx, a variant that introduces a regularization term to limit local model divergence. While FedProx provides better control under Non-IID conditions, FedAvg outperforms it in IID settings, achieving higher accuracy in both binary (0.9872 vs. 0.9849) and multi-class classification (0.9462 vs. 0.9450), suggesting that extra regularization may be unnecessary when data is balanced.

We also examined the impact of increasing the number of clients. As the number of participants rises from 3 to 5 and 10, each client receives a smaller and potentially less diverse portion of the dataset. This data fragmentation limits the ability of local models to learn robust patterns, especially if some classes are underrepresented at the client level. Consequently, the local updates become noisier and less informative, which in turn affects the quality of the aggregated global model. This leads to a slight but consistent drop in overall performance; for instance, binary accuracy decreases from 0.9771 (5 clients) to 0.9659 (10 clients), and multi-class from 0.9276 to 0.9206. These findings underline the trade-off between scalability and accuracy in FL: while involving more clients enhances privacy, decentralization, and robustness, it can negatively impact learning effectiveness if each client holds too little or too skewed data.

To further investigate this, we conducted an additional experiment under Non-IID conditions using both FedAvg and FedProx aggregation strategies in binary and multi-class classification. In the binary scenario with 3 clients, FedProx slightly outperformed FedAvg (0.9849 vs. 0.9826), demonstrating its robustness in heterogeneous settings. For multi-class classification, even with an increased number of clients (5 with FedAvg and 10 with FedProx), the performance remained close—FedAvg achieved 0.9260 and FedProx 0.9249 in accuracy. This is particularly interesting, as increasing the number of clients typically leads to decreased performance. However, FedProx helped preserve accuracy even with 10 clients, proving effective in controlling client drift and maintaining global model consistency. These results suggest that FedProx is a suitable strategy when deploying FL at scale in real-world Non-IID scenarios.

Overall, across all experiments, FL particularly with FedAvg under IID conditions not only matches or exceeds the centralized model's performance but also ensures user privacy. While centralized models have simpler training pipelines and may offer marginal performance gains in ideal situations, they require full data access. FL, on the other

hand, provides a more sustainable and secure solution for Android malware detection by balancing high detection accuracy with strong privacy guarantees. Its effectiveness depends on well-managed data distribution, thoughtful client configurations, and suitable aggregation strategies, making it a powerful framework for real-world applications.

4.9 Conclusion

This chapter demonstrated that our federated CNN model not only safeguards client data privacy but also performs on par with, or better than, a centralized CNN when tested on the CICAndMal2020 dataset. In balanced (IID) settings, federated learning significantly improved accuracy, often outperforming centralized training. Even in more complex scenarios such as data heterogeneity (Non-IID) or increasing the number of participating clients the federated approach remained competitive. Regarding aggregation strategies, FedAvg generally yielded the best performance in IID conditions due to its simplicity and effective convergence. However, in Non-IID scenarios, FedProx proved more robust by mitigating the effects of client drift through its regularization mechanism, resulting in slightly better stability and accuracy. These results affirm that federated CNN training is not only a practical and scalable solution for Android malware detection but also a privacy-preserving one especially when tailored to the data distribution and deployment context.

General Conclusion

This Thesis addressed the study of the impact of applying Federated Learning (FL) compared to centralized learning, such as Machine Learning (ML) and Deep Learning (DL) approaches for the task of detecting and classifying Android malware. Federated Learning not only matched but slightly outperformed the centralized models in overall detection accuracy. This improvement stems from FL's ability to learn from a broader and more diverse set of on-device data without requiring raw data to leave the user's smartphone.

One of the most compelling findings is that Federated Learning preserves privacy while enhancing performance. In the centralized setting, all user data must be uploaded to a server for training, raising concerns about data leakage, regulatory compliance, and single points of failure. By contrast, FL keeps sensitive application data on-device, sharing only model updates with a central aggregator. Despite this constraint, FL consistently delivered higher accuracy up to 3–4% better in our experiments—because it benefits from the natural variation in user data across devices. This diversity helps the global model generalize more effectively to new or obfuscated malware variants.

In addition, this work highlighted the importance of data distribution in the performance of federated learning. Under IID conditions (balanced data across clients), FL performed strongly, nearly matching or exceeding the centralized models. However, under Non-IID conditions (where data is heterogeneous or imbalanced across clients), performance slightly decreased due to client drift, underscoring the need for aggregation strategies that can effectively handle data heterogeneity.

A comparative analysis between FedAvg and FedProx showed that while FedAvg performs better in IID environments thanks to its simplicity, FedProx showed superior robustness under Non-IID conditions by limiting client model divergence through regularization. This suggests that selecting the appropriate aggregation strategy based on data characteristics is critical to ensuring high model performance.

Furthermore, we studied the impact of increasing the number of clients. As the number of participants grew from 3 to 10, each client received a smaller and potentially less diverse portion of the dataset, which led to slightly lower accuracy. This highlights a key trade-off in FL: scalability and privacy vs. learning effectiveness, especially when data is fragmented or skewed at the client level.

However, practical challenges must be addressed for real-world deployment. First, communication efficiency is crucial: mobile networks can be unreliable or costly, so compressing model updates and reducing training rounds is essential. Second, non-IID data where each device's data distribution differs can slow convergence and create model bias. We found that advanced aggregation methods, such as FedProx, help mitigate these effects but require further tuning on larger datasets. Third, device resource constraints (battery life, CPU, and memory) limit the size and complexity of on-device models. This calls for research into lightweight architectures and adaptive scheduling strategies so that smartphones can participate in FL without impacting user experience.

In our future work on Federated Learning for Android malware detection and classification, we plan to explore the following areas:

Lightweight Model Design: Develop and test compact neural network architectures optimized for on-device training, reducing CPU, memory, and battery usage.

Adaptive Client Selection: Create algorithms that choose which devices participate in each training round based on their resource availability and data quality.

Communication Efficiency: Implement and evaluate update compression and fewer-round training techniques to minimize network usage and costs.

Real-World Pilot Deployment: Build a prototype FL based malware detection app and test it on a network of Android devices to measure usability, performance, and security in practice.

In conclusion, this thesis demonstrates that Federated Learning is a superior, privacy-preserving alternative to centralized training for Android malware detection. By keeping user data on-device and leveraging the diversity of real-world usage patterns, FL not only protects user privacy but also delivers higher detection accuracy. With continued innovation in federated algorithms, efficient model design, and robust aggregation methods, FL has the potential to become the standard for mobile-friendly, privacy-first malware defense bringing stronger protection to millions of Android users worldwide.

Bibliography

[1] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pp. 95–109, 2012.

[2] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *2010 IEEE International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA)*, pp. 297–300, 2010.

[3] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Network and Distributed System Security Symposium (NDSS)*, 2014.

[4] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1273–1282, PMLR, 2017.

[6] A. Abusitta, M. Q. Li, and B. C. Fung, "Malware classification and composition analysis: A survey of recent developments," *Journal of Information Security and Applications*, vol. 59, p. 102828, 2021.

[7] M. N. Alenezi, H. Alabdulrazzaq, A. A. Alshaher, and M. M. Alkharang, "Evolution of malware threats and techniques: A review," *International journal of communication networks and information security*, vol. 12, no. 3, pp. 326–337, 2020.

[8] Rapid7, "Malware attacks." https://www.rapid7.com/fundamentals/malware-attacks/, 2024. Accessed: 2025-04-13.

[9] M. Polychronakis and N. Provos, "Ghost turns zombie: Exploring the life cycle of web-based malware.," *LEET*, vol. 8, pp. 1–8, 2008.

[10] malwarebytes, "Malware attacks." https://www.malwarebytes.com/malware/, 2024. Accessed: 2025-04-13.

[11] crowdstrike, "types-of-malware." https://www.crowdstrike.com/en-us/cybersecurity-101/malware/types-of-malware/, 2024. Accessed: 2025-04-13.

[12] Sudhakar and S. Kumar, "An emerging threat fileless malware: a survey and research challenges," *Cybersecurity*, vol. 3, no. 1, p. 1, 2020.

[13] R. Tahir, "A study on malware and malware detection techniques," *International Journal of Education and Management Engineering*, vol. 8, no. 2, p. 20, 2018.

[14] T. Malinda, "Android architecture, application components, and security model," 10 2023.

[15] P. Schulz and D. Plohmann, "Android security-common attack vectors," *Rheinische Friedrich-Wilhelms-Universitat Bönn, Germany, Tech. Rep*, 2012.

[16] cyberarrow, "android-malware-101-top-variants-how-to-detect-and-remove-it." https://www.cyberarrow.io/blog/android-malware-101-top-variants-how-to-detect-and-remove-it/, 2024. Accessed: 2025-04-14.

[17] L. Meijin, F. Zhiyang, W. Junfeng, C. Luyu, Z. Qi, Y. Tao, W. Yinwei, and G. Jiaxuan, "A systematic overview of android malware detection," *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2007327, 2022.

[18] csoonline, "malware-cybercrime-droiddream-autopsy-anatomy-of-an-android-malware-attack." https://www.csoonline.com/article/529368/malware-cybercrime-droiddream-autopsy-anatomy-of-an-android-malware-attack.html/, 2024. Accessed: 2025-04-14.

[19] medium, "analysis-of-joker-a-spy-premium-subscription-bot-on-googleplay." https://medium.com/csis-techblog/analysis-of-joker-a-spy-premium-subscription-bot-on-googleplay-9ad24f044451/, 2024. Accessed: 2025-04-14.

[20] developer, "privacy-and-security." https://developer.android.com/privacy-and-security/security-tips/, 2024. Accessed: 2025-04-14.

[21] googleplay, "android-developer." https://support.google.com/googleplay/android-developer/answer/10146128?hl=en&sjid=8619452913595451379-EU/, 2024. Accessed: 2025-04-15.

[22] digitdefence, "the-role-of-cyber-security-apps-for-modern-safety." https://digitdefence.com/blog/the-role-of-cyber-security-apps-for-modern-safety/. Accessed: 2025-04-15.

[23] ciosea, "security/top-five-malware-detection-evasion-techniques-in-2023." https://ciosea.economictimes.indiatimes.com/news/security/top-five-malware-detection-evasion-techniques-in-2023/103839267?utm_source/, 2023. Accessed: 2025-04-15.

[24] F. A. Aboaoja, A. Zainal, F. A. Ghaleb, B. A. S. Al-rimy, T. A. E. Eisa, and A. A. H. Elnour, "Malware detection issues, challenges, and future directions: A survey," *Applied Sciences*, vol. 12, no. 17, 2022.

[25] webasha, "future-of-ai-in-malware-analysis-how-ai-is-revolutionizing-cybersecurity." https://www.webasha.com/blog/future-of-ai-in-malware-analysis-how-ai-is-revolutionizing-cybersecurity#sec3/. Accessed: 2025-04-15.

[26] A. Intelligence, "Artificial intelligence (ai)," *How Does AI Work*, 2020.

[27] S. Das, A. Dey, A. Pal, and N. Roy, "Applications of artificial intelligence in machine learning: review and prospect," *International Journal of Computer Applications*, vol. 115, no. 9, 2015.

[28] J. E. van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, pp. 373–440, Feb 2020.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[30] B. Mahesh *et al.*, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, no. 1, pp. 381–386, 2020.

[31] Y. Song and Y. Lu, "Decision tree methods: applications for classification and prediction," *Shanghai Archives of Psychiatry*, vol. 27, pp. 130–135, Apr. 2015.

[32] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, vol. 408, pp. 189–215, 2020.

[33] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[34] K. Y. Chan, B. Abu-Salih, R. Qaddoura, A. M. Al-Zoubi, V. Palade, D.-S. Pham, J. D. Ser, and K. Muhammad, "Deep neural networks in the cloud: Review, applications, challenges and research directions," *Neurocomputing*, vol. 545, p. 126327, 2023.

[35] A. Tedjini and C. Mustapha, "An intrusion detection system based on federated deep learning," Master's thesis, 06 2024.

[36] M. Waheed and S. Qadir, "Effective and efficient android malware detection and category classification using the enhanced kronodroid dataset," *Security and Communication Networks*, vol. 2024, 04 2024.

[37] K. Kılıç, İsmail Atacak, and İbrahim Alper Doğru, "Fabldroid: Malware detection based on hybrid analysis with factor analysis and broad learning methods for android applications," *Engineering Science and Technology, an International Journal*, vol. 62, p. 101945, 2025.

[38] Daniel Arp1, Michael Spreitzenbarth2, Malte H¨ ubner1, Hugo Gascon1, Konrad Rieck1, "Drebin: Effective and explainable detection of android malware in your pocket." https://www.mlsec.org/docs/2014-ndss.pdf. Accessed: 2025-04-20.

[39] "Android malware dataset (cic-andmal2017)." https://www.unb.ca/cic/datasets/andmal2017.html. Accessed: 2025-04-20.

[40] A. K. Patel, "Understanding the confusion matrix for machine learning classification," *Machine Learning Advances*, vol. 5, no. 1, pp. 45–60, 2020.

[41] D. M. W. Powers, "Evaluation: From precision, recall and f-measure to roc, informedness, markedness & correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[42] L. Zhang, "Roc curve and auc in classification performance analysis," *Journal of Statistical Analysis*, vol. 7, no. 4, pp. 234–249, 2019.

[43] R. Thompson and M. Brown, "Far and frr in intrusion detection systems," *International Journal of Security and Privacy*, vol. 12, no. 2, pp. 67–82, 2018.

[44] P. Kaur and V. Laxmi, "Hybrid deep learning model for malware detection in android applications," *Machine Intelligence Research*, vol. 19, no. 1, pp. 461–473, 2025.

[45] M. N.-U.-R. Chowdhury, A. Haque, H. Soliman, M. S. Hossen, T. Fatima, and I. Ahmed, "Android malware detection using machine learning: A review," 2023.

[46] H. Gao, S. Cheng, and W. Zhang, "Gdroid: Android malware detection and classification with graph convolutional network," *Computers & Security*, vol. 106, p. 102264, 2021.

[47] P. Feng, J. Ma, T. Li, X. Ma, N. Xi, and D. Lu, "Android malware detection via graph representation learning," *Mobile Information Systems*, vol. 2021, pp. 1–14, 2021.

[48] J. Arrowsmith, T. Susnjak, and J. Jang-Jaccard, "Multimodal deep learning for android malware classification," *Machine Learning and Knowledge Extraction*, vol. 7, no. 1, 2025.

[49] W. Fang, J. He, W. Li, X. Lan, Y. Chen, T. Li, J. Huang, and L. Zhang, "Comprehensive android malware detection based on federated learning architecture," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3977–3990, 2023.

[50] R. Gálvez, V. Moonsamy, and C. Diaz, "Less is more: A privacy-respecting android malware classifier using federated learning," 2021.

[51] A. Chaudhuri, A. Nandi, and B. Pradhan, "A dynamic weighted federated learning for android malware classification," in *Soft computing: Theories and applications: Proceedings of SoCTA 2022*, pp. 147–159, Springer, 2023.

[52] C. Jiang, K. Yin, C. Xia, and W. Huang, "Fedhgcdroid: An adaptive multi-dimensional federated learning for privacy-preserving android malware classification," *Entropy*, vol. 24, no. 7, 2022.

[53] D. Hamouda, M. A. Ferrag, N. Benhamida, Z. E. Kouahla, and H. Seridi, "Android malware detection based on network analysis and federated learning," in *Cyber Malware: Offensive and Defensive Systems*, pp. 23–39, Springer, 2023.

[54] P. K. et al, "Advances and open problems in federated learning," 2021.

[55] "Ieee computer society call for papers," *IEEE Security & Privacy*, vol. 18, no. 3, pp. 57–57, 2020.

[56] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," 2019.

[57] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[58] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, pp. 611–629, 2018.

[59] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[60] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, p. 5455–5516, Apr. 2020.

[61] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Blockchain and federated learning for privacy-preserved data sharing in industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4177–4186, 2020.

[62] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5330–5340, 2017.

[63] Q. Yang, Y. Liu, Y. Kang, Y.-Q. Zhang, and T. Chen, "A comprehensive survey of federated transfer learning: challenges, applications and future directions," *Frontiers of Computer Science*, 2024.

[64] L. Yuan, Z. Wang, L. Sun, P. S. Yu, and C. G. Brinton, "Decentralized federated learning: A survey and perspective," *arXiv preprint arXiv:2306.01603*, 2023.

[65] Y. Liu, Y. Kang, Y.-Q. Zhang, and Q. Yang, "Hybrid federated learning: Algorithms and implementation," *arXiv preprint arXiv:2012.12420*, 2020.

[66] Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y.-Q. Zhang, and Q. Yang, "Vertical federated learning: Concepts, advances and challenges," *arXiv preprint arXiv:2211.12814*, 2022.

[67] J. Xu, B. S. Glicksberg, C. Su, P. Walker, and J. Bian, "Fairness and accuracy in horizontal federated learning," *Information Sciences*, vol. 571, pp. 94–107, 2021.

[68] S. Saha and T. Ahmad, "Federated transfer learning: Concept and applications," *arXiv preprint arXiv:2010.15561*, 2020.

[69] G. M. Nyabuto, V. Mony, and S. Mbugua, "Architectural review of client-server models," n.d. Unpublished manuscript.

[70] N. Abirami, S. Lavanya, and A. Madhanghi, "A detailed study of client-server and its architecture," n.d. Unpublished academic work.

[71] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[72] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan, *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[73] Y. Zhao, M. Li, M. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," in *arXiv preprint arXiv:1806.00582*, 2018.

[74] A. Hard, K. Rao, R. Mathews, and F. Beaufays, "Federated learning for mobile keyboard prediction," in *arXiv preprint arXiv:1811.03604*, 2018.

[75] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1322–1333, ACM, 2015.

[76] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[77] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *NeurIPS Workshop on Machine Learning on the Phone and other Consumer Devices*, 2017. https://arxiv.org/abs/1712.07557.

[78] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[79] Q. Qi, X. Luo, Z. Chen, W. Ni, and D. O. Liu, "Federated reinforcement learning: Techniques, applications, and open challenges," *IEEE/ACM Transactions on Networking*, 2021. https://arxiv.org/abs/2104.07078.

[80] Python.org, "Welcome to python.org," 2024. Accessed: May 11, 2025.

[81] F. Chollet, "Keras." https://keras.io, 2015. Accessed: May 11, 2025.

[82] T. Team, "Tensorflow: Large-scale machine learning on heterogeneous systems." https://www.tensorflow.org, 2016. Accessed: May 11, 2025.

[83] N. Developers, "Numpy." https://numpy.org, 2020. Accessed: May 11, 2025.

[84] W. McKinney, "Pandas: Python data analysis library." https://pandas.pydata.org, 2010. Accessed: May 11, 2025.

[85] S. learn Developers, "Scikit-learn: Machine learning in python." https://scikit-learn.org, 2011. Accessed: May 11, 2025.

[86] J. D. Hunter, "Matplotlib: 2d plotting library for python." https://matplotlib.org, 2007. Accessed: May 11, 2025.

[87] P. Team, "Pytorch: Deep learning framework." https://pytorch.org, 2019. Accessed: May 11, 2025.

[88] D. J. B. et al., "Flower: Federated learning framework." https://flower.dev, 2020. Accessed: May 11, 2025.

[89] C. I. for Cybersecurity (CIC) project in collaboration with Canadian Centre for Cyber Security (CCCS), "dataset: cicandmal2020." https://www.unb.ca/cic/datasets/andmal2020.html, 2020. Accessed: May 13, 2025.

[90] B. Salem *et al.*, "Principal component analysis (pca).," *La Tunisie Medicale*, vol. 99, no. 4, pp. 383–389, 2021.

[91] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[92] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023.

[93] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," 2019.