박 사 학 위 논 문
Ph.D. Dissertation

# 안드로이드 악성 코드 탐지 용 심층 추상화 및 프라이버시 보존 심층 학습 최신 기법 연구

Deep Abstraction for Android Malware Detection and Recent Development on Privacy-Preserving Deep Learning

2021

단우위자자 헤리 찬드라 (Tanuwidjaja, Harry Chandra)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

# 안드로이드 악성 코드 탐지 용 심층 추상화 및 프라이버시 보존 심층 학습 최신 기법 연구

2021

단우위자자 헤리 찬드라

한 국 과 학 기 술 원

전산학부

# 안드로이드 악성 코드 탐지 용 심층 추상화 및 프라이버시 보존 심층 학습 최신 기법 연구

단우위자자 헤리 찬드라

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2020년 12월 15일

심사위원장 김 광 조 (인)

심 사 위 원 이 기 혁 (인)

심 사 위 원 최 호 진 (인)

심 사 위 원 임 을 규 (인)

심 사 위 원 김 익 균 (인)

# Deep Abstraction for Android Malware Detection and Recent Development on Privacy-Preserving Deep Learning

Harry Chandra Tanuwidjaja

Advisor: Kwangjo Kim

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
December 15, 2020

Approved by

---

Kwangjo Kim
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS
20165650

단우위자자 헤리 찬드라. 안드로이드 악성 코드 탐지 용 심층 추상화 및 프라이버시 보존 심층학습 최신 기법 연구. 전산학부 . 2020년. 86+vi 쪽. 지도교수: 김 광 조.(영문 논문)

Harry Chandra Tanuwidjaja. Deep Abstraction for Android Malware Detection and Recent Development on Privacy-Preserving Deep Learning. School of Computing . 2020. 98+vi pages. Advisor: Kwangjo Kim. (Text in English)

**Abstract**

In this dissertation, we present our study about deep learning in two parts. The first part is about leveraging deep learning for Android malware detection. The second part is about Privacy-Preserving Deep Learning (PPDL) for Machine Learning as a Service (MLaaS)

Part I is focused on implementing feature learning for Android malware detection. The current Android malware detection method is limited to two kinds of methods, static and dynamic. Static method is easy to use but difficult to detect new kinds of malware. On the other hand, dynamic method is strong against a new malware but needs an expert skill to manipulate it. For the last decades, machine learning has advanced rapidly as a new malware detection method. We propose a modified feature learning method for malware detection, which is based on Deep Abstraction and Weighted Feature Selection proposed (DFES) for Intrusion Detection System. The methodology consists of a combination between Stacked Autoencoder (SAE) for feature extraction and weight based Artificial Neural Network (ANN) for feature selection and classification. The goal of this dissertation is to conduct a study to compare the performance of Modified DFES (mDFES), DFES, and a simple Feature Extraction and Selection (FES).

Part II is focused on study about leveraging deep learning for privacy-preserving. The exponential growth of big data and deep learning has increased the data exchange traffic in society. MLaaS, which leverages deep learning techniques for predictive analytics to enhance decision-making, has become a hot commodity. However, the adoption of MLaaS introduces data privacy challenges for data owners and security challenges for deep learning model owners. Data owners are concerned about the safety and privacy of their data on MLaaS platforms, while MLaaS platform owners worry that their models could be stolen by adversaries who pose as clients. Consequently, PPDL arises as a possible solution to this problem.We present a comprehensive study of privacy-preserving techniques, starting from classical privacy-preserving techniques to well-known deep learning techniques. Additionally, we provide a detailed description of PPDL and address the issue of using PPDL for MLaaS. Furthermore, we undertake detailed comparisons between state-of-the-art PPDL methods. Subsequently, we classify an adversarial model on PPDL by highlighting possible PPDL attacks and their potential solutions. Ultimately, our study serves as a single point of reference for detailed knowledge on PPDL and its applicability to MLaaS environments for both new and experienced researchers.

**Keywords** Android malware detection, feature learning, feature extraction, feature selection, privacy-preserving, machine learning as a service

# Contents

# List of Tables

# List of Figures

# Chapter 1. Preliminaries

## 1.1 Background

In this dissertation, we will present our research in deep learning for malware detection over Android and for privacy-preserving. The first part of this dissertation will talk about leveraging deep learning for Android malware detection, while the second part of this dissertation will discuss about our study in privacy-preserving deep learning for machine learning as a service.

Malware, which is an abbreviation of malicious software, is a hostile software that purpose to damage, steal, break, or do any illegal action to the victims. According to Statcounter, per October 2020 Android-based smartphone covers 72.92% of mobile operating system worldwide. On the other hand, Agrawal *et al.* [1] states that there are more than 2 million of Android applications in the market. Those applications cannot be 100% guaranteed safe, even by Google Android market. Currently, there are three kinds of Android malware detection methods: static method, dynamic method, and machine learning-based method. Static methods do the manual examination of manifest file and Dalvik byte code of an Android file. On the other hand, dynamic methods run an Android application in an emulated environment to observe its behavior. Unlike the other two methods, machine learning-based method for Android malware detector learns the general rules and features from benign and malicious dataset; then runs prediction to decide whether a file is a malware or not. The machine learning-based method is very beneficial for finding the regularities in a dataset, generating feature vectors based on that patterns. Those feature vectors are the most important thing that decides the performance of the machine learning model. Good feature vectors have features that highly represent its class, either it is benign or malware. This is the main challenge in leveraging machine learning for Android malware detection. Previous publications on malware detection have been proposed as follows: Alkhateeb [2] suggested the use of API similarity for detection. Lansheng *et al.* [3] had an idea to do classification on malware based on their task behavior. Yin *et al.* [4] showed how to find malware by analyzing the network traffic. However, all these methods require expert analysis and consume too much time. Our method leverages machine learning as substitution of human experts that requires much shorter time compared to human analysis. The challenges in utilizing machine learning are how to improve detection accuracy and how to reduce the false alarm rate as low as possible, but still require some amount of processing time.

In the part I of this dissertation, we use Android malware dataset, which consists of benign and malware files. Both benign and malware binaries were extracted into a list of feature in CSV format. This dataset then needs to be pre-processed in order to train the model during the learning process. Then, we use this pre-processed dataset as the input of our feature learning approach that consists of four main parts: dataset pre-processing, feature extraction, feature selection, and classification. During feature extraction, important features of malware will be extracted from a dataset and the features were going to be stored as our sample. Then, those new features and original features are concatenated and important features are selected from the concatenated features in feature selection phase. The feature extraction expands new features and feature selection chooses important features from them. During the evaluation, we check the detection accuracy and false alarm rate of our proposed approach.

MLaaS is a service, which usually runs on a cloud platform, with the purpose is to provide prediction service to clients by utilizing machine learning [5]. The service runs on a cloud environment so that

clients do not need to build their own machine learning model to do a prediction [6]. However, there is a problem. To perform predictions, a model owner needs to receive data from clients. The data may consist of sensitive information. Thus, clients are reluctant to provide their data. On the other hand, a model owner will also be worried that an adversary could be disguised as a client to try to steal the model. Furthermore, there is an issue about the privacy of the prediction result and whether will it be safe from access by unauthorized parties. In this scenario, Privacy-Preserving Deep Learning (PPDL) is needed as a solution.

For the part II of this dissertation, we will present our study about Privacy-Preserving Deep Learning (PPDL). In a business environment, prediction and decision-making are two important processes that require careful consideration. Good judgement can lead to large profits, but bad decisions can ruin everything. There was a hypothesis that a computer could help a user predict something or decide what next step should be taken. As Artificial Intelligence (AI) has grown dramatically, this plan is no longer considered impossible. AI has the ability to sense, understand, learn, and respond [7]. This solves the weaknesses of computers without these four abilities. Prediction, on the other hand, is a process of learning available information and then using that knowledge to generate new information that is not yet available. A Deep Learning (DL) algorithm is a type of AI that has the ability to interpret data like a human brain and can learn and classify objects. By leveraging the ability of deep learning, we can predict the future and make decisions based on the currently available information, which becomes our training data when we train the DL model. After the training process is completed, a prediction model is produced. Based on this model, predictions based on clients' data will be performed. That is how Machine Learning as a Service (MLaaS), a promising business opportunity, was born.

## 1.2 Our Contribution

Here is the contribution of this dissertation:

1. Conducting a study about deep learning techniques for Android malware detection and Privacy-Preserving Deep Learning (PPDL) for Machine Learning as a Service (MLaaS).
2. Design mDFES, a feature learning method for malware detection over Android.
3. Run experiments and analyze the performance of FES, DFES, and mDFES in nine different scenarios.
4. Propose a multi-scheme PPDL taxonomy that classifies adversarial models in PPDL, PP methods used in PPDL, and the challenges and weaknesses in state-of-the-art PPDL methods.
5. Provide detailed comparisons of the surveyed PPDL works based on our defined metrics and covers the most recent and groundbreaking methods in PPDL.

## 1.3 Structure of Dissertation

This dissertation is divided into two parts. First part is about feature learning for Android malware detection and the second part is about privacy-preserving deep learning. The remainder of this dissertation is organized as follows.

In Part I, we present our research about deep abstraction for Android malware detection. Chapter 2 provides the introduction and Chapter 3 states our objectives. In Chapter 4, we discuss about malware

and anti-malware detections. In Chapter 5, we briefly discuss about previous publications about malware detection in general. In Chapter 6, we explain our methods; while in Chapter 7, we present our experiment result and comparison. Finally, in Chapter 8 we conclude our findings as the closing of Part I.

In Part II, we present our study about recent development on privacy-preserving deep learning. Chapter 9 provides the introduction and Chapter 10 discusses the classical privacy-preserving method. In Chapter 11 we discuss about utilizing deep learning for privacy-preserving. In Chapter 12, we present state-of-the-art PPDL methods. Next, in Chapter 13, we provide the comparison of discussed PPDL methods in previous chapter. Chapter 14 talks about attacks on deep learning model and privacy-preserving as deep learning as a possible solution. Finally, in Chapter 15, we provide the concluding remark of Part II and also the possible future work.

Lastly, the final Chapter 16 provides the dissertation summary and the open problems that concludes this dissertation.

# Chapter 2. Introduction

## 2.1 Definition and Classification of Deep Learning

### 2.1.1 Supervised Learning

Supervised learning is an approach where there is already trained data, and there are targeted variables so that the purpose of this approach is to group data into existing data, unlike unsupervised learning, unsupervised learning does not have training data, so that from existing data, we group the data into 2 parts or 3 parts and so on.

### 2.1.2 Unsupervised Learning

The unsupervised learning approach does not use training data or training data to make predictions or classifications. Based on the mathematical model, this algorithm does not have a target variable. One of the goals of this algorithm is to group objects that are almost the same in a certain area.

### 2.1.3 Semi-supervised Learning

Semi-supervised learning combines the two algorithms above (supervised and unsupervised), where the input samples given are labeled and some are not labeled. This algorithm generates an appropriate function based on all given inputs.

**Reinforcement Learning** Reinforcement learning studies a policy of how to take action based on observations of the existing environment. Each action produces an impact on the environment, and the environment provides feedback (feedback) to guide the algorithm.

## 2.2 Definition and Classification of Malware

### 2.2.1 Malware Classification Based on the Characteristic

Based on its characteristic, malware can be classified into ten types. For the classification, Aycock [8] uses three parameters: self-replicating, number of population growth, and parasitic traits. Based on those parameters, there are logic bomb, Trojan horse, virus, worm, rabbit, spyware, adware, hybrid, and zombies. Self-replicating is a malware ability for replicate himself. Number of population growth is the increase number of malware population. Meanwhile, parasitic malware is a malware that will damage the victim's system when the malware is activated.

#### Logic Bomb

Logic bomb is a malware that consists of two parts, payload and trigger. Payload is a part that contains code about the action performed by the malware. Malware will be activated when the activation condition is fulfilled. That condition is called trigger. The trigger can be various, depends on the creator of the malware. One example of the most commonly used triggers are date and time. Logic bomb cannot replicate himself, its population growth is zero, and it has possibility to be parasitic.

**Trojan horse**

Trojan horse is a malware whose name is taken from Greek's myth. Greek's soldiers hid inside a giant wood horse sent to Troy city and did surprise attack when the enemies were off guard. Trojan horse has the same characteristic, it does its activities in secret. One example of the activity is recording the victim's keyboard input. Trojan horse does not have self-replicating ability, its population growth is zero, and it is parasitic.

**Backdoor**

Backdoor is a malware that has an ability to bypass normal security check. Usually, programmers make a backdoor to bypass authentication process that takes a long time when they do server debugging. One example of backdoor is called Remote Administration Tool (RAT). Backdoor monitors victim's activity remotely. Usually, a backdoor is used by workers to access their office computers when they are at home. However, adversaries take advantage of RAT and install it on victim's computer. Backdoor does not have self-replicating ability, zero population growth, and has possibility to become parasitic.

**Virus**

Virus is a malware that tries to duplicate itself to other executable files when the malware is executed. This will make the victim's device run slow as thousands of virus duplicating themselves. Virus has self-replicating ability, positive population growth, and it is parasitic.

**Worm**

Worm is a malware that has similar characteristic with virus. The main difference between worm and virus is that worm is standalone; does not parasitize other files and spreads over network, not through physical media. The word worm was first used in 1975 by John Brunner in his book "The Shockwave Rider". The first computational experiment on worms was carried out by Xerox PARC in 1980, after previously a worm called Creeped Crawled appeared around 1970. Worms have the ability to self-replicate, have population growth, and are not parasitic.

**Rabbit**

Rabbit is a malware that has self-replicating capabilities, but zero population growth. This is because that malware moves to another device by deleting himself from the previous infected device. Therefore, rabbit has zero population growth but has the ability to self-replicate. Rabbit is not parasitic. Rabbit is often used when the adversary needs a malware that can reproduce rapidly. Basically, a rabbit has two main characteristics. The first one is trying to eat all system resources; for example, a fork bomb that attempts to create a new process in an infinite loop. The second characteristic is that only one rabbit hopping from 1 PC to another PC. After moving, the previous data will be deleted.

**Spyware**

Spyware is a malware that collects information from a victim's computer, then sends it to the attacker. Information commonly targeted by spyware users are usernames and passwords, email addresses, bank accounts and credit card numbers, or software license keys that can be used to commit piracy. Viruses and worms can collect similar information, but the difference is that a spyware does not have

self-replicating capabilities. Spyware is usually spread by injection into pirated software, so that when the victim installs the software, the spyware will be embedded in the victim's computer. In addition, spyware can also be installed when the victim enters a website, which is often referred to as a drive-by download. Spyware has zero population growth and it is not parasitic.

#### Adware

Adware is a malware that has spyware-like characteristics. The difference is that an adware has a specific purpose, namely spying to find out user habits; which is usually for the sake of advertising. The sign that our computers are infected with malware is the sudden appearance of pop up ads on our computer screens. The adware has no self-replicating capability, zero population growth, and it is not parasitic.

#### Hybrid

Hybrid is a malware that results from a combination of several types of malware. An example of a hybrid malware was introduced by Ken Thompson at the ACM Turing Award Lecture [9]. The lecture introduces a Trojan that can serve as a backdoor and can reproduce itself like a virus. Another example of a hybrid is called Animal, which was created by John Walker in 1975 [10]. An Animal copies itself to all directories on the computer, but does not destroy any data. It also does not use the Internet when it is active, either to spread itself or to send information. Therefore, Animals are a mixture of worms, Trojans and viruses.

#### Zombies

Devices that have been taken by the adversaries so that they can be used to carry out attacks are called zombies. Zombies are usually used by adversaries to send spam and launch denial of service attacks. In order to carry out the Denial-of-service (DDoS) attack, it takes a lot of computers to send requests to the target computer so that the target computer traffic becomes overloaded. That is why, a huge number of zombies are required.

### 2.2.2 Malware Classification Based on the Target

Based on its target, malware can be classified into three kinds: boot sector infectors, File Infectors, and Macro Virus.

#### Boot Sector Infectors

The Boot Sector Infector (BSI), is malware that infects itself by copying itself to the boot block of the victim's PC. It will copy the original boot block component to another place first so that the malware can complete the infection process after the boot process starts. In the past, this type of malware was indeed quite effective. This is because that it attacks the computer during the boot process, before the antivirus can work. However, now it is very rare to find malware that attacks the boot sector. Most BIOSes are protected so that the operating system will prohibit all writing request to the boot block without official authorization from the admin.

#### File Infectors

It is a virus that infects executable files on the target computer. If the boot sector malware infects a boot block that is outside the OS, then the file infector will infect the files on our computer system. There are five alternative locations for file infectors to infect the target file. The five sections are the beginning of file, end of file, overwritten into file, inserted into file, and not in file. The file infector that infects the beginning of the file will place itself at the beginning of the script file. Therefore, when the file is opened by the victim, the malware will run immediately. Contrary to the beginning of the file, there is also a malware that infects the end of the file. This type of malware will change the start location address to the location of the malware code, then use the jump command to return to the actual initial code position so that the file can run normally. The third alternative location for malware infection file infectors is overwritten into the file. Overwriting into file is intended so that the original file size does not change drastically as in the case of the beginning of a file and the end of a file. The overwriting process is quite difficult as it can damage the original file. The overwritten part must be the data sector, not the code sector because it could potentially damage the original file. The fourth alternative is inserted into a file. In this method, a malware is inserted into a file in the code sector. This method is quite difficult to do because you have to restructure the file or the file will be damaged. The last alternative is not in file. Malware will place itself with the same name as the target file so that the malware will be executed first when someone tries to run the target file. The target file will be renamed with a different name.

#### Macro Virus

Some applications allow the use of file data, such as in Microsoft word. These data files are commonly called macros. Macros are code written in a language that the application can interpret. After the macros are installed to global macros, all files opened by the application become exposed to the malware. When someone runs an application to open a file, the application will open global macros and the file becomes infected with malware.

### 2.2.3 Malware Classification Based on the Concealment Strategy

Based on its concealment strategy, malware is classified into seven types. The seven types of malware include no concealment, encryption, stealth, metamorphism, strong encryption, polymorphism, and oligomorphism.

#### No Concealment

This type of malware does not have the ability to hide itself from antivirus detection so that it can be detected easily.

#### Encryption

The body of the virus is encrypted with a certain key so that the antivirus will not be able to detect any malicious code in it. There are five types of encryption; namely simple encryption, static encryption keys, variable encryption keys, substation ciphers, and strong encryption. Simple encryption does not use keys, only arithmetic and basic logic such as plus, minus, rotation, negation. Static encryption uses one static key. Variable encryption key uses a key that changes based on the variable value used so that the malware code will be more difficult to be detected. Substitution cipher Is a 1:1 encryption technique.

Strong encryption combines arithmetic logic, variable encryption key, and substitution cipher so that it becomes harder to be broken.

**Stealth**

Stealth malware is malware that can completely hide the infection. One example is by not changing the timestamp of the infected file so that the file is still the same as before.

**Oligomorphism**

Oligomorpism malware is a semi polymorphic malware that has a small loop decryptor. Therefore, antivirus experts can easily brute force it to solve it.

**Polymorphism**

Polymorphism is a developed form of oligomorphism; by increasing the number of the loop decryptor. One of them is called Tremor which has 6 million loop decryptor. Due to the large number of them, it is impossible to do brute force since it takes a very long time.

**Metamorphism**

Metamorphism is a malware that has a modified body structure. Therefore, the antivirus will not detect it because with a changed body structure, the location of the virus signature is changed too.

**Strong Encryption**

Strong encryption is a malware with two encryption keys that are distributed separately. However, the chances of the two parts meets on the same device are quite small so that they are rarely used.

# Chapter 3. Objectives

## 3.1 Research Goals

In the first part of this dissertation, we want to achieve these goals:

1. Conducting a study about deep learning techniques for Android malware detection.
2. Propose mDFES, a stacked autoencoder-based feature learning method for detecting Android malware.
3. Providing comparison and analysis for three methods: FES, DFES, and mDFES using Drebin and Malgenome dataset.

## 3.2 Research Hypothesis

We also set these hypothesis in our Android malware detection experiment:

1. Among FES, DFES, and mDFES, FES should be the fastest but has the worst accuracy
2. mDFES should be faster than DFES since mDFES reduce the number of non-extracted feature in the classification phase.
3. The type of dataset affects the performance of deep learning method. In a same environmental setting, method X gives better performance than method Y for Dataset A, but method Y gives better performance than method X for Dataset B.

# Chapter 4. Malware and Anti-Malware Detections

## 4.1 Malware Detection Method

Generally, malware detection method can be divided into three kinds: static detection method, dynamic detection method, and machine learning-based detection method.

### 4.1.1 Static Detection Method

The static detection method is a detection method without executing the file that is going to be examined. There are three kinds of static detection techniques: scanners, heuristics, and integrity checkers. There are two type of scanners method; on demand and on access. On demand means that the scanning process is done by the user request. On access means that the scanning process will automatically start when a file is accessed. The heuristics method looks for certain codes; such as self-modifying code, decryption loop, and unusual instructions that tries to access the root or registry. Lastly, integrity checkers compares the checksums of a file over time. If the file has been infected with a virus, the checksum will change.

### 4.1.2 Dynamic Detection Method

Dynamic detection method is a detection method by executing the file that want to be examined and then observing its behavior. Dynamic detection methods can be divided into two types, namely behavior monitoring and emulation. Behavior monitoring checks the behavior of a program, while emulation executes the program in an emulated environment.

### 4.1.3 Machine Learning-based Detection Method

Machine learning-based methods can be classified into supervised learning [11], unsupervised learning [12], semi-supervised learning [13], and reinforcement learning. Autoencoder [14] is a neural network model that has similar input and output. It works by learning the characteristic of the input then try to do reconstruction to that input data. Autoencoder is useful for reducing the dimensionality of a complex dataset [15]. When we have high dimensional data with many features, high number of training instances are needed to achieve a good accuracy. In order to solve this problem, the dimensionality reduction technique is required. An autoencoder consists of two parts: encoder and decoder. Between the encoder and the decoder, there are several layers called hidden layer. In those hidden layers, representations of the data are generated. A sparse autoencoder is an autoencoder that has sparsity penalty, which is applied on the hidden layer when a rescontruction error occurred. The advantage of the sparsity penalty is preventing overfitting. A sparse autoencoder also takes the highest activation value in the hidden layers and zero out the rest of the nodes, which forces to use a reduced number of hidden nodes; not all of the hidden nodes.

## 4.2 Anti-Malware Detection Method

### 4.2.1 Retroviruses

Retroviruses are malwares that actively work to disable antivirus software so that the antivirus cannot work.

### 4.2.2 Obfuscation

Obfuscation is a technique to modify a malware so that the antivirus cannot find the virus signature inside it. The virus signature can be removed or relocated.

### 4.2.3 Anti-Emulation

Anti-Emulation can be divided into outlast, outsmart, and overextend. The outlast method is a method for testing the patience of users who want to emulate malware files by buying some time. Outsmart is a technique that restructures malware code so that it does not look suspicious. Overextend is a method for pushing the emulator to its limit so that it crashes.

### 4.2.4 Armoring

Armoring is a technique of adding protection to malware so that it becomes difficult to analyze. Against dynamic analysis, anti-debugging armor is used, whereas to counter static analysis, anti-disassembly armor is required.

### 4.2.5 Tunneling

Antivirus software usually monitors connections using API code to an operating system to detect suspicious activity. Tunneling malware is a malware that is able to track whether an API code is under surveillance or not.

### 4.2.6 Integrity Checker Attack

This technique works against antiviruses that match the checksum of files over time. The process is usually done by deleting the checksum database that is stored on the computer so that the antivirus cannot check it.

### 4.2.7 Avoidance

Avoidance is a technique of avoiding infecting file types that are usually checked by antivirus. For example, the files that access the operating system registry or root.

# Chapter 5. Previous Publications

Malware detection has been studied for decades. The old detection technique includes static detection and dynamic detection. Lately, several machine learning based malware detection methods have been proposed.

Some methods use malware behaviors as the features of machine learning for malware detection. Shibahara *et al.* [16] proposed a deep learning approach based on characteristic of malware communication using Recurrent Neural Network (RNN). They showed that the proposed method reduced 67.1% of analysis time while keeping the range of covered URL to 97.9% compared to full analysis method. Kolosnjaji *et al.* [17] proposed deep learning based malware detection using system call sequence. They showed that the combination of Convolutional Neural Network (CNN) and Long-Short Term Memory (LSTM) gives better accuracy, compared to feedforward network and convolutional network. Firdausi *et al.* [18] proposed an automatic behavior based malware detection using machine learning. They used five classifiers including k-Nearest Neighbors (k-NN), Naïve Bayes, J.48 Decision Tree, Support Vector Machine (SVM), and Multilayer Perceptron Neural Network (MLP). Their experiment showed that J.48 classifier gave the best performance. Rieck *et al.* [19] proposed a malware detection scheme based on malware behavior using machine learning. They showed that the incremental technique in malware behavior based analysis successfully decreased run time and memory requirement, compared to regular clustering.

There are methods that combine feature extraction and classification in machine learing for malaware detection. Tobiyama *et al.* [20] proposed a malware detection method using deep neural network based on data traffic on computer. They used RNN for feature extraction and CNN for classification. Their proposed method achieved 92% detection accuracy. David *et al.* [21] proposed Deepsign, a deep learning approach for automatic malware signature generation and classification. They used Deep Belief Network (DBN) to produce malware signatures. Their proposed approach reached 98.6% accuracy with 0.2 input noise and 0.001 learning rate. Xu *et al.* [22] proposed machine learning based malware detection by using virtual memory access patterns. They used three classifiers (SVM, Random Forest, and Logistic Regression) to do training phase. They showed that the best performance was achieved by random forest with 99% true positive rate and 1% false positive rate. Liu *et al.* [23] proposed a combination between image processing and machine learning. They used opcode n-gram with gray scale images to extract malware features. Then, they did clustering process using Shared Nearest Neighbor (SNN) clustering algorithm. They reached 96.5% accuracy by using random forest classifier.

Rathore *et al.* [24] proposed random forest based deep learning with opcode frequency as feature vector for malware detection. Vinayakumar *et al.* [25] combined image processing with deep learning for hybrid zero-day malware detection. While Xiao *et al.* [26] proposed behavior based deep learning framework to detect malware in cloud service environment. The extracted API calls and use it as features during the learning process. Zhong *et al.* [27] proposed multi level deep learning structure that utilizes tree structure to do clustering on malware detection system. Liu *et al.* [28] implemented malware detection system by leveraging deep learning on API calls. Karbab *et al.* [29] extraced API from Android devices as features for deep learning based malware detection on IoT devices.

Other methods combine feature selection and classification in machine learning for malware detection. Raman *et al.* [30] proposed an approach to do feature selection in malware classification, with the

addition of using intuitive method during feature selection. They used random forest algorithm to do feature selection. Then, four classifiers, PART, IBk, J48Graft, and J48 were used to choose the highest seven features. Gandotra *et al.* [31] proposed zero-day malware detection by combining static and dynamic malware analysis with machine learning algorithm. They generated their own dataset from Virus Share (for malwares) and Windows system directories (for benign files). They did feature selection by using information gain method, an entropy based technique for selecting features. Then, for classification, they used seven classifiers from Weka, including IB1, Naïve Bayes, J48, Random Forest, Bagging, Decision Table, and Multi-layer Perceptron. The best performance was achieved by Random Forest with 99.97% accuracy.

# Chapter 6. Our Methods

**Dataset Pre-processing** In this dissertation, we use two well known Android malware dataset: Malgenome [32] and Drebin [33] collected by Yerima *et al.* [34]. The Malgenome dataset contains 3,799 instances with 215 static features extracted from 1260 malware apps and 2539 benign apps. The Drebin dataset contains 15,036 instances with 215 static features extracted from 5560 malware apps and 9476 benign apps. The details of those dataset can be seen in the appendix and source code page. In order to use the dataset; firstly, we need to pre-process it. The purpose of dataset pre-processing is to convert the data format into a process-able input for the feature extraction. There two main processes: dataset normalization and dataset balancing. In dataset normalization, we convert benign (B) string in the dataset into 1 value and malware (S) string in the dataset into 0 value. In dataset balancing, we balance the size of the dataset. We ignore this step if we want to use unbalanced dataset during the experiment. The next process is converting the data into double format and exporting it into csv file. The flowchart of dataset pre-processing can be seen in Fig. 6.1.

**Algorithm 1** Dataset Pre-processing Algorithm

```
1: function DATASET PRE-PROCESSING(InputData)
2:     Open Dataset
3:     Read InputData
4:     function DATASET NORMALIZATION(InputData)
5:         return NormalizedDataset
6:     end function
7:     function DATASET BALANCING(NormalizedDataset)
8:         return BalancedDataset
9:     end function
10:    return InputDataFinal
11: end function
```

**Feature Extraction** Feature extraction is a process to extract new features from the original features. The main purpose is to reduce the original features' dimensionality. The compressed representation preserve the essential information from the original input. In this dissertation, we use Autoencoder [35] for the feature extraction phase. Autoencoder is a neural network model for dimensionality reduction in a learning process. When we have high dimensional data with many features, the feature can be spread. As a result, we need many training data during the learning process. Another way to address this issue is by reducing the data dimension. That is the reason why we need an autoencoder. An autoencoder has one hidden layer and same number of input and output [36]. Stacked Autoencoder (SAE), as shown in Fig. 6.2, is a neural network that consists of multiple encoders [37]. In our approach, we use two autoencoders for feature extraction [38] and train them separately. However, the two autoencoders are not dependent from each other. The second autoencoder uses the hidden layer from the first autoencoder and the number of neurons in each hidden layer will decrease accordingly. During the SAE training, we use unlabeled dataset. The labeled dataset can be used during classification process. The classification process can be done by softmax regression function in the last step of the training process [39]. The

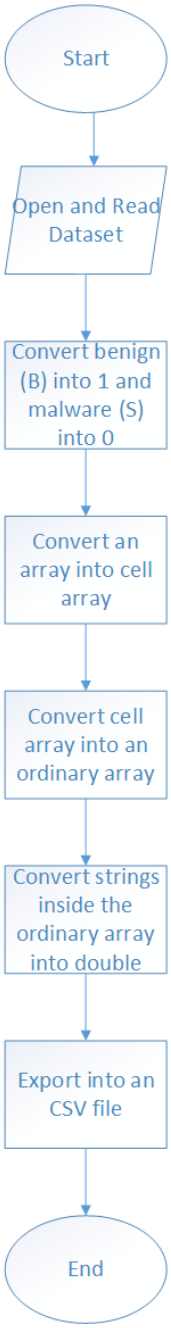


Figure 6.1: The flowchart of dataset pre-processing

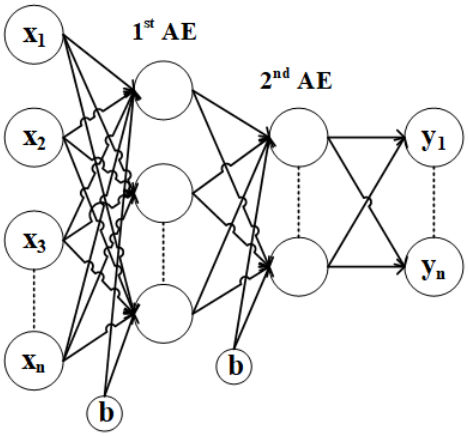


Figure 6.2: Stacked Autoencoder (SAE) network with two hidden layers

flowchart of feature extraction can be seen in Fig. 6.3.

**Algorithm 2** Feature Extraction Algorithm

```
1: function FEATURE EXTRACTION(InputDataFinal)
2:     Import InputDataFinal
3:     for i=1 to j do                          ▷ j=number of hidden layers
4:         for each data instance do
5:             Compute hidden representation y_i
6:             Reconstruct the instance z_i
7:             Minimize the cost function E_i
8:         end for
9:     end for
10:    return Extracted Features
11: end function
```

**Feature Selection** Feature selection is a process to select important features from the input features. Compared to feature extraction, no new feature is produced in feature selection; since we only choose the essentail representation. Feature selection is done by measuring the weight of all features, calculating the average weight value, set the threshold, and finally choose all features that have weight value higher than the threshold. In this dissertation we use Artificial Neural Network (ANN) [40] for feature selection phase. ANN, as shown in Fig. 6.5, is a neural network model that consists of many neurons. Each neuron receives input, multiplies the input with weight, and adds bias, resulting in parameters for activation function. The activation function decides whether that neuron should be active or not, based on the weighted sum [41]. In our proposed approach, we use ANN to do feature selection and classification. The ANN can be trained with two target classes: benign class and malware class. The flowchart of feature extraction can be seen in Fig. 6.4.

We measure the weight of each feature in dataset to decide which feature is important. The weight here represents the level of influence from input feature to the first hidden layer [42]. If the value is small (nearly zero), it means that the feature is not a deciding factor to pick whether a file is a malware or benign file. We will measure the average weight of all features and set a threshold value. We pick all

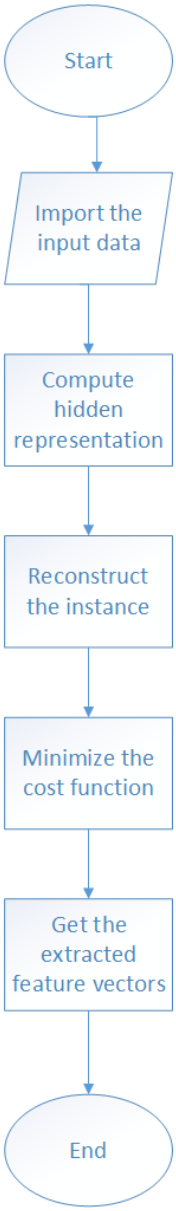


Figure 6.3: The flowchart of feature extraction

**Algorithm 3** Feature Selection Algorithm

**function** FEATURE SELECTION($FeatureVectors$)
    Import $FeatureVectors$
    Training ANN
    $W_{ij}$
    **for** each input feature **do**
        Compute *weight*
    **end for**
    Sort descending
    $SelectedFeatures \leftarrow weight > threshold$
    **return** $SelectedFeatures$
**end function**

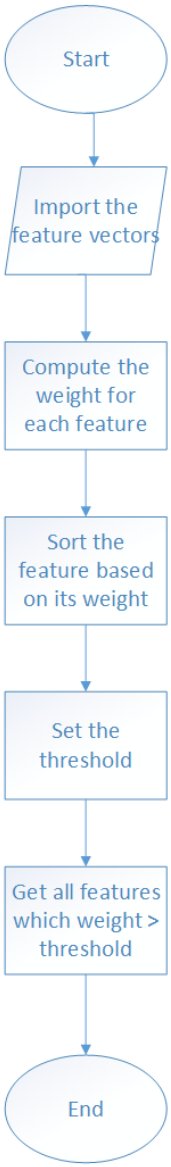


Figure 6.4: The flowchart of feature selection

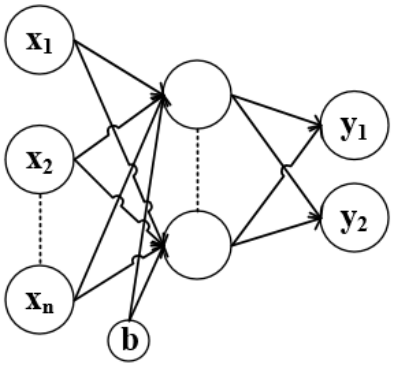


Figure 6.5: The structure of Artificial Neural Network

features that have weight value higher than the threshold. This is how we do the feature selection [43]. We also use ANN for classification process. Our scheme executes minimum global error function with a scale conjugate gradient optimizer [44] in supervised learning environment. We decide to use supervised approach because it will increase the performance of our classifier.

## 6.1 Feature Extraction Selection (FES)

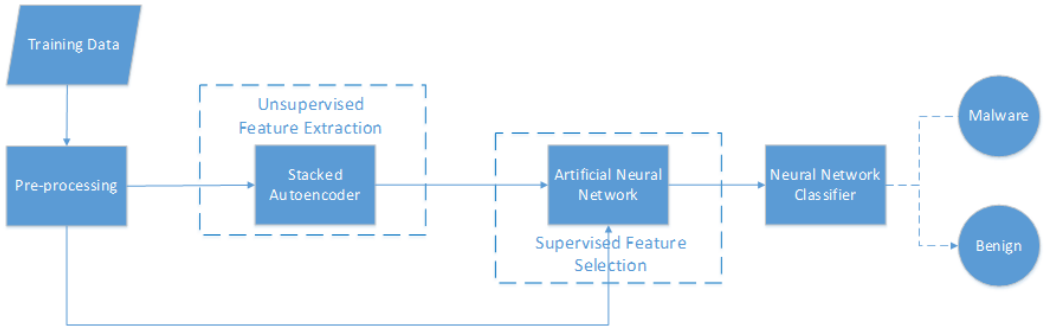


Figure 6.6: Structure of FES

The structure of FES is shown in Fig. 6.6. This method consists of feature extraction and feature selection, which is the basic structure for feature learning. The feature extraction reconstructs its input and transforms the original inputs from dataset into a meaningful representation. The feature selection measures the weight of each features and selects features which weights are higher than the average weight. The classification does learning process with a minimum global error function with a scaled conjugate gradient optimizer.

## 6.2 Deep Feature Extraction and Selection (DFES)

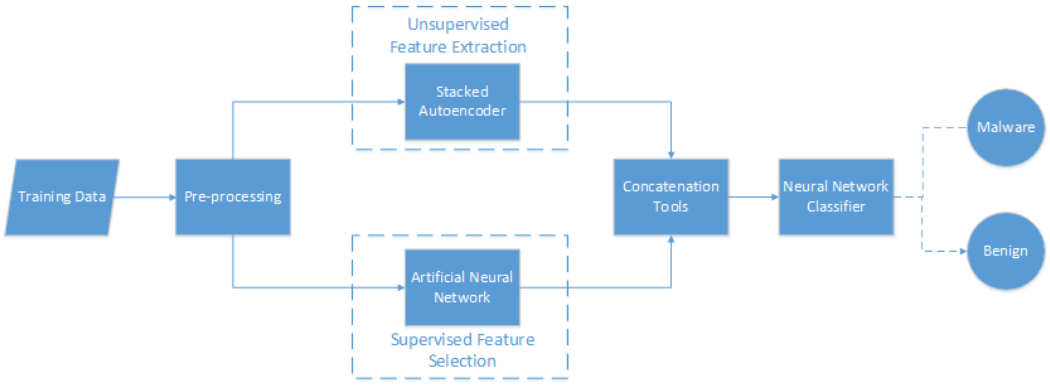


Figure 6.7: Structure of DFES

The structure of DFES is shown in Fig. 6.7. This method consists of a combination between feature extraction, feature selection, and classification for learning the essential meaning from a massive dataset. DFES enhanced the FES method by providing deep abstraction, concatenating the extracted features with the original features as the input of feature selection.

## 6.3 Modified Deep Extraction and Selection (mDFES)

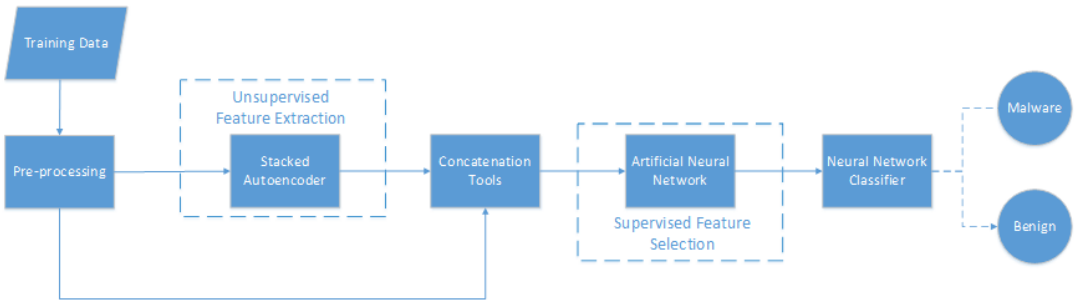


Figure 6.8: Structure of mDFES

The structure of mDFES is shown in Fig. 6.8. This method modifies DFES by concatenating the extracted features (result of feature extraction) and selected feature from the original features. The difference with the DFES is that we increase the output of feature extraction, while reducing the input of feature selection. The feature selection only selects features from the original features. This idea comes from the fact that the feature extraction of SAE is capable to transform original features from a massive dataset into more meaningful; but less complex features. That is why we decide to do feature selection on original features only and then concatenate the result with extracted features. The concatenated features become the input of learning process during the classification phase. Extracted features should have more meaningful representation than original features. The nature of stacked Autoencoder guarantees that the autoencoder is actually learning latent representations, instead of redundant information in the input data [45]. So, in mDFES; we reduce the number of original feature and use all extracted features during the classification. Based on those logical theory, we expect that mDFES can gives better accuracy than DFES with faster classification time.

# Chapter 7. Our Experiments and Comparison

## 7.1 Evaluation Metrics

In order to measure the performance of our proposed method, we use several evaluation metrics. We use the most well referenced parameter measurements [46], including accuracy, detection rate, false alarm rate, false negative rate, F1 score, and precision. Accuracy (Acc) means the proximity of measured result to the true value. Detection rate (DR) refers to the number of malwares detected, divided by the total number of malwares. False Alarm Rate (FAR) is the number of benign files that is detected as malwares, divided by the total number of benign files in the dataset. False Negative Rate (FNR) is the number of malwares that is wrongly detected as benign, divided by the total number of malwares in the dataset. F1 score is a measurement of harmonic mean between precision and recall. Precision is the number of correctly detected malwares, divided by the number of files that is detected as malwares. The measurement formulas can be defined as shown in Eqs. (7.1) to (7.5):

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

$$DR = Recall = \frac{TP}{TP + FN} \tag{7.2}$$

$$FAR = \frac{FP}{TN + FP} \tag{7.3}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{7.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{7.5}$$

## 7.2 Outcome

In this dissertation, we conduct a study of stacked autoencoder-based feature learning method using two Android malware datasets: Malgenome and Drebin dataset. We performed our experiment in several scenarios, as shown in Table 7.1.

**Scenario 1: FES-DREB-BAL** In this scenario, we used balanced Drebin dataset with FES method. The experiment result shows that this setup achieved 92% accuracy. The detection rate is 93.83%, the false alarm rate is 9.74%, and the F1 score is 92.07%. Theoretically, FES structure should give the worst performance compared to the DFES and mDFES. When we compared this scenario result with DFES (Scenario 5) and mDFES (Scenario 9), the performance result of this scenario is lower; the accuracy result of Scenario 5 and 9 are 97.95% and 95.66%, respectively.

**Scenario 2: FES-DREB-UNB** In this scenario, we use unbalanced Drebin dataset with FES method. The experiment result shows that this setup achieved 92.86% accuracy. The detection rate is 89.42%, the false alarm rate is 5.13%, and the F1 score is 90.24%. Theoretically, FES structure should give the worst performance compared to the DFES and mDFES. When we compared this scenario result with DFES (Scenario 6) and mDFES (Scenario 10), the performance result of this scenario is lower; the accuracy result of Scenario 5 and 9 are 97.10% and 96.33%, respectively.

Table 7.1: List of All Experiment Scenarios

| Scenario Number | Method | | | Dataset Name | | Dataset Type | |
|---|---|---|---|---|---|---|---|
| | FES | DFES | mDFES | DREB | MALG | BAL | UNB |
| 1 | √ | | | √ | | √ | |
| 2 | √ | | | √ | | | √ |
| 3 | √ | | | | √ | √ | |
| 4 | √ | | | | √ | | √ |
| 5 | | √ | | √ | | √ | |
| 6 | | √ | | √ | | | √ |
| 7 | | √ | | | √ | √ | |
| 8 | | √ | | | √ | | √ |
| 9 | | | √ | √ | | √ | |
| 10 | | | √ | √ | | | √ |
| 11 | | | √ | | √ | √ | |
| 12 | | | √ | | √ | | √ |

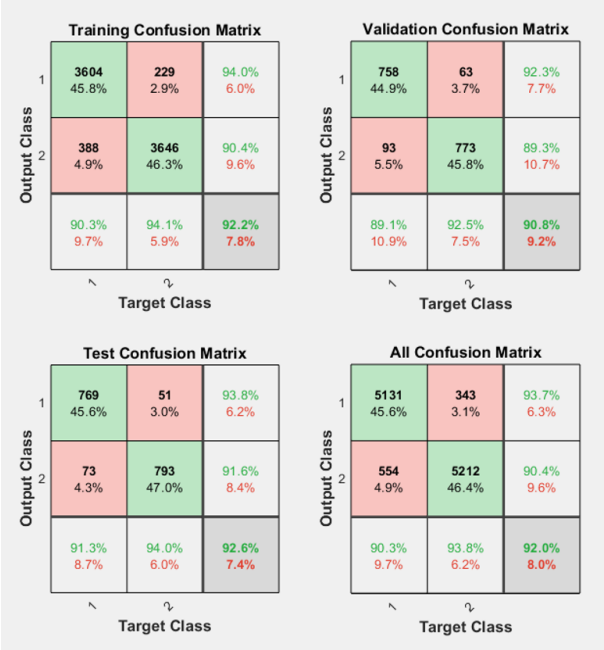


Figure 7.1: Confusion matrix of scenario 1

Table 7.2: Experimental result of FES-DREB-BAL

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 3604 | 388 | 3646 | 229 | 94.090% | 9.719% | 92.199% | 92.157% |
| **Validate** | 758 | 93 | 773 | 63 | 92.464% | 10.928% | 90.834% | 90.753% |
| **Test** | 769 | 73 | 793 | 51 | 93.957% | 8.670% | 92.749% | 92.645% |
| **All** | 5131 | 554 | 5212 | 343 | 93.825% | 9.745% | 92.077% | 92.020% |

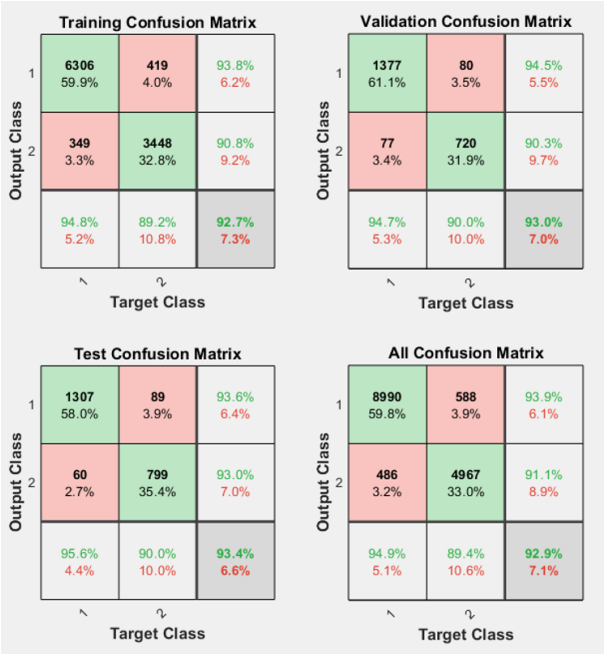


Figure 7.2: Confusion matrix of scenario 2

Table 7.3: Experimental result of FES-DREB-UNB

| | **TN** | **FP** | **TP** | **FN** | **DR** | **FAR** | **F1** | **Acc** |
|---|---|---|---|---|---|---|---|---|
| **Train** | 6306 | 349 | 3448 | 419 | 89.165% | 5.244% | 89.979% | 92.701% |
| **Validate** | 1377 | 77 | 720 | 80 | 90.000% | 5.296% | 90.169% | 93.035% |
| **Test** | 1307 | 60 | 799 | 89 | 89.977% | 4.389% | 91.471% | 93.392% |
| **All** | 8990 | 486 | 4967 | 588 | 89.415% | 5.129% | 90.243% | 92.855% |

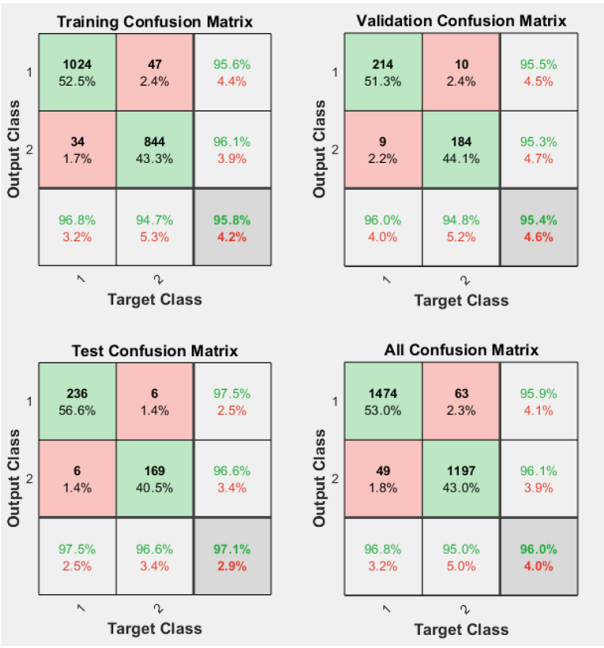


Figure 7.3: Confusion matrix of scenario 3

Table 7.4: Experimental result of FES-MALG-BAL

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 1024 | 34 | 844 | 47 | 94.725% | 3.214% | 95.421% | 95.844% |
| **Validate** | 214 | 9 | 184 | 10 | 94.845% | 4.036% | 95.090% | 95.444% |
| **Test** | 236 | 6 | 169 | 6 | 96.571% | 2.479% | 96.571% | 97.122% |
| **All** | 1474 | 49 | 1197 | 63 | 95.000% | 3.217% | 95.531% | 95.976% |

**Scenario 3: FES-MALG-BAL** In this scenario, we use balanced Malgenome dataset with FES method. The experiment result shows that this setup achieved 95.98% accuracy. The experiment result shows that this setup achieved 95.98% accuracy. The detection rate is 95.00%, the false alarm rate is 3.22%, and the F1 score is 95.53%. Theoretically, FES structure should give the worst performance compared to the DFES and mDFES. When we compared this scenario result with DFES (Scenario 7) and mDFES (Scenario 11), the performance result of this scenario is lower; the accuracy result of Scenario 7 and 11 are 96.36% and 98.26%, respectively.

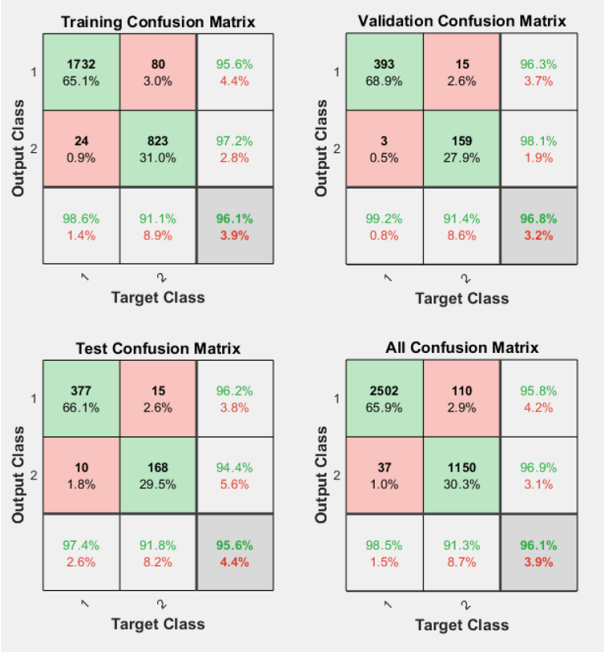


Figure 7.4: Confusion matrix of scenario 4

Table 7.5: Experimental result of FES-MALG-UNB

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 1732 | 24 | 823 | 80 | 91.141% | 1.367% | 94.057% | 96.089% |
| **Validate** | 393 | 3 | 159 | 15 | 91.379% | 0.758% | 94.643% | 96.842% |
| **Test** | 377 | 10 | 168 | 15 | 91.803% | 2.584% | 93.075% | 95.614% |
| **All** | 2502 | 37 | 1150 | 110 | 91.270% | 1.457% | 93.993% | 96.131% |

**Scenario 4: FES-MALG-UNB** In this scenario, we use unbalanced Malgenome dataset with FES method. The experiment result shows that this setup achieved 96.13% accuracy. The detection rate is

91.27%, the false alarm rate is 1.46%, and the F1 score is 93.99%. Theoretically, FES structure should give the worst performance compared to the DFES and mDFES. When we compared this scenario result with DFES (Scenario 8) and mDFES (Scenario 12), the performance result of this scenario is lower; the accuracy result of Scenario 8 and 12 are 96.58% and 97.92%, respectively.

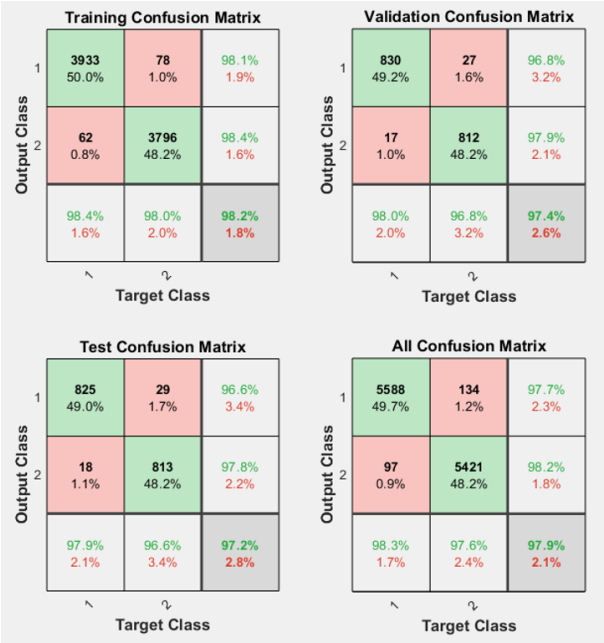


Figure 7.5: Confusion matrix of scenario 5

Table 7.6: Experimental result of DFES-DREB-BAL

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 3933 | 62 | 3796 | 78 | 97.987% | 1.552% | 98.189% | 98.221% |
| **Validate** | 830 | 17 | 812 | 27 | 96.782% | 2.007% | 97.362% | 97.390% |
| **Test** | 825 | 18 | 813 | 29 | 96.556% | 2.135% | 97.191% | 97.211% |
| **All** | 5588 | 97 | 5421 | 134 | 97.588% | 1.706% | 97.914% | 97.945% |

**Scenario 5: DFES-DREB-BAL** In this scenario, we use balanced Drebin dataset with DFES method. The experiment result shows that this setup achieved 97.95% accuracy. The detection rate is 97.59%, the false alarm rate is 1.71%, and the F1 score is 97.91%. When we compared this scenario result with DFES (Scenario 5) and mDFES (Scenario 9), the accuracy result of Scenario 1 and 9 are 92.02% and 95.66%, respectively. This shows that for balanced Drebin dataset, DFES structure gives the best performance compared to FES and mDFES.

**Scenario 6: DFES-DREB-UNB** In this scenario, we use unbalanced Drebin dataset with DFES method. The experiment result shows that this setup achieved 97.10% accuracy. The detection rate is 97.12%, the false alarm rate is 2.92%, and the F1 score is 97.07%. When we compared this scenario result with FES (Scenario 2) and mDFES (Scenario 10), the accuracy result of Scenario 2 and 10 are 92.86% and 96.33%, respectively. This shows that for unbalanced Drebin dataset, DFES structure gives the best performance compared to FES and mDFES.

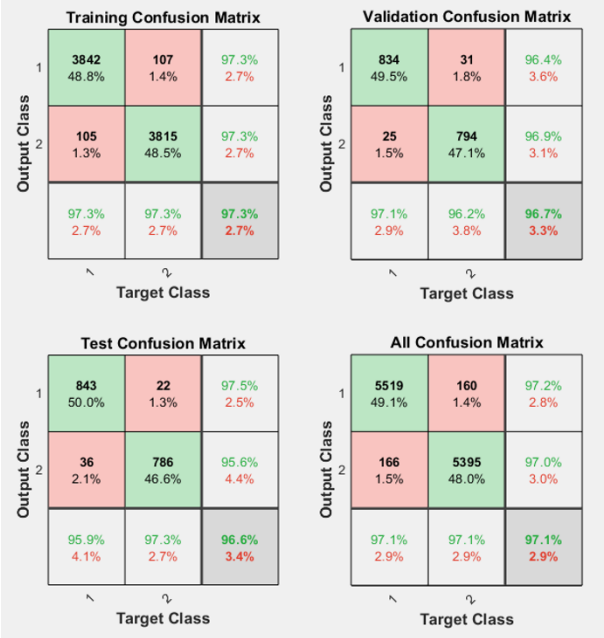


Figure 7.6: Confusion matrix of scenario 6

Table 7.7: Experimental result of DFES-DREB-UNB

| | **TN** | **FP** | **TP** | **FN** | **DR** | **FAR** | **F1** | **Acc** |
|---|---|---|---|---|---|---|---|---|
| **Train** | 3842 | 105 | 3815 | 107 | 97.272% | 2.660% | 97.297% | 97.306% |
| **Validate** | 834 | 25 | 794 | 31 | 96.242% | 2.910% | 96.594% | 96.675% |
| **Test** | 843 | 36 | 786 | 22 | 97.277% | 4.096% | 96.442% | 96.562% |
| **All** | 5519 | 166 | 5395 | 160 | 97.120% | 2.920% | 97.067% | 97.100% |

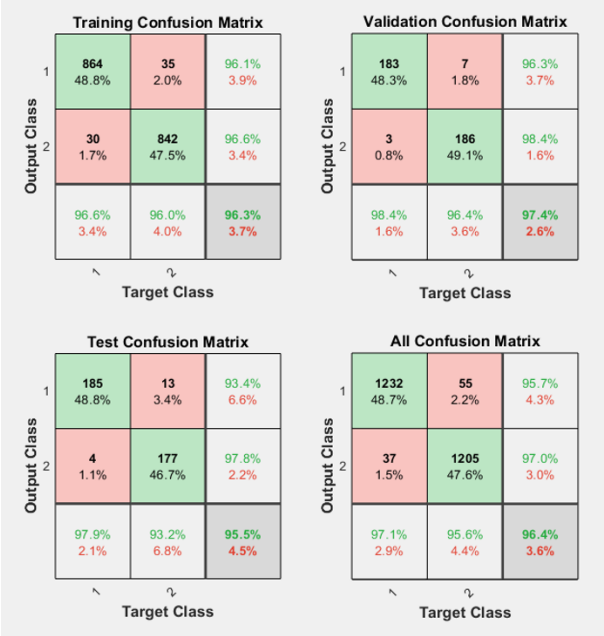


Figure 7.7: Confusion matrix of scenario 7

Table 7.8: Experimental result of DFES-MALG-BAL

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 864 | 30 | 842 | 35 | 96.009% | 3.356% | 96.284% | 96.330% |
| **Validate** | 183 | 3 | 186 | 7 | 96.373% | 1.613% | 97.382% | 97.361% |
| **Test** | 185 | 4 | 177 | 13 | 93.158% | 2.116% | 95.418% | 95.515% |
| **All** | 1232 | 37 | 1205 | 55 | 95.635% | 2.916% | 96.323% | 96.362% |

**Scenario 7: DFES-MALG-BAL** In this scenario, we use balanced Malgenome dataset with DFES method. The experiment result shows that this setup achieved 96.36% accuracy. The detection rate is 95.64%, the false alarm rate is 2.92%, and the F1 score is 96.32%. When we compared this scenario result with FES (Scenario 3) and mDFES (Scenario 11), the accuracy result of Scenario 3 and 11 are 95.98% and 98.26%, respectively. This shows that for balanced Malgenome dataset, DFES structure gives better performance compared to FES but lower performance compared to mDFES.

Table 7.9: Experimental result of DFES-MALG-UNB

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 1733 | 28 | 842 | 56 | 93.764% | 1.590% | 95.249% | 96.841% |
| **Validate** | 382 | 15 | 164 | 9 | 94.798% | 3.778% | 93.182% | 95.789% |
| **Test** | 370 | 11 | 178 | 11 | 94.180% | 2.887% | 94.180% | 96.140% |
| **All** | 2485 | 54 | 1184 | 76 | 93.968% | 2.127% | 94.796% | 96.578% |

**Scenario 8: DFES-MALG-UNB** In this scenario, we use unbalanced Malgenome dataset with DFES method. The experiment result shows that this setup achieved 96.58% accuracy. The detection rate is 93.97%, the false alarm rate is 2.13%, and the F1 score is 94.80%. When we compared this

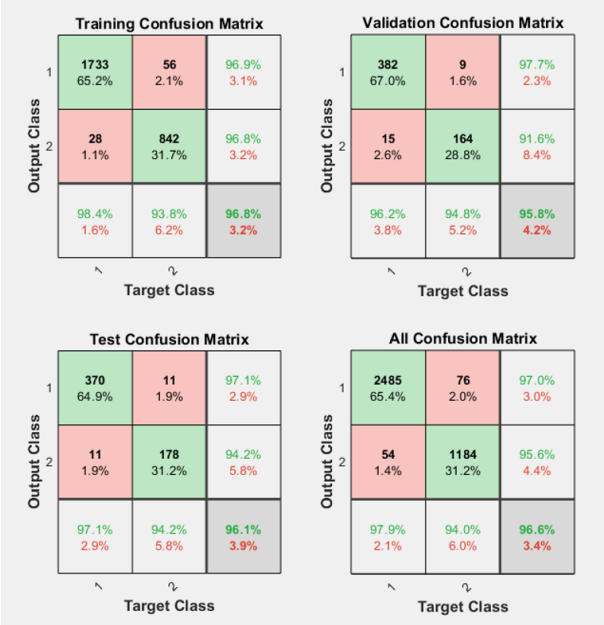


Figure 7.8: Confusion matrix of scenario 8

scenario result with FES (Scenario 4) and mDFES (Scenario 12), the accuracy result of Scenario 1 and 9 are 96.13% and 97.92%, respectively. This shows that for unbalanced Malgenome dataset, DFES structure gives better performance compared to FES but lower performance than mDFES.

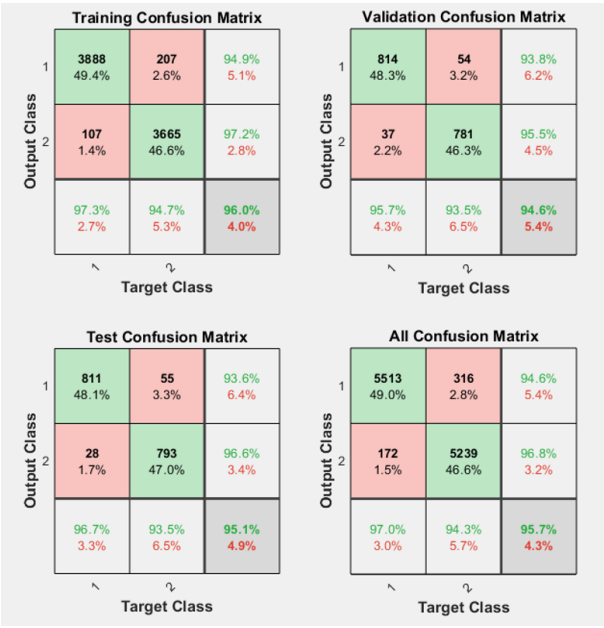


Figure 7.9: Confusion matrix of scenario 9

**Scenario 9: mDFES-DREB-BAL** In this scenario, we use balanced Drebin dataset with mDFES method. The experiment result shows that this setup achieved 95.66% accuracy. The detection rate is 94.31%, the false alarm rate is 3.03%, and the F1 score is 95.55%. When we compared this scenario result with FES (Scenario 1) and DFES (Scenario 5), the accuracy result of Scenario 1 and 5 are 92.02%

Table 7.10: Experimental result of mDFES-DREB-BAL

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 3888 | 107 | 3665 | 207 | 94.654% | 2.678% | 95.892% | 96.009% |
| **Validate** | 814 | 37 | 781 | 54 | 93.533% | 4.348% | 94.495% | 94.603% |
| **Test** | 811 | 28 | 793 | 55 | 93.514% | 3.337% | 95.027% | 95.080% |
| **All** | 5513 | 172 | 5239 | 316 | 94.311% | 3.026% | 95.550% | 95.658% |

and 97.95%, respectively. This shows that for balanced Drebin dataset, mDFES structure gives better performance compared to FES but lower performance than DFES.

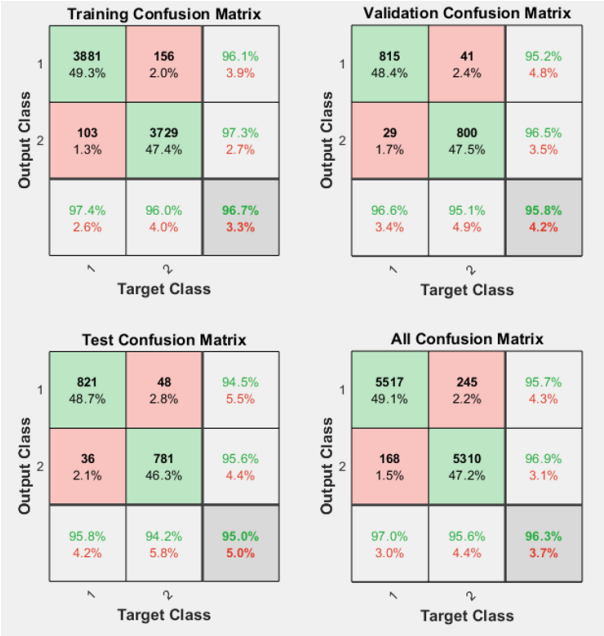


Figure 7.10: Confusion matrix of scenario 10

Table 7.11: Experimental result of mDFES-DREB-UNB

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 3881 | 103 | 3729 | 156 | 95.985% | 2.585% | 96.644% | 96.709% |
| **Validate** | 815 | 29 | 800 | 41 | 95.125% | 3.436% | 95.808% | 95.846% |
| **Test** | 821 | 36 | 781 | 48 | 94.210% | 4.201% | 94.897% | 95.018% |
| **All** | 5517 | 168 | 5310 | 245 | 95.590% | 2.955% | 96.257% | 96.326% |

**Scenario 10: mDFES-DREB-UNB** In this scenario, we use unbalanced Drebin dataset with mDFES method. The experiment result shows that this setup achieved 96.33% accuracy. The detection rate is 95.59%, the false alarm rate is 2.96%, and the F1 score is 96.26%. When we compared this scenario result with FES (Scenario 2) and DFES (Scenario 6), the accuracy result of Scenario 2 and 6 are 92.86% and 97.10%, respectively. This shows that for unbalanced Drebin dataset, mDFES structure gives better performance compared to FES but lower performance than DFES.

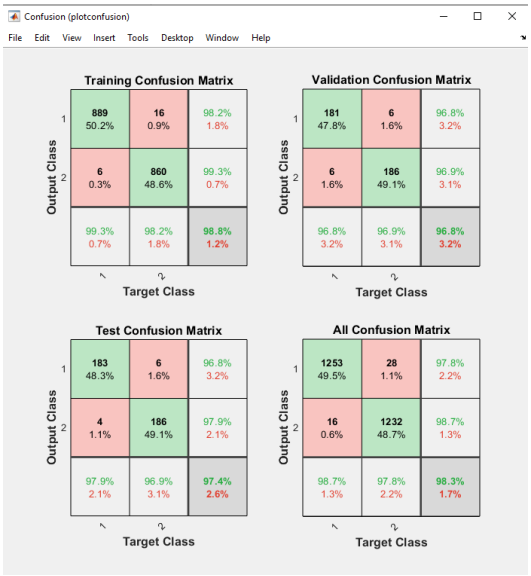


Figure 7.11: Confusion matrix of scenario 11

Table 7.12: Experimental result of mDFES-MALG-BAL

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 889 | 6 | 860 | 16 | 98.174% | 0.670% | 98.737% | 98.758% |
| **Validate** | 181 | 6 | 186 | 6 | 96.875% | 3.209% | 96.875% | 96.834% |
| **Test** | 183 | 4 | 186 | 6 | 97.382% | 2.139% | 97.382% | 97.361% |
| **All** | 1253 | 16 | 1232 | 28 | 98.246% | 1.261% | 98.246% | 98.260% |

**Scenario 11: mDFES-MALG-BAL** In this scenario, we use balanced Malgenome dataset with mDFES method. The experiment result shows that this setup achieved 98.26% accuracy. The detection rate is 97.78%, the false alarm rate is 1.26%, and the F1 score is 98.25%. When we compared this scenario result with FES (Scenario 3) and DFES (Scenario 7), the accuracy result of Scenario 1 and 5 are 95.98% and 96.36%, respectively. This shows that for balanced Malgenome dataset, mDFES structure gives the best performance compared to FES and DFES.

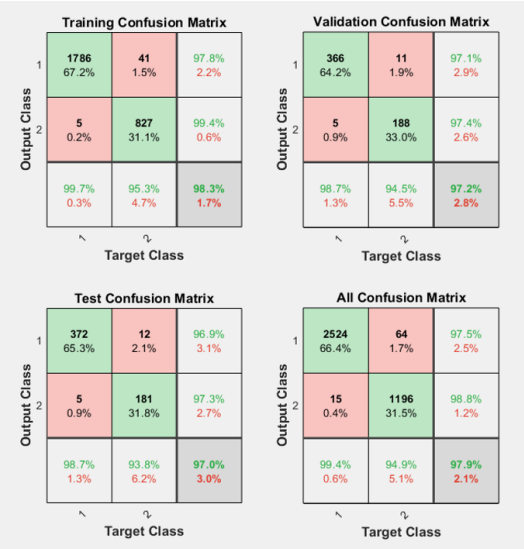


Figure 7.12: Confusion matrix of scenario 12

Table 7.13: Experimental result of mDFES-MALG-UNB

| | TN | FP | TP | FN | DR | FAR | F1 | Acc |
|---|---|---|---|---|---|---|---|---|
| **Train** | 1786 | 5 | 827 | 41 | 95.276% | 0.279% | 97.294% | 98.270% |
| **Validate** | 366 | 5 | 188 | 11 | 94.472% | 1.348% | 95.918% | 97.193% |
| **Test** | 372 | 5 | 181 | 12 | 93.782% | 1.326% | 95.515% | 97.018% |
| **All** | 2524 | 15 | 1196 | 64 | 94.921% | 0.591% | 96.803% | 97.921% |

**Scenario 12: mDFES-MALG-UNB** In this scenario, we use unbalanced Malgenome dataset with mDFES method. The experiment result shows that this setup achieved 97.92% accuracy. The detection rate is 94.92%, the false alarm rate is 0.59%, and the F1 score is 96.80%. When we compared this scenario result with FES (Scenario 4) and DFES (Scenario 8), the accuracy result of Scenario 4 and 8 are 96.13% and 96.58%, respectively. This shows that for unbalanced Malgenome dataset, mDFES structure gives the best performance compared to FES and DFES.

## 7.3 Comparison

Droid-Fusion [34] proposed a four-ranking based algorithm for classifying Android malware dataset. The classifier has stacked generalization with multilevel architecture. They used Drebin dataset and Malgenome dataset for the evaluation. Because it has multilevel architecture, Droid-Fusion can be

applied for singular classifier and ensemble classifier. Firstly, the model is trained using a N-fold cross-validation technique to estimate the prediction accuracies. Then, the output is used by four different ranking-based algorithms to determine the selected items. The results are combined in pairs to find the strongest pair that will be used to build the final model after testing against the unweighted parallel combination. The training phase is distinct from the prediction phase. Different validation and testing dataset have been prepared for the classification phase. For the feature extraction, python-based static analyzer is deployed to do automatic feature extraction. While for feature selection, Information Gain (IG) method is utilized. Droid-NNet [47] proposed an optimized neural network framework optimized with different parameters and tested with two real-world Android datasets: Malgenome and Drebin. The authors utilize random forest classifier and deep neural network with 2, 4, and 7 layers. They compare the performance of four experimental scenarios: support vector machine, decision tree, logistic regression, and neural network with various number of features. The dataset is split into training and testing dataset while maintaining the ratio between benign and malware instances. Droid-NNet leverages ReLu activation function in the hidden layer and sigmoid function in the output layer. For the optimizer, Adam optimizer is chosen; while for loss function, binary cross-entropy is implemented. In order to stop the training once the model's performance has stopped improving, a selected validation loss is deployed. Finally, in order to prevent overfitting, regularization technique is applied at the output of the hidden layer. All parameters are optimized my grid search. Deep-Droid [48] proposed a combination of feature extraction and sequential neural network model to detect Android malware using Drebin and Malgenome dataset. The sequential neural network consists of input layer, one or more hidden layer, and output layer. Nonlinear activation function is deployed to capture nonlinear data. During the training phase, the weight value is constantly updated by applying backpropagation algorithm. The input layer of Deep-Droid has 215 neurons, the hidden layer has 215 neurons, and the output layer has one neuron to do binary classification. Binary entropy is applied as a loss function and optimizer to estimate the adaptive movements, calculating the error and updating the weight. They compare the performance of Deep-Droid with several well-known approaches such as: random forest, support vector machine, and k-nearest neighbor. Deep-Droid successfully returned the best F-measure compared to those methods.

We will compare the performance of three works mentioned above. All those works proposed deep learning for Android malware detection using Drebin and Malgenome dataset. Droid-Fusion [34] achieved 98.40% F1 score with 0.07 seconds classification time for Malgenome dataset and 98.10% F1 score with 0.38 seconds classification time for Drebin dataset. Droid-NNet [47] achieved 99.26% F1 score for Malgenome dataset and 98.80% F1 score with for Drebin dataset. Deep-Droid [48] achieved 99.20% F1 score for Malgenome dataset and 98.70% F1 score for Drebin dataset. Table. 7.14 shows the performance comparison of Droid-Fusion, Droid-NNET, and Deep-Droid.

Table 7.14: The Performance Comparison of Droid-Fusion, Droid-NNET, and Deep-Droid

| 2*Method | F1 Score (%) | |
|---|---|---|
| | **Malgenome** | **Drebin** |
| **Droid-Fusion** | 98.40 | 98.10 |
| **Droid-NNet** | 99.26 | 98.80 |
| **Deep-Droid** | 99.20 | 98.70 |

Overall, in a similar setting, for balanced Drebin dataset; the best performance is given by DFES with 97.95% accuracy, 97.59% detection rate, 1.71% FAR, and 97.91% F1 score. For unbalanced Drebin

dataset, the best performance is given by DFES with 97.10% accuracy, 97.12% detection rate, 2.92% FAR, and 97.07% F1 score. For balanced Malgenome dataset, the best performance is given by mDFES with 96.40% accuracy, 95.71% DR, 2.92% FAR, and 96.36% F1 score. For unbalanced Malgenome dataset, the best performance is given by mDFES with 98.26% accuracy. 97.78% DR, 1.26% FAR, and 98.25% F1 score. The experimental result shows that all our hypothesis are correct. Among FES, DFES, and mDFES; FES is the fastest, but has worst accuracy. mDFES is faster than DFES. Finally, the type of dataset affects the performance of deep learning method, as shown that mDFES is better than DFES for Malgenome dataset, while DFES is better than mDFES for Drebin dataset.

# Chapter 8. Concluding Remarks

To summarize, we have proposed mDFES, a feature learning method for malware detection over Android. We did experiments on FES, DFES, and mDFES using 9 different scenarios to compare their performance. From our experiments, we can conclude that FES is the fastest method but has the worst accuracy. For Malgenome dataset, mDFES is the best performer. For Drebin dataset, DFES is the best performer. Using DFES for Android malware Dataset achieves 96-98% accuracy, while using DFES for AWID Dataset [37] (with same configuration) achieved 99% accuracy. So, different type of dataset requires different deep learning approach to achieve the best result.

# Chapter 9. Introduction

In a business environment, prediction and decision-making are two important processes that require careful consideration. Good judgement can lead to large profits, but bad decisions can ruin everything. There was a hypothesis that a computer could help a user predict something or decide what next step should be taken. As Artificial Intelligence (AI) has grown dramatically, this plan is no longer considered impossible. AI has the ability to sense, understand, learn, and respond [7]. This solves the weaknesses of computers without these four abilities. Prediction, on the other hand, is a process of learning available information and then using that knowledge to generate new information that is not yet available. A Deep Learning (DL) algorithm is a type of AI that has the ability to interpret data like a human brain and can learn and classify objects. By leveraging the ability of deep learning, we can predict the future and make decisions based on the currently available information, which becomes our training data when we train the DL model. After the training process is completed, a prediction model is produced. Based on this model, predictions based on clients' data will be performed. That is how Machine Learning as a Service (MLaaS), a promising business opportunity, was born.

MLaaS is a service, which usually runs on a cloud platform, with the purpose is to provide prediction service to clients by utilizing machine learning [5]. The service runs on a cloud environment so that clients do not need to build their own machine learning model to do a prediction [6]. However, there is a problem. To perform predictions, a model owner needs to receive data from clients. The data may consist of sensitive information. Thus, clients are reluctant to provide their data. On the other hand, a model owner will also be worried that an adversary could be disguised as a client to try to steal the model. Furthermore, there is an issue about the privacy of the prediction result and whether will it be safe from access by unauthorized parties. In this scenario, Privacy-Preserving Deep Learning (PPDL) is needed as a solution.

# Chapter 10. Classical Privacy-Preserving Method

We classify the classical PP method into three categories, as shown in Fig. 10.1. The three categories are group-based anonymity, cryptography method, and differential privacy.

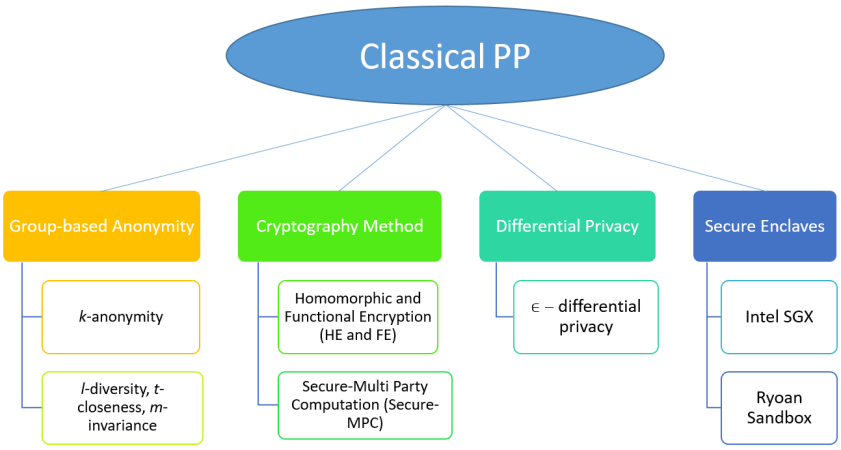


Figure 10.1: Classical PP Classification

## 10.1 Group Anonymity

While homomorphic encryption, functional encryption, and secure multi-party computation techniques enable computation on encrypted data without revealing the original plaintext, we need to preserve the privacy of sensitive personal data such as medical and health data. One of the earliest milestones to preserving this privacy is to hide these sensitive personal data using data anonymization techniques.

The concept of k-anonymity was first introduced by Sweeney and Samarati [49] in 1998 to solve the problem: "Given sensitive personal data, produce the modified data which remains useful while the data cannot specify the corresponding person." Modified data are said to have $k$-anonymity if the information for any person whose information is in the modified data cannot be distinguished from at least $k-1$ individuals in the modified data. While $k$-anonymity is a simple and promising approach for group-based anonymization, it is susceptible to attacks such as a homogeneity attack or background knowledge attack [50] when background knowledge is available to an attacker. To overcome these issues, there are many privacy definitions, such as $l$-diversity, $t$-closeness, and $m$-invariance [50, 51, 52]. The concept of $l$-diversity means that each equivalent class has at least $l$ distinct values for each sensitive attribute, and $t$-closeness is a further refinement of $l$-diversity created by also maintaining the distribution of sensitive attributes.

## 10.2 Cryptography Method

### 10.2.1 Homomorphic and Functional Encryption

In 1978, Rivest *et al.* [53] questioned whether any encryption scheme can support computation of encrypted data without knowledge of the encrypted information. If some encryption scheme supports an operation $\circ$ on encrypted data $Enc(m_1 \circ m_2)$, this scheme is called Homomorphic Encryption (HE) on an operation $\circ$. Depending on the computation type that HE supports, it is called partially HE when it supports the specific computation on encrypted data and Fully HE (FHE) when it supports arbitrary computation. For example, the well-known RSA encryption [54] supports multiplication on encrypted data without decryption, therefore RSA encryption is called multiplicative HE. Likewise, a scheme is additive HE if it supports addition on encrypted data without decryption.

The design of FHE remained as an interesting open problem in cryptography for decades, until Gentry suggested the first FHE in 2009 [55]. Afterwards, there have been a number of studies of HE schemes based on lattices with Learning With Errors (LWE) and Ring Learning With Errors (Ring-LWE) problems and schemes over integers with the approximate Greatest Common Divisor (GCD) problem [56, 57, 58, 59, 60, 61, 62, 63, 64, 65]. Earlier works focused on HE were impractical for implementation; however, there are currently many cryptographic algorithm tools that support HE efficiently, such as HElib, FHEW, and HEEAN [66, 67, 68].

Functional Encryption (FE) was proposed by Sahai and Waters [69] in 2005 and formalized by Boneh *et al.* [70] in 2011. Let a functionality $F : K \times X \rightarrow \{0,1\}^*$. The functionality $F$ is a deterministic function over $(K, X)$ that outputs $(0,1)^*$ where $K$ is the key space and the set $X$ is the plaintext space. We say a scheme is FE for a functionality $F$ over $(K, X)$ if it can calculate $F(k, x)$ given a ciphertext of $x \in X$ and a secret key $sk_k$ for $k \in K$.

Predicate encryption [71] is a subclass of FE scheme with a polynomial-time predicate $P : K \times I \rightarrow \{0,1\}$ where $K$ is the key space, $I$ is the index set, and the plaintext $x \in X$ is defined as $(\mathsf{ind}, m)$; $X$ is the plaintext space, $\mathsf{ind}$ is an index, and $m$ is the payload message. As an example, we can define FE functionality $F_{\mathsf{FE}}(k \in K, (\mathsf{ind}, m) \in X) = m$ or $\perp$ depending on whether the predicate $P(k, \mathsf{ind})$ is 1 or 0, respectively. Depending on the choice of the predicate, Identity-based Encryption (IBE) [72, 73, 74, 75, 76, 77, 78] and Attribute-based Encryption (ABE) [69, 79] are well-known examples of predicate encryption schemes.

Both FE and HE enable computation over encrypted data. The difference is that the computation output of FE is a plaintext, while the output of HE remains encrypted as HE evaluates the encrypted data without decryption. There is no need for a trusted authority within HE systems. Additionally, HE enables the evaluation of any circuit g over the encrypted data if $sk_g$ is given, but FE enables the computation of only some functions.

### 10.2.2 Secure Multi-party Computation

The purpose of Multi-party Computation (MPC) is to solve the problem of collaborative computing that keeps the privacy of an honest/dishonest user in a group without using any trusted third party. Formally, in MPC, for a given number of participants, $p_1, p_2, \cdots, p_n$, each participant has private data, $d_1, d_2, \cdots, d_n$, respectively. Then, participants want to compute the value of a public function $f$ on those private data, $f(d_1, d_2, \cdots, d_n)$, while keeping their own inputs secret.

The concept of secure computation was formally introduced as secure two-party computation in 1986

by Yao [80] with the invention of Garbled Circuit (GC). Yao's GC requires only a constant number of communication rounds, and all functions are described as a Boolean circuit. To transfer the information obliviously, Oblivious Transfer (OT) is used. The OT protocol allows a receiver $P_R$ to obliviously select $i$ and receive a message $m_i$ from a set of messages $M$ that belong to a sender party $P_S$. $P_R$ does not know the other messages in $M$ while $P_S$ does not know the selected message.

Secret sharing is yet another building block for secure MPC protocols, *e.g.*, Goldreich *et al.* [81] suggested a simple and interactive secure MPC protocol using the secret-shared values to compute the value. Secret sharing is a cryptographic algorithm where a secret is parted and distributed to each participant. To reconstruct the original value, a minimum number of secret-shared values are required.

Compared with HE and FE schemes, in secure MPC, parties jointly compute a function on their inputs using a protocol instead of a single party. During the process, information about parties' secret must not be leaked. In secure MPC, each party has almost no computational cost with a huge communication cost, whereas the server has a huge computational cost with almost no communication cost in the HE scheme. The parties encrypt their data and send them to the server. The server computes the inner product between the data and the weight value of the first layer and sends the computation result back to the parties. Then, the parties decrypt the results and compute the non-linear transformation. The result is encrypted and transmitted again to the server. This process continues until the last layer has been computed. To apply secure MPC to deep learning, we must handle the communication cost as it requires many rounds of communication between the parties and the server, which is non-negligible.

## 10.3 Differential Privacy

Differential privacy (DP) was first proposed by Dwork *et al.* [82] in 2006 as a strong standard to guarantee the privacy of the data. A randomized algorithm $A$ gives $\epsilon$-differential privacy if for all datasets $D_1$ and $D_2$ differ in at most one element, and for all subsets $S \in Range(imA)$, where $imA$ denotes the image of $A$, such that

$$\Pr[\mathcal{A}(D_1) \in S] \leq \exp(\epsilon) \cdot \Pr[\mathcal{A}(D_2) \in S].$$

Differential privacy addresses when a trusted data manager wants to release some statistics on the data while the adversary cannot reveal whether some individual's information is used in the computation. Thus, differentially private algorithms probably resist identification and reidentification attacks.

An example of the latest implementation of differential privacy technique was proposed by Qi *et al.* [83]. They suggested a privacy-preserving method for a recommender system using Locality-Sensitive Hashing (LSH) technique, which is more likely to assign two neighboring points to the same label. As a result, sensitive data can be converted into less sensitive ones.

## 10.4 Secure Enclaves

Secure enclaves, also known as Trusted Execution Environments (TEEs), are a secure hardware method that provides enclaves to protect code and data from another software on the related platform, including the operating system and hypervisor [84]. The concept of an enclave was firstly introduced by Intel [85], which introduced Software Guard Extensions (SGX), available on Intel processors starting from the Skylake generation [86]. Utilizing only SGX for privacy-preserving is not sufficient from the security and privacy perspectives because the code from the MLaaS provider is not trusted. SGX only protects the execution of trusted code on an untrusted platform. A code is called trusted if it is public

and users can inspect the code. If the code is private, users cannot be assured that the code does not steal their data. Because of this, SGX needs to be confined to a sandbox to prevent data exfiltration. The most widely used sandbox for SGX is Ryoan [87]. The Ryoan sandbox also enables users to verify that the enclave executes standard ML code without seeing the model specifications. As a result, a combination of the SGX and Ryoan sandboxes can guarantee the privacy of both clients and ML models.

# Chapter 11. Deep Learning for Privacy-Preserving

PPDL is a development from the classical DL method. It combines the classical PP method with the emerging DL field. DL itself is a sub-class of machine learning the structure and functionality of that resemble a human brain. The structure of a deep learning model is modelled like a layered architecture. It starts from an input layer and ends with an output layer. Between an input layer and an output layer, there can be one or more hidden layers. The more hidden layers are used, the more accurate the DL model becomes. This is caused by the characteristic of a hidden layer. The output of one hidden layer will become the input of the next hidden layer. If we use more hidden layers, the deeper hidden layer will learn about more specific features. There are several DL methods that are widely used for PP. Based on our research, the most popular DL methods for PP are the Deep Neural Network (DNN), Convolutional Neural Network (CNN), and Generative Adversarial Network (GAN).

## 11.1 Deep Neural Network

There are several commonly used layers in DNN, including the activation layer, pooling layer, fully connected layer, and dropout layer.

### 11.1.1 Activation Layer

The activation layer, as shown in Fig. 11.1, decides whether the data is activated (value one) or not (value zero). The activation layer is a non-linear function that applies a mathematical process on the output of a convolutional layer. There are several well-known activation functions, such as Rectified Linear Unit (ReLU), sigmoid, and tanh. Because those functions are not linear, the complexity becomes really high if we use the functions to compute the homomorphically encrypted data. Hence, we need to find a replacement function that only contains multiplication and addition operations. The replacement function will be discussed later.


Figure 11.1: Activation Layer

### 11.1.2 Pooling Layer

A pooling layer, as shown in Fig. 11.2, is a sampling layer with the purpose of reducing the size of the data. There are two kinds of pooling: max and average pooling. In HE, we cannot use a max pooling function because we cannot search for the maximum value of encrypted data. As a result, average pooling is the solution to be implemented in HE. Average pooling calculates the sum of values; thus, there is only the addition operation here, which can be used over homomorphically encrypted data.

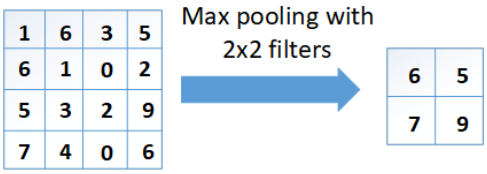


Figure 11.2: Pooling Layer

### 11.1.3 Fully Connected Layer

An illustration of a fully connected layer is shown in Fig. 11.3. Each neuron in this layer is connected to a neuron in the previous layer; thus, it is called a fully connected layer. The connection represents the weight of the feature like a complete binary graph. The operation in this layer is the dot product between the value of the output neuron from the previous layer and the weight of the neuron. This function is similar to a hidden layer in a Neural Network (NN). There is only a dot product function that consists of multiplication and addition function; thus, we can use it over homomorphically encrypted data.

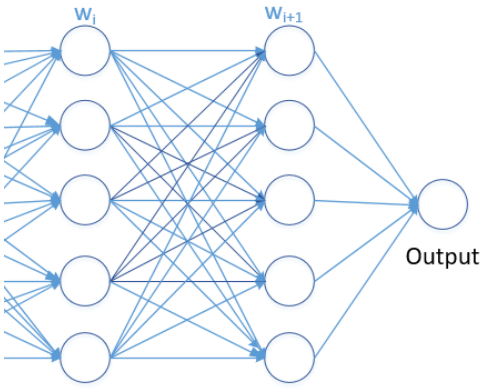


Figure 11.3: Fully Connected Layer

### 11.1.4 Dropout Layer

A dropout layer, shown in Fig. 11.4, is a layer created to solve an over-fitting problem. Sometimes, when we train our machine learning model, the classification result will be too good for some kind of data, showing bias based on the training set. This situation is not ideal, resulting in a large error during the testing period. The dropout layer will drop random data during training and set the data to zero. By doing this iteratively during the training period, we can prevent over-fitting during the training phase.

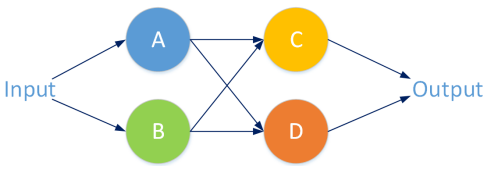


Figure 11.4: Dropout Layer

## 11.2 Convolutional Neural Network

CNN [88] is a class of DNN usually used for image classification. The characteristic of CNN is a convolutional layer, as shown in Fig. 11.5, the purpose of which is to learn features extracted from the dataset. The convolutional layer has $n \times n$ size, on which we will perform a dot product between neighboring values to make a convolution. As a result, only addition and multiplication occurs in the convolutional layer. We do not need to modify this layer as it can be used for HE data, which are homomorphically encrypted. Table 11.1 shows the commonly used layers in DNN and CNN models.

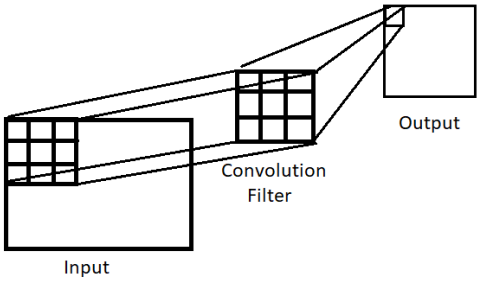


Figure 11.5: Convolutional Layer

## 11.3 Generative Adversarial Network

GAN [89] is a class of DNN usually used for unsupervised learning. GAN, as shown in Fig. 11.6, consists of two NNs that generate a candidate model and an evaluation model in a zero-sum game framework. The generative model will learn samples from a dataset until it reaches a certain accuracy. On the other hand, the evaluation model discriminates between the true data and the generated candidate model. GAN learns the process by modeling the distribution of individual classes.

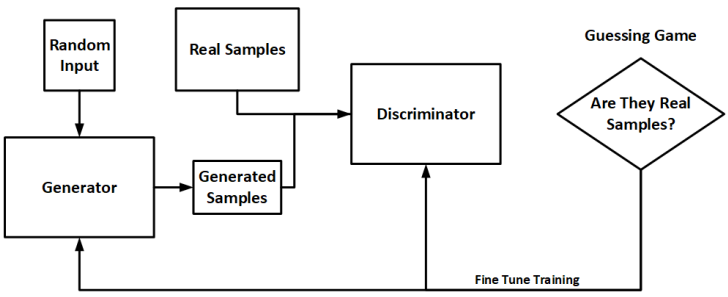


Figure 11.6: GAN Structure

## 11.4 Limitation of Implementing Deep Learning for Privacy-Preserving

During our studies, we found some incompatibilities between DL structures and classical PP techniques. Modifications had to be made to combine the DL structures and PP techniques. We cover the three most widely required modifications, as shown in Fig. 11.7, including the batch normalization layer, an approximation of activation function, and the convolutional layer with increased stride.

Table 11.1: Commonly Used Layers in DNN and CNN models

| **Deep Learning Layer** | | **Description** | **Function** |
|---|---|---|---|
| Activation Function | | ReLu | Maximum |
| | | Sigmoid | Hyperbolic |
| | | Tanh | Trigonometric |
| | | Softmax | Hyperbolic |
| 2Pooling | Max Pooling | Computing the maximum value of overlapping region in the preceding layer | Maximum |
| | Mean Pooling | Computing the average value of non-overlapping region in the preceding layer | Mean |
| Fully Connected | | Dot product between the value of output neuron from the previous layer and the weight of the neuron | Matrix-vector multiplication |
| Dropout | | Set random data to zero during training to prevent overfitting | Drop |
| Convolutional | | Dot product between neighbor values in order to make convolution, then sum up the result | Weighted Sum |

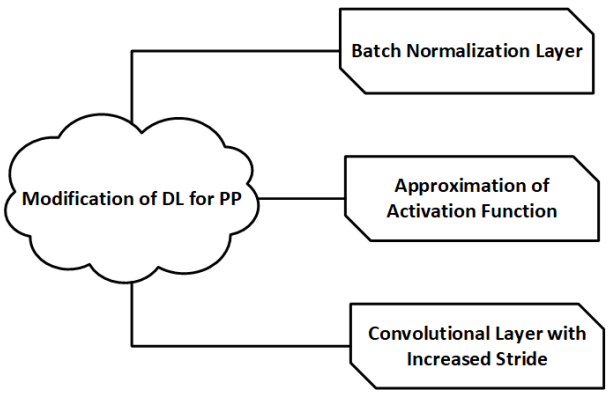


Figure 11.7: Required Modification for PPDL

### 11.4.1 Batch Normalization Layer

The Batch Normalization (BN) layer was proposed by Ioffe and Szegedy [90]. The main purpose of the BN layer is to accelerate the training process by increasing the stability of the NN. This layer receives the output from the activation layer and then performs the re-scaling process, resulting in a value between zero and one. The BN layer computes the subtraction of each input with the batch mean value, and then divides it by the average value of the batch.

### 11.4.2 Approximation of Activation Function

Several studies [91, 92, 93] have performed polynomial approximations for the activation function. Some well-known methods include numerical analysis, Taylor series, and establishing polynomials based on the derivative of the activation function. Numerical analysis generates some points from the ReLU function and then uses the points as the inputs of the approximation function. The Taylor series uses polynomials of different degrees to approximate the activation function.

### 11.4.3 Convolutional Layer with Increased Stride

This architecture was proposed by Liu *et al.* [93] to replace the pooling layer. The architecture leverages a convolutional layer with increased stride as a substitution of the pooling layer. The BN layer is used between the fully connected layer and ReLU. By doing this, the depth of the data stays the same, but the dimension is reduced [90].

# Chapter 12. State-of-the-Art PPDL Methods

As shown in Fig. 12.1, we classified each PPDL method by its privacy-preserving techniques: HE-based PPDL, secure MPC-based PPDL, differential privacy-based PPDL, secure enclaves-based PPDL, and hybrid-based PPDL. Hybrid-based PPDL means that the PPDL method combines more than one privacy-preserving technique mentioned before.

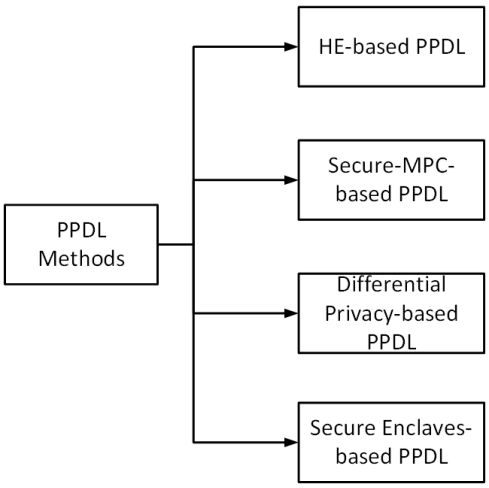


Figure 12.1: Classification of PPDL Methods by Its Privacy Preserving Techniques

We have surveyed several key publications on PPDL per each year since 2016 as shown in Fig. 12.2.

## 12.1 HE-based PPDL

HE-based PPDL combines homomorphic encryption with deep learning. The structure of HE-based PPDL is shown in Fig. 12.3. Generally, there are three phases in HE-based PPDL: the training phase (T1-T2-T3-T4), inference phase (I1-I2-I3)), and result phase (R1-R2-R3). In the training phase, a client encrypts the training dataset using HE (T1) and sends the encrypted dataset to the cloud server (T2). In the cloud server, secure training is executed (T3), resulting in a trained model (T4). This is the end of the training phase. For the inference phase, the client sends the testing dataset to the cloud server (I1). The testing dataset becomes the input of the trained model (I2). Then, the prediction process is run using the trained model (I3), resulting in an encrypted computation result. This is the end of the inference phase. Next, the cloud server prepares to transport the encrypted computation result (R1) and sends it to the client (R2). The client finally decrypts it and obtains its computation result (R3).

Cryptonets was proposed by Gilad-Bachrach et al. *et al.* [94] to address the privacy issue in Machine Learning as a Service (MLaaS). The author combined cryptography and machine learning to present a machine learning framework that can receive encrypted data as an input. Cryptonets improves the performance of ML Confidential [95] developed by Graepel *et al.*, a modified PPDL scheme based on Linear Means Classifier [96] and Fisher Linear Discriminant [97] that works on HE. ML Confidential uses polynomial approximation to substitute for the nonlinear activation function. In this case, the PoM is not guaranteed because the client must generate the encryption parameter based on the model. ML

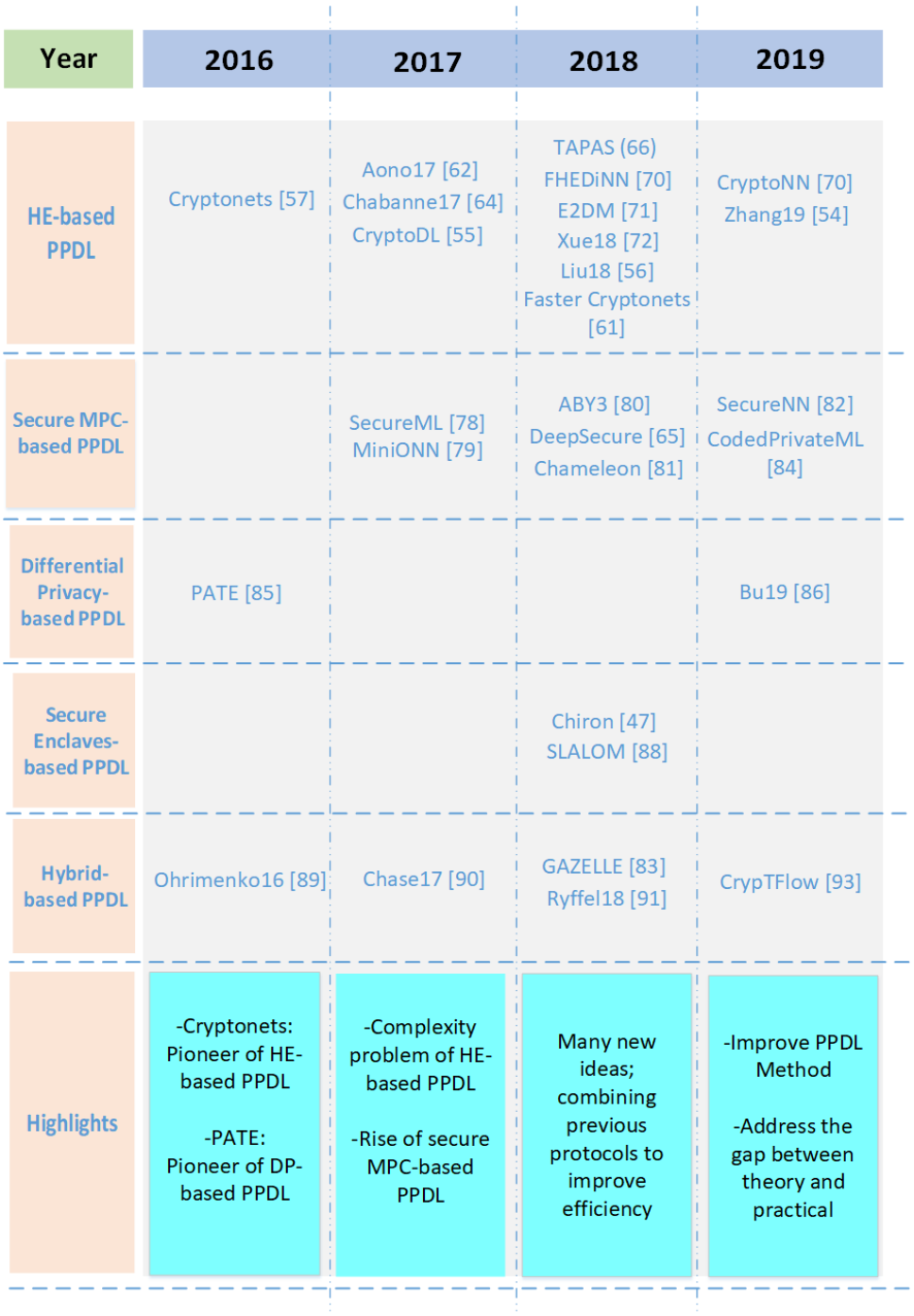


Figure 12.2: The Surveyed Paper of PPDL Since 2016

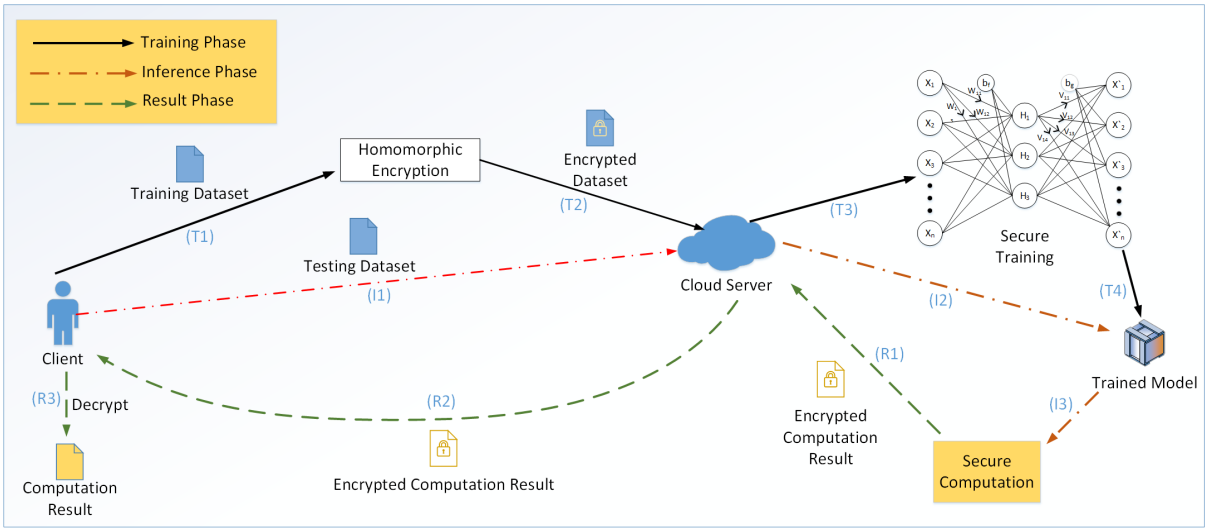


Figure 12.3: The Structure of HE-based PPDL

Confidential uses a cloud service-based scenario, and its main feature is ensuring the privacy of data during the transfer period between the client and the server. At first, the cloud server produces a public key and its private key for each client. Then, the client data are encrypted using HE and transferred to the server. The cloud server will perform the training process using the encrypted data and use the training model to perform classification on the testing dataset.

Cryptonets applies prediction based on encrypted data and then provides the prediction result, also in encrypted form, to users. Later, users can use their private key to decrypt the prediction result. By doing this, the privacy of the client and the privacy of the result are guaranteed. However, the privacy of model is not guaranteed because the client must generate an encryption parameter based on the model. The weakness of Cryptonets is the performance limitation because of the complexity issue. It does not work well on deeper NNs that have a large number of non-linear layers. In this case, the accuracy will decrease and the error rate will increase.

Cryptonets has trade-off between accuracy and privacy. This is caused by the utilization of activation function approximation using low-degree polynomial during the training phase. The neural network needs to be retrained again using plaintext with the same activation function in order to achieve good accuracy. Another weakness of Cryptonets is the limited number of neural network layer. The multiplicative leveled HE cannot be run on deep neural network with many layers. Faster Cryptonets [98] accelerates homomorphic evaluation in Cryptonets [94] by pruning network parameter such that many multiplication operations can be omitted. The main weakness of Faster Cryptonets is that it has vulnerability to membership inference attack [99] and model stealing attack [100].

Aono17 [101] is a PPDL system based on a simple NN structure. The author shows a weakness in the paper by Shokri and Shmatikov [102] that leaks client data during the training process. The weakness is called Gradients Leak Information. It is an adversarial method for obtaining input values by calculating the gradient of the corresponding truth function to weight and the gradient of the corresponding of truth function to bias. If we divide the two results, we obtain the input value. Because of that reason, Aono17 [101] proposes a revised PPDL method to overcome this weakness. The key idea is allowing the cloud server to update the deep learning model by accumulating gradient values from users. The author also

utilized additively HE to protect gradient values against curious servers. However, a weakness actually remains in this approach because it does not prevent attacks between participants. Proper authentication of participants should be performed by the cloud server to prevent this vulnerability. This method is able to prevent data leakage by encrypting the gradient value. However, it has some limitations as the homomorphic encryption is compatible with parameter server only.

Chabanne17 [103] is a privacy-preserving scheme on DNN. The scheme is a combination of HE and CNN. The main idea is to combine Cryptonets [94] with polynominal approximation for the activation function and batch normalization layer proposed by Ioffe and Szegedy [90]. The scheme wants to improve the performance of Cryptonets, which is only good when the number of non-linear layers in the model is small. The main idea is to change the structure of the regular NN by adding a batch normalization layer between the pooling layer and activation layer. Max pooling is not a linear function. As a result, in pooling layers average pooling is used instead of max pooling to provide the homomorphic part with a linear function. The batch normalization layer contributes to restricting the input of each activation layer, resulting in a stable distribution. Polynomial approximation with a low degree gives a small error, which is very suitable for use in this model. The training phase is performed using the regular activation function, and the testing phase is performed using the polynomial approximation as a substitution to non-linear activation function. Chabanne17 showed that their model achieved 99.30% accuracy, which is better than that of Cryptonets (98.95%). The pros of this model is its ability to work in a NN with a high number of non-linear layers while still providing higher than 99% accuracy, unlike Cryptonets which exhibits a decrease in accuracy when the number of non-linear layers is increased. Chabanne17's weakness is that the classification accuracy relies on the approximation of activation function. If the approximation function has high degree, it will be hard to get best approximation so that the accuracy will decrease.

CryptoDL [92], proposed by Hesamifard *et al.*, is a modified CNN for encrypted data. The activation function part of CNN is substituted with a low-degree polynomial. That paper showed that the polynomial approximation is indispensable for NN in HE environments. The authors tried to approximate three kinds of activation functions: ReLU, sigmoid, and tanh. The approximation technique is based on the derivative of the activation function. First, during the training phase, CNN with polynomial approximation is used. Then, the model produced during the training phase is used to perform classification over encrypted data. The authors applied the CryptoDL scheme to the MNIST dataset and achieved 99.52% accuracy. The weakness of this scheme is not covering privacy-preserving training in DNN. The privacy-preserving is only applied for the classification process. The advantage of this work is that it can classify many instances (8,192 or larger) for each prediction round, whereas DeepSecure [104] classifies one instance per round. Hence, we can say that CryptoDL works more effectively than DeepSecure. The weakness of CryptoDL is claimed to be the limited number of layers in DNN. Since as the number of layer increases, the complexity is also increased multiplicatively due to HE operations, reducing its performance like Cryptonets [94].

In TAPAS [105], the author addresses the weakness of Fully Homomorphic Encryption in PPDL, which requires a large amount of time to evaluate deep learning models for encrypted data [64]. The author developed a deep learning architecture that consists of a fully-connected layer, a convolutional layer, and a batch normalized layer [90] with sparsified encrypted computation to reduce the computation time. The main contribution here is a new algorithm to accelerate binary computation in the binary neural network [106], [107]. Another superiority of TAPAS is supporting parallel computing. The technique can be parallelized by evaluating gates in the same level at the same time. A serious limitation

of TAPAS is that it only supports binary neural network. In order to overcome this limitation, a method to encrypt non-binary or real-valued neural network is required.

FHE DiNN [108] is a PPDL framework that combines FHE with a discretized neural network. It addresses the complexity problem of HE in PPDL. FHE-DiNN offers a NN with linear complexity with regard to the depth of the network. In other words, FHE-DiNN has the scale invariance property. Linearity is achieved by the bootstrapping procedure on a discretized NN with a weighted sum and a sign activation function that has a value between -1 and 1. The sign activation function will maintain linearity growth such that it will not be out of control. The computation of the activation function will be performed during the bootstrapping procedure to refresh the ciphertext, reducing its cumulative noise. When we compare the discretized neural network to a standard NN, there is one main difference: the weight, the bias value, and the domain of the activation function in FHE DiNN needs to be discretized. The sign activation function is used to limit the growth of the signal in the range of -1 and 1, showing its characteristic of linear scale invariance for linear complexity. Compared with Cryptonets [94], FHE DiNN successfully improves the speed and reduces the complexity of FHE but with a decrease in accuracy; thus, a trade-off exists. The weakness of this method happens in the discretization process, which uses sign activation function that leads to a decrease in accuracy. It gets better if the training process is directly executed in a discretized neural network, rather than by converting a regular network into a discretized one.

E2DM [109] converts an image dataset into matrices. The main purpose of doing this is to reduce the computational complexity. E2DM shows how to encrypt multiple matrices into a single ciphertext. It extends some basic matrix operations such as rectangular multiplication and transposition for advanced operations. Not only is the data encrypted; the model is also homomorphically encrypted. As a result, PoC and PoM are guaranteed. E2DM also fulfills the PoR as only the client can decrypt the prediction result. For the deep learning part, E2DM utilizes CNN with one convolutional layer, two fully connected layers, and a square activation function. The weakness of E2DM is that it can only support simple matrix operation. Extending the advanced matrix computation will be a promising future work.

Xue18 [110] tries to enhance the scalability of the current PPDL method. A PPDL framework with multi-key HE was proposed. Its main purpose [110] was to provide a service to classify large-scale distributed data. For example, in the case of predicting road conditions, the NN model must be trained from traffic information data from many drivers. For the deep learning structure, [110] modification to the conventional CNN architecture is necessary, such as changing max pooling into average pooling, adding a batch normalization layer before each activation function layer, and replacing The ReLU activation function with a low-degree approximation polynomial. PoC and PoR are guaranteed here. However, the privacy of the model is not guaranteed because the client must generate an encryption parameter based on the model. The weakness of this approach is that the neural network must be trained by using encrypted data during the training phase. So, privacy leakage may happen if appropriate countermeasure is not deployed.

Liu18 [93] is a privacy-preserving technique for convolutional networks using HE. The technique uses an MNIST dataset that contains handwritten numbers. Liu18 [93] encrypts the data using HE and then uses the encrypted data to train CNN. Later, the classification and testing process is performed using the model from CNN. The idea is to add a batch normalization layer before each activation layer and approximate activation layer using Gaussian distribution and the Taylor series. The non-linear pooling layer is substituted for with the convolutional layer with increased stride. By doing this, the author successfully modified CNN to be compatible with HE, achieving 98.97% accuracy during the testing

phase. The main difference between regular CNN and modified CNN in privacy-preserving technology is the addition of the batch normalization layer and the change of the non-linear function in the activation layer and the pooling layer into a linear function. The proposed approach has weakness from the point of complexity since the HE has massive computational overhead leading to huge memory overhead.

CryptoNN [111] is a privacy-preserving method that utilizes functional encryption for arithmetic computation over encrypted data. The FE scheme protects the data in the shape of a feature vector inside matrices. By doing this, the matrix computation for NN training can be performed in encrypted form. The training phase of CryptoNN comprises two main steps: a secure feed-forward step and a secure back-propagation step. The CNN model is adapted with five main functions: a dot-product function, weighted-sum function, pooling function, activation function, and cost function. During the feed-forward phase, the multiplication of the weight value and feature vector cannot be performed directly because the vector value is encrypted. As a result, a function-derived key is used to transform the weight value such that it can computed. However, the scalability of CryptoNN is still in question since the dataset used in their experiment is a simple one. It needs to be tested with more complex dataset and deeper neural network model.

Zhang19 [91] is a secure clustering method for preserving data privacy in cloud computing. The method combines a probabilistic C-Means algorithm [112] with a BGV encryption scheme [12] to produce HE-based big data clustering on a cloud environment. The main reason for choosing BGV in this scheme is its ability to ensure a correct result on the computation of encrypted data. The author also addresses the weakness of the probabilistic C-Means algorithm, which is very sensitive and needs to be initialized properly. To solve this problem, fuzzy clustering [113] and probabilistic clustering [114] are combined. During the training process, there are two main steps: calculating the weight value and updating the matrix. To this end, a Taylor approximation for the activation function is used as the function is polynomial with addition and multiplication operations only. The main weakness is that the computation cost will increase proportionally to the number of neural network layers due to characteristic of HE.

According to Boulemtafes *et al.* [115], based on its learning method, PPDL techniques can be classified into two kinds; server-based and server-assisted. Server-based means that the learning process is executed on the cloud server. On the other hand, server-assisted means that the learning process is performed collaboratively by the parties and the server. Table 12.1 shows the features of our surveyed HE-based PPDL.

## 12.2 Secure MPC-based PPDL

Generally, the structure of a secure MPC-based PPDL is shown in Fig. 12.4. Firstly, users perform local training using their private data (1). Then, the gradient result from the training process is secret-shared (2). The shared gradient is transmitted to each server (3). After that, the server aggregates the shared gradient value from users (4). The aggregated gradient value is transmitted from each server to each client (5). Each client reconstructs the aggregated gradient and updates the gradient value for the next training process (6). In the case of multi-party computation, secret sharing is used to preserve the data privacy. However, for specific secure two-party computation, a garbled circuit with secret sharing is widely used instead of secret sharing.

The structure of secure two-party computation is shown in Fig. 12.5. In secure two-party computation, a client uses garbled circuit to protect the data privacy. The communication between the client and the server is securely guaranteed by using oblivious transfer. At first, a client sends the private data

Table 12.1: Features of Our Surveyed HE-based PPDL

| **References** | **Key Concept** | **Learning Type** | **Dataset** |
|---|---|---|---|
| Cryptonets [94] | Enables cloud-based NN training using polynomial approximation of activation function | Server-based | MNIST |
| Aono17 [101] | Enables collaborative learning between participants over combined datasets | Server-assisted | MNIST |
| Chabanne17 [103] | Applies FHE with low degree approximation of activation function | Server-based | MNIST |
| CryptoDL [92] | Applies leveled HE with approximation of activation function based on the derivative of the function | Server-based | MNIST CIFAR-10 |
| TAPAS [105] | Proposes a sparsified encrypted computation to speed up the computation in binary neural network | Server-based | MNIST |
| FHEDiNN [108] | Applies FHE in discretized neural network with linear complexity, in regards to the network's depth | Server-based | MNIST |
| E2DM [109] | Proposes multiple matrices encryption into a single chipertext to reduce computational complexity | Server-based | MNIST |
| Xue18 [110] | Applies multi-key HE with batch normalization layer before each activation function layer | Server-based | MNIST |
| Liu18 [93] | Proposes the addition of batch normalization layer and approximates activation function using Gaussian distribution and Taylor series | Server-based | MNIST |
| Faster Cryptonets [98] | Accelerates homomorphic evaluation by pruning the network parameters | Server-based | MNIST |
| CryptoNN [111] | Utilizes FE for arithmetic computation over encrypted data | Server-based | MNIST |

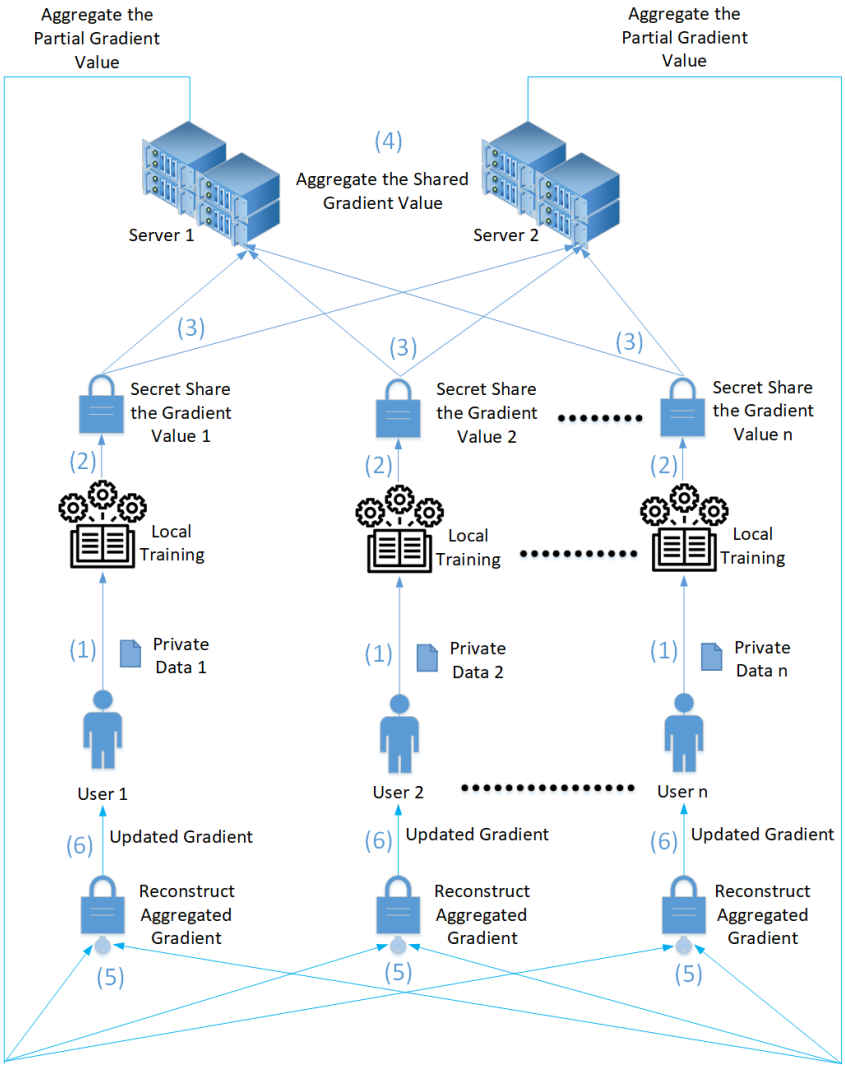


Figure 12.4: The Structure of Secure MPC-based PPDL

input to the garbled circuit for the garbling process (1). Then, the next process is the data exchange between the client and the server using oblivious transfer (2). After the data exchange has been completed, the server runs the prediction process, using the data as an input in the deep learning model (3). The prediction result is sent back to the client. The client uses the garbled table to aggregate the result (4) and obtain the final output (5).

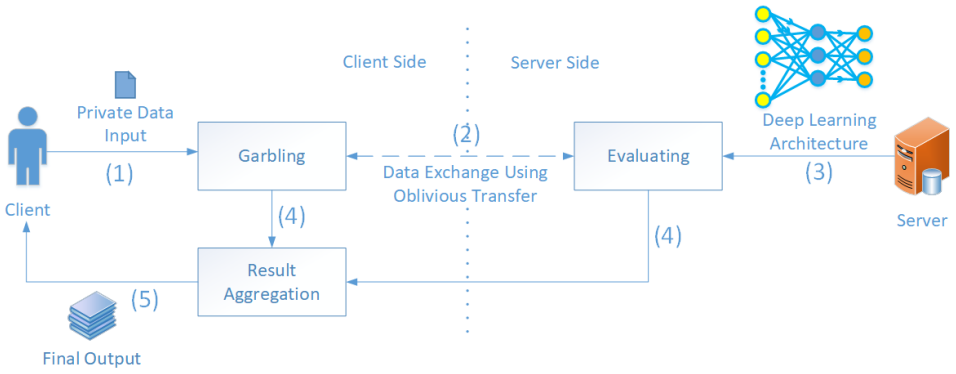


Figure 12.5: The Structure of Secure Two-party Computation-based PPDL

SecureML [116] is a new protocol for privacy-preserving machine learning. The protocol uses Oblivious Transfer (OT), Yao's GC, and Secret Sharing to ensure the privacy of the system. For the deep learning part, it leverages linear regression and logistic regression in a DNN environment. The protocol proposes an addition and multiplication algorithm for secretly shared values in the linear regression. The Stochastic Gradient Descent (SGD) method is utilized to calculate the optimum value of regression. The weakness of this scheme is that it can only implement a simple NN without any convolutional layer; thus, the accuracy is quite low. The weakness of SecureML relies on the non-colluding assumption. In the two-servers model, the servers can be untrusted but not collude with each other. If the servers may collude, the privacy of participants can be compromised.

MiniONN [117] is a privacy-preserving framework for transforming a NN into an oblivious Neural Network. The transformation process in MiniONN includes the nonlinear functions, with a price of negligible accuracy lost. There are two kinds of transformation provided by MiniONN, including oblivious transformation for the piecewise linear activation function and oblivious transformation for the smooth activation function. A smooth function can be transformed into a continuous polynomial by splitting the function into several parts. Then, for each part, polynomial approximation is used for the approximation, resulting in a piecewise linear function. Hence, MiniONN supports all activation functions that have either a monotonic range or piecewise polynomial or can be approximated into a polynomial function. The experiment showed that MiniONN outperforms Cryptonets [94] and SecureML [116] in terms of message size and latency. The main weakness is that MiniONN does not support batch processing. MiniONN is also based on honest-but-curious adversary, so it has no countermeasure against malicious adversary.

ABY3 [118] proposed by Mohassel *et al.*, is a protocol for privacy-preserving machine learning based on three-party computation (3PC). The main contribution of this protocol is its ability to switch among arithmetic, binary, and Yao's 3PC depending on the processing needs. The main purpose of ABY3 is to solve the classic PPDL problem that requires switching back and forth between arithmetic (for example addition and multiplication) and non-arithmetic operations (such as activation function approximation). The usual machine learning process works on arithmetic operations. As a result, it cannot perform a

polynomial approximation for activation function. ABY3 can be used to train linear regression, logistic regression, and NN models. Arithmetic sharing is used when training linear regression models. On the other hand, for computing logistic regression and NN models, binary sharing on three-party GC is utilized. The author also introduced a new fixed-point multiplication method for more than three-party computation, extending the 3PC scenario. This multiplication method is used to solve the limitation of using MPC with machine learning. MPC is suitable for working over rings, unlike machine learning which works on decimal values. ABY3 provides a new framework that is secure against malicious adversaries; so it is not limited to honest-but-curious adversary. However, since the protocols are built in their own framework, it will be difficult to be implemented with other deep learning scheme.

DeepSecure [104] is a framework that enables the use of deep learning in privacy-preserving environments. The author used OT and Yao's GC protocol [80] with CNN to perform the learning process. DeepSecure enables a collaboration between client and server to perform the learning process on a cloud server using data from the client. The security of the system was proven using an honest-but-curious adversary model. The GC protocol successfully keeps the client data private during the data transfer period. The weakness of this method is its limitation of number of instances processed each round. The method can only classify one instance during each prediction round. DeepSecure offers a preprocessing phase that reduces the size of data. The strength of DeepSecure is that the preprocessing phase can be adopted easily because it is independent from any cryptographic protocol. Its main weakness is the inability to process batch processing.

Chameleon [119] is a PPDL method that combines Secure-MPC and CNN. For the privacy part, Chameleon uses Yao's GC which enables two parties to perform joint computation without disclosing their own input. There are two phases: an online phase and offline phase. During the online phase all parties are allowed to communicate, whereas during the offline phase the cryptographic operations are precomputed. Chameleon utilizes vector multiplication (dot product) of signed fixed-point representation which improves the efficiency of heavy matrix multiplication for encrypted data classification. It successfully achieves faster execution compared with CryptoNets [94] and MiniONN [117]. Chameleon requires two non-colluding servers to ensure the data privacy and security. For the private inference, it requires an independent third party or a secure hardware such as Intel SGX. Chameleon is based on honest-but-curious adversary, there is no countermeasure against malicious adversary. Chameleon's protocol is based on two party computation, so it is not efficient to implement in more than two-party scenario.

SecureNN [120] provides the first system that ensures the privacy and correctness against honest-but-curious adversaries and malicious adversaries for complex NN computation. The system is based on secure MPC combined with CNN. SecureNN was tested on an MNIST dataset and successfully achieved more than 99% prediction accuracy with execution times 2-4 times faster than other secure MPC based PPDL, such as SecureML [116], MiniONN [117], Chameleon [119], and GAZELLE [121]. Its main contribution is developing a new protocol for Boolean computation (ReLU, Maxpool, and its derivatives) that has less communication overhead than Yao GC. This is how SecureNN achieves a faster execution time than the other techniques mentioned above. The weakness of SecureNN is claimed to refine more communication overhead compared to ABY3 [86]. If the SecureNN protocol is modified so that it utilizes matrix multiplication like ABY3, the number of communication rounds will be reduced.

CodedPrivateML [122] distributes the training computation across several stations and proposes a new approach for secret sharing of the data and DL model parameter that significantly reduces the computation overhead and complexity. However, the accuracy of this method is only about 95%, which

Table 12.2: Features of Our Surveyed Secure MPC-based PPDL

| References | Key Concept | Learning Type | Dataset |
|---|---|---|---|
| SecureML [116] | Proposes a combination of garbled circuit with oblivious transfer and secret sharing in a DNN environment | Server-assisted | MNIST CIFAR-10 |
| MiniONN [117] | Transforms a NN into an oblivious NN | Server-assisted | MNIST |
| ABY3 [118] | Provides an ability to switch between arithmetic, binary, and three-party computation | Server-assisted | MNIST |
| DeepSecure [104] | Enables a collaboration between client and server to do a learning process on cloud server | Server-assisted | MNIST |
| Chameleon [119] | Enables a secure joint computation with two distinguished phases; online and offline | Server-assisted | MNIST |
| SecureNN [120] | Develops a new protocol for Boolean computation that has small overhead | Server-assisted | MNIST |
| Coded PrivateML [122] | Proposes a distributed training computation across clients with a new approach of secret sharing | Server-assisted | MNIST |

is not as high as other method such as GAZELLE [121] or Chameleon [119].

Table 12.2 shows the features of our surveyed secure MPC-based PPDL.

## 12.3 Differential Privacy-based PPDL

The structure of differential privacy-based PPDL is shown in Fig. 12.6. First, training data are used to train the teacher model (1). Then, the teacher model is used to train the student model. In this case we illustrated the student model as a GAN model that consists of a generator and discriminator (2). Random noise is added to the generator as it generates fake training data (3). On the other hand, the teacher model trains the student model using the public data (4). The student model runs a zero-sum game between the generator and the discriminator. Then, the student model is ready to be used for the prediction process. A client sends a query (5) to the student model. The student model runs the

Table 12.3: Features of Our Surveyed Differential Privacy-based PPDL

| References | Key Concept | Learning Type | Dataset |
|---|---|---|---|
| PATE [123] | Proposes a differentially private learning process by utilizing teacher models and student models | Server-based | MNIST SVHN |
| Bu19 [124] | Proposes a Gaussian differential privacy that formalizes the original differential privacy-based PPDL | Server-based | MNIST MovieLens |

inference phase and returns the prediction result to the user (6).

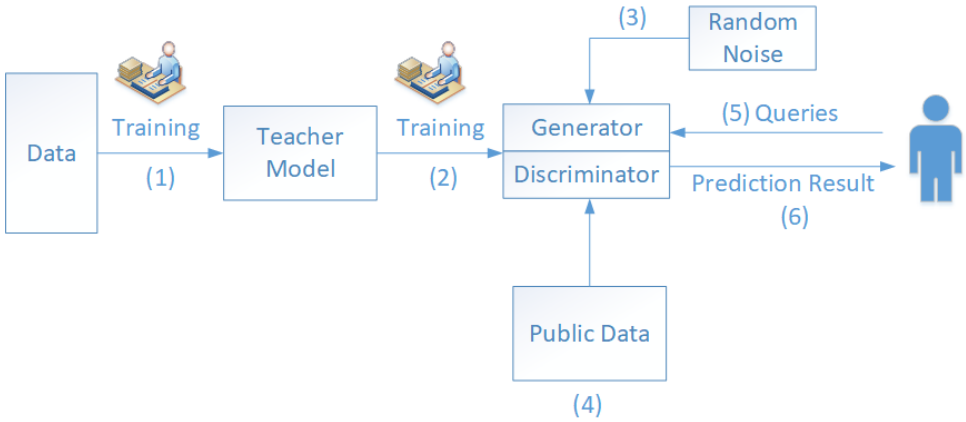


Figure 12.6: The Structure of Differential Privacy-based PPDL

Table 12.3 shows the features of our surveyed differential privacy-based PPDL.

Private Aggregation of Teacher Ensembles (PATE) [123] is a PPDL method for MLaaS that uses a differential privacy-based approach in Generative Adversarial Network (GAN). PATE is a black box approach that tries to ensure the privacy of data during training by using teacher-student models. During the training phase, the dataset is used to train the teacher models. Then, student models learn from the teacher models using a voting-based differential privacy method. By doing this, the teacher model is kept secretive, and the original data cannot be accessed by the student. The advantage of this model is due to the distinguished teacher model; when an adversary obtains a student model, the model will not give the adversary any confidential information. PATE has a serious weakness, which is not to provide good accuracy for complex data. If the data is too diverse, adding noise to the data will lower the performance of PATE. So, the performance of PATE depends on the type of input. It is only suitable for simple classification task. Furthermore, the computation cost is expensive due to many interactions between server and clients.

Another PPDL method that utilizes differential privacy is Bu19 [124]. Bu19 proposes Gaussian Differential Privacy (Gaussian DP) which formalizes the original DP technique as a hypothesis test from the adversaries' perspective.The concept of adding Gaussian noise is interesting. It must be evaluated

in order to analyze the trade-off between the noise and the accuracy. The scalability issue implemented in the daily life remains in question.

## 12.4 Secure Enclaves-based PPDL

The structure of secure enclaves-based PPDL is shown in Fig. 12.7. At first, a client sends data to the secure enclave environment (1). Then, the model provider sends the deep learning model to the enclaves (2). In the secure enclaves environment, the prediction process is executed using the client's data and the deep learning model (3). Then, the prediction result is sent to the client (4). The process in secure enclaves is guaranteed to be secure, and all of the data and models inside cannot be revealed to any other party outside the enclaves.

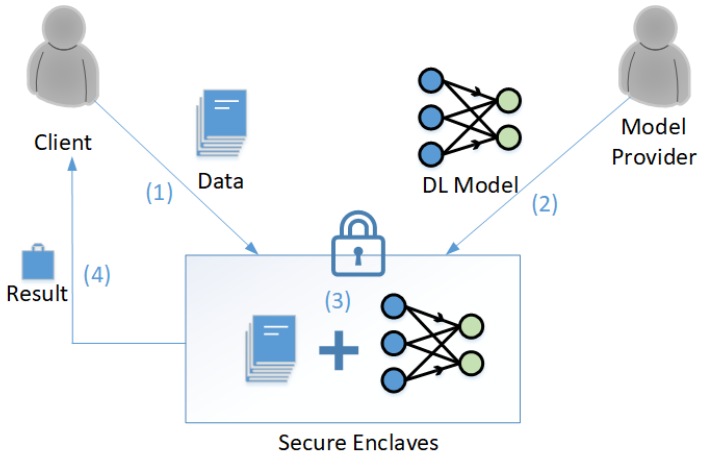


Figure 12.7: The Structure of Secure Enclaves-based PPDL

SLALOM [125] uses Trusted Execution Environments (TEEs), which isolate the computation process from untrusted software. The DNN computation is partitioned between trusted and untrusted parties. SLALOM runs DNN in the Intel SGX enclave which delegates the computation process to an untrusted GPU. The weakness of this approach is believed to limit CPU operation since the TEE does not allow to access GPU. A vulnerability by side channel attack may occur as shown by Van *et al.* [126].

Chiron [84] provides a black-box system for PPDL. The system conceals training data and model structure from the service provider. It utilizes SGX enclaves [127] and the Ryoan sandbox [87]. As SGX enclaves only protect the privacy of the model, the Ryoan sandbox is chosen here to ensure, even if the model tries to leak the data, that the data will be confined inside the sandbox, preventing the leakage. Chiron also supports a distributed training process by executing multiple enclaves that exchange model parameters through the server.

Chiron focuses on outsourced learning by using a secure enclave environment. The main difference between Chiron and Ohrimenko16 [128] is the code execution. Chiron allows the execution of untrusted code to update the model and implements protection by using sandboxes such that the code will not leak the data outside the enclave. On the other hand, Ohrimenko16 requires all codes inside the SGX enclave to be public to ensure that the code is trusted. The main weakness relies on the assumption that the model is not exposed to other parties. As a result, if an adversary can get an access to the trained model, there will be leakage of data, as shown by Shokri *et al.* [129]. This leakage problem can be solved by using differential privacy-based training algorithm [123].

Table 12.4: Features of Our Surveyed Secure Enclaves-based PPDL

| References | Key Concept | Learning Type | Dataset |
|---|---|---|---|
| Chiron [84] | Combines SGX enclaves and Ryoan sandbox to provide a black-box system for PPDL | Server-based | CIFAR ImageNet |
| SLALOM [125] | Utilizes a trusted execution environment to isolate computation processes | Server-based | - |

Table 12.4 shows the features of our surveyed secure enclaves-based PPDL.

## 12.5 Hybrid-based PPDL

Ohrimenko16 [128] proposes a secure enclave platform based on the SGX system for secure MPC. It focuses on collaborative learning, providing a prediction service in a cloud. Ohrimenko16 requires all codes inside the SGX enclave to be public to ensure that the code is trusted. The main weakness of this method is claimed to its inherent vulnerability to information leakage due to GAN attack as shown by Hitaj *et al.* [130].

Chase17 [131] wants to propose a private collaborative framework for machine learning. The main idea is to combine secure MPC with DP for the privacy part and leverage NN for the machine learning part. The weakness of this method is found to undergo a decrease in accuracy when implemented in a large network, exhibiting the scalability issue. In addition, its data privacy can only be guaranteed if the participants are non-colluding.

In GAZELLE [121], HE is combined with GC to ensure privacy and security in a MLaaS environment. For the HE library, it utilizes Single Instruction Multiple Data (SIMD) which includes addition and multiplication of ciphertext to improve the encryption speed. The Gazelle algorithm accelerates the convolutional and the matrix multiplication processes. An automatic switch between HE and GC is implemented such that encrypted data can be processed in NN. For the deep learning part, it leverages CNN comprising two convolutional layers, two ReLU layers as activation layers, one pooling layer, and one fully connected layer. The author used MNIST and CIFAR-10 datasets during the experiment and successfully showed that Gazelle outperforms several popular techniques such as MiniONN [117] and Cryptonets [94] in terms of run time. Furthermore, to prevent a linkage attack, Gazelle limits the number of classification queries from a client. The limitation of GAZELLE is claimed to support two-party computation scheme only since it utilizes garbled circuit for the secure exchange of two parties.

Ryffel18 [132] introduces a PPDL framework using federated learning built over PyTorch [133]. Federated learning requires multiple machines to train data in a decentralized environment. It enables clients to learn a shared prediction model using the data in their own device. The author combines secure MPC with DP to build a protocol enables federated learning. Overall, the proposed approach has overhead problem because of the bottleneck in the low-level library, compared to the high level python API. The proposed approach is vulnerable to collusion attack if the participants collude with each other.

CrypTFlow [134] combines secure enclaves with secret sharing in DNN to secure the learning process of the ImageNet dataset. The main weakness of CrypTFlow is believed not to support GPU processing. As a result, the computation overhead during the secure training is still high.

Table 12.5 shows the features of our surveyed hybrid-based PPDL.

Table 12.5: Features of Our Surveyed Hybrid-based PPDL

| **References** | **Key Concept** | **Learning Type** | **Dataset** |
|---|---|---|---|
| Ohrimenko16 [128] | Applies secure enclaves platform based for secure MPC | Server-based | MovieLens MNIST |
| Chase17 [131] | Proposes a private collaboration framework by combining secure MPC with DP | Server-assisted | MNIST |
| GAZELLE [121] | Combines HE with garbled circuit using SIMD | Server-based | MNIST CIFAR-10 |
| Ryffel18 [132] | Builds a new protocol for federated learning in PPDL | Server-assisted | Boston Housing Pima Diabetes |
| CrypTFlow [134] | Combines secure enclaves with secret sharing in DNN | Server-assisted | ImageNet |

We have summarized the general limitations of each PPDL method and our idea to overcome those limitations in table 12.6.

Table 12.6: Summary of Weakness and How to Overcome It

| **Method** | **Main Limitation** | **How to Overcome** |
|---|---|---|
| HE-based PPDL | More layer leads to more complexity because of the property of HE. | Adding batch normalization layer between pooling layer and activation layer. |
| | Accuracy highly dependents on the approximation of the activation function. | Using the polynomial with lowest possible degree as the activation function. |
| SecureMPC-based PPDL | Most of the current publications only guarantee privacy of clients based on honest-but-curious adversary, but they do not have protection against malicious adversary. | Designing a protocol that each participant shares their secret data, then the server runs secure computation to generate the output. By doing this, even if the server is malicious, it cannot see either the input or output value. |
| Differential Privacy-based PPDL | Differential privacy is computationally very expensive because it requires many interactions between server and client when adding the noise. | Distributed differential privacy can be a good solution to reduce the interactions. Since the architecture of previous differential privacy-based PPDL techniques are centralized, distributed differential privacy will be an interesting option. |
| Secure Enclaves-based PPDL | There is a data leakage issue due to side channel attack. | Utilizing differential privacy so that the adversary cannot get the real data, even if the side channel attack is successful. |

# Chapter 13. Comparison of State-of-the-Art PPDL Methods

## 13.1 Comparison Metrics

To compare the performances of each surveyed article, we used two kinds of metrics, qualitative metrics and quantitative metrics. Fig. 13.1 shows the metrics for the surveyed PPDL works in this dissertation. Qualitative metrics include Privacy of Client (PoC), Privacy of Model (PoM), and Privacy of Result (PoR). PoC means that neither the model owner nor the cloud server or any other party knows about the client data. PoM means that neither the client nor the cloud server or any other party knows about the DL model. PoR means that neither the model owner nor the cloud server or any other party can obtain the information about the prediction result. Quantitative metrics include accuracy and inference time. Accuracy means the percentage of correct predictions made by a PPDL model. The inference time is the time needed by the model to perform encryption/decryption, send data from the client to the server, and execute the classification process. We measured the average accuracy and inference time of each method. Then, we set the average value as the relative evaluation. If the accuracy value is higher than average, the accuracy of the proposed method is good. Furthermore, if the run time and data transfer are lower than average, the run time and data transfer of the proposed methods are good. We used the comparison data from the respective papers as we believe they are the best result to be achieved. We did not re-execute their codes as not all of the codes are open to the public. We focused our dissertation on the Hybrid PPDL method, which combines classical privacy-preserving with various deep learning practices.

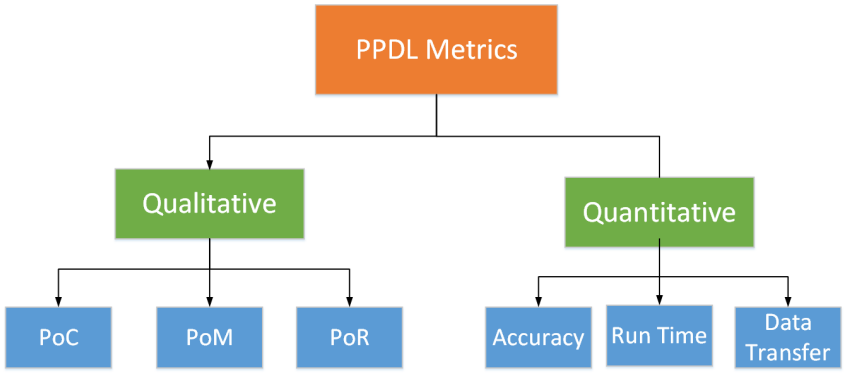


Figure 13.1: Metrics for Surveyed PPDL Works

## 13.2 Performance Comparison

We divided our comparison table into two types: performance comparison I and performance comparison II in Fig. 13.2 and Fig. 13.3, respectively. To compare the performance of each surveyed paper, we used the privacy metrics and performance metrics defined in section **??**. The privacy metrics include Privacy of Client (PoC), Privacy of Model (PoM), and Privacy of Result (PoR). The performance metrics include accuracy, run time, and data transfer.

| References | Used PP Technique | | | | | Privacy Parameters | | | ML/DL Method | Dataset Type |
|---|---|---|---|---|---|---|---|---|---|---|
| | HE | Secure-MPC | Oblivious Transfer | Differential Privacy | Secure Enclaves | PoC | PoM | PoR | | |
| CryptoNets [61] | O | X | X | X | X | O | X | O | CNN | Image |
| PATE [90] | X | X | X | O | X | O | O | X | GAN | Image |
| GAZELLE [88] | O | O | O | X | X | O | O | O | CNN | Image |
| TAPAS [72] | O | X | X | X | X | O | O | O | CNN | Image |
| FHE DiNN [75] | O | X | X | X | X | O | O | O | CNN | Image |
| E2DM [76] | O | X | X | X | X | O | O | O | CNN | Image |
| ABY3 [85] | X | O | O | X | X | X | O | O | CNN | Image |
| Xue18 [77] | O | X | X | X | X | O | X | O | CNN | Image |
| Aono17 [68] | O | X | X | X | X | O | X | O | NN | Image |
| Zhang19 [58] | O | X | X | X | X | O | X | O | C-Means algorithm | Image |
| ML Confidential [62] | O | X | X | X | X | O | X | O | Linear Means Classifier | CSV |
| Chabanne17 [70] | O | X | X | X | X | O | X | O | DNN | Image |
| CryptoDL [59] | O | X | X | X | X | O | X | O | CNN | Image |
| Liu18 [60] | O | X | X | X | X | O | X | O | CNN | Image |
| SecureML [83] | X | O | O | X | X | X | O | O | NN | Image |
| DeepSecure [71] | X | O | O | X | X | X | O | O | CNN | Image |
| MiniONN [84] | X | O | O | X | X | X | O | O | CNN | Image |
| SLALOM [92] | X | X | X | X | O | O | O | O | DNN | Image |
| SecureNN [87] | X | O | O | X | X | X | O | O | CNN | Image |
| Ryffel18 [99] | X | O | O | O | X | O | O | O | Federated Learning | CSV |
| Faster Crytonets [65] | O | X | X | X | X | O | X | O | CNN | Image |
| CodedPrivateML [89] | X | O | O | X | X | O | O | O | Logistic Regression | Image |
| Chiron [51] | X | X | X | X | O | O | O | O | DNN | Image |
| Bu19 [91] | X | X | X | O | X | O | O | O | DNN | Image |
| CrypTFlow [101] | X | O | O | X | O | O | O | O | DNN | Image |
| Chameleon [86] | X | O | O | X | X | X | O | O | CNN | Image |
| Chase17 [98] | X | O | O | O | X | X | O | O | CNN | Image |
| Ohrimenko16 [95] | X | O | O | X | O | O | X | O | DNN | CSV |

Figure 13.2: Performance Comparison I

| PPDL Type | Proposed Scheme | Accuracy (%) | Inference Time (s) |
|---|---|---|---|
| HE-based PPDL | Cryptonets [61] | Good (98.95) | Bad (297.5) |
| | Aono17 [68] | Good (97.00) | Good (120) |
| | Chabanne17 [70] | Good (99.30) | - |
| | CryptoDL [59] | Good (99.52) | Bad (320) |
| | TAPAS [72] | Good (98.60) | Good (147) |
| | FHE-DiNN [75] | Bad (96.35) | Good (1.64) |
| | E2DM [76] | Good (98.10) | Good (28.59) |
| | Xue18 [77] | Good (99.73) | - |
| | Liu18 [60] | Good (98.97) | Bad (477.6) |
| | Faster Cryptonets [65] | Bad (80.61) | - |
| | CryptoNN [78] | Bad (95.48) | - |
| | Zhang19 [58] | - | - |
| | ML Confidential [62] | Bad (95.00) | Bad (255.7) |
| Secure MPC-based PPDL | SecureML [83] | Bad (93.40) | Good (4.88) |
| | MiniONN [84] | Good (98.95) | Good (9.32) |
| | ABY3 [85] | Bad (94.00) | Good (0.01) |
| | DeepSecure [71] | Good (98.95) | Good (9.67) |
| | Chameleon [86] | Good (99.00) | Good (2.24) |
| | SecureNN [87] | Good (99.15) | Good (0.076) |
| | CodedPrivateML [89] | Bad (95.04) | Bad (110.9) |
| Differential Privacy-based PPDL | PATE [90] | Good (98.10) | - |
| | Bu19 [91] | Good (98.00) | - |
| Secure Enclaves-based PPDL | Chiron [51] | Bad (89.56) | - |
| | SLALOM [92] | Bad (92.4) | - |
| Hybrid-based PPDL | Ohrimenko16 [95] | Good (98.7) | Good (2.99) |
| | Chase17 [98] | Good (98.9) | - |
| | Gazelle [88] | - | Good (0.03) |
| | Ryffel18 [99] | Bad (70.3) | - |
| | CrypTFlow [101] | Bad (93.23) | Good (30) |

Figure 13.3: Performance Comparison II

## 13.3 Challenges and Weaknesses

In this section, we will discuss the challenges and weaknesses of utilizing PPDL for MLaaS from the papers that we surveyed. To analyze the limitations, we divided the PPDL approach into two main categories based on the type of transmission: the model parameter transmission approach and the data transmission approach. The model parameter transmission approach means that the model parameter is transmitted from the client to the server while the local data is kept by the client, and the training is performed on the client side. On the other hand, the data transmission approach means that the client data itself is transferred to the server for the training process. In short, the challenges and weaknesses of state-of-the-art PPDL methods are shown in Fig. 13.4.

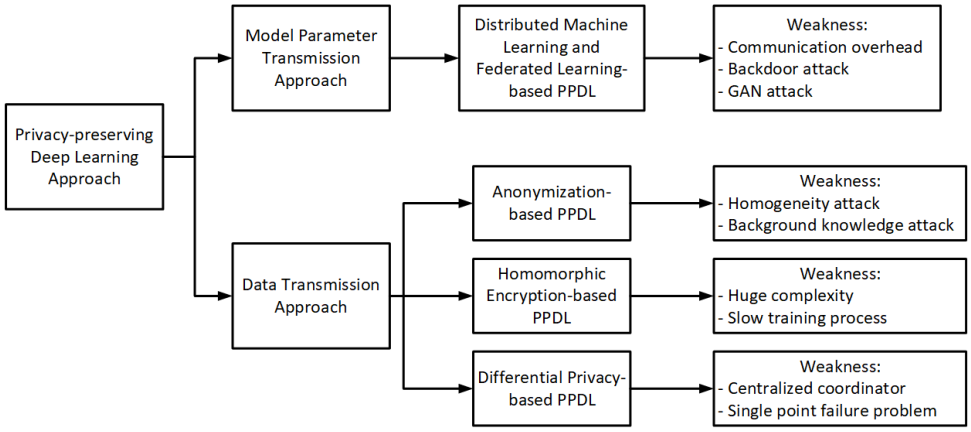


Figure 13.4: The Challenges and Weaknesses of State-of-the-Art PPDL Methods

### 13.3.1 Model Parameter Transmission Approach

In this approach, during the training process, a model parameter is transmitted instead of the training data. PPDLs based on distributed machine learning and federated learning are included in this scenario. In distributed learning [135, 136, 122, 137], data owners keep their own data secret without revealing this data to another party. During each training stage, participants send their locally computed parameters to the server. By doing this, the participants can learn collaboratively without revealing their own data [138]. On the other hand, in federated learning [139, 140, 141], model provider sends the model to participants. Then, each participant executes the training process using their local data, resulting in an updated model. After that, the updated model is sent back to the model provider. The model provider will measure the average value of the gradient descent and update the model. We can see that the model parameter transmission approach reduces the communication overhead but increases the local computation overhead. This learning approach is vulnerable to backdoor attacks [142] and GAN-based attacks [143].

### 13.3.2 Data Transmission Approach

In this approach, the participants send their data to the training server. Some PPDL methods that belong to this class are anonymization, HE, and DP-based PPDL. The main purpose of the anonymization technique is to remove the correlation between the data owner and the data entries. However, it requires a trusted coordinator to perform the anonymization process and distribute the result to the participants.

It is also vulnerable to a single point of failure as a trusted proxy needs to perform the anonymization process and send the result to the participants [144]. HE-based PPDL does not require key distribution management because the computation can be performed on encrypted data. However, it has limitations in the computation format. The computation is limited to a polynomial of bounded degree; thus, it works in a linear nature. Another weakness of HE-based PPDL is the slow training process as it has huge complexity, and the computation process will lead to data swelling [145].

A bootstrapping idea [67, 64, 146] has been introduced to solve this problem by reducing the complexity and the computation time. The majority of the work focuses on polynomial approximation for non-linear operations. The main goal of DP-based PPDL is to perturb the sample data for the training process. It is often used for data such as histograms or tables. The main weakness of DP-based PPDL is its centralized nature. One main trusted coordinator that is responsible for data collection and giving the response to queries from participants. This trusted coordinator is vulnerable to a single point of failure. If this kind of failure occurs and each participant perturbs the training data, the model will yield poor accuracy [145]. Thus, a centralized coordinator is very susceptible to the single point of failure problem. In a nutshell, we can conclude that the data transmission approach reduces the computation overhead but increase the communication overhead.

## 13.4 Analysis and Summary

After discussing the challenges and weaknesses in PPDL from the two categories above, we summarize the two main problems in PPDL: the computation overhead and communication overhead.

### 13.4.1 Computation Overhead

One of the most important issues in MLaaS is the computation overhead. In MLaaS, the overhead issues occur during the HE process, deep learning training (including inferencing), and data perturbation. Currently, utilizing deep learning for large-scale service is not feasible in real life because of this scalability problem.

### 13.4.2 Communication Overhead

In MLaaS, communication overhead occurs during the interaction among clients, model providers, and server providers. In particular, we can categorize communication overhead into the HE process, additive or multiplicative perturbation, and iterative communication. In the distributed machine learning scenario, including the federated learning, this factor is the main scalability problem that becomes the main issue. The iterative communications to exchange data and model parameters between each party will produce a significant overhead problem.

# Chapter 14. Attacks on DL Model and PPDL as a Possible Solution

## 14.1 Adversarial Model and Security Goals of PPDL

PPDL solutions on DL-as-a-service frameworks have three major security goals. The first goal is to prevent the server from acquiring the training data in the training phase which would be sensitive data owned by the client. All PPDL schemes contain privacy measures to prevent the direct leak of the training data. HE- and MPC-based approaches solve this by encrypting and distributing the training data, respectively. Some methods perform lower-layer calculations in the client side while hardware-based approaches encapsulate lower-layer calculations inside some confidential environment.

The second security goal of PPDL aims to prevent the server from directly acquiring the input to the model in the prediction phase. In most cases, this goal is achieved together with the first goal. This goal is only applied when the client delegates prediction to the server.

The third goal is to prevent the server from taking advantage of white-box access of the model. With the white-box access on a model, a server (as an adversary) may deploy several known attacks which are known to be easy on the white-box assumption. As DNNs tend to have more parameters than other machine learning algorithms due to the hidden layers, black-box models could retain more information on training data.

Many cryptography-based approaches achieve the third goal by keeping the parameters encrypted. However, some PPDL models do not assume this third goal and allow the server to access the plaintext parameters of the model. For instance, DP-based models allow white-box access, but the schemes aim to make the information extractable from the parameters negligible.

Although there are many other types of attacks on DL models, in this section we only discuss the attacks that can be mitigated by some of the PPDL approaches. In other words, the following attacks are related to one of the goals of PPDL. Table 14.1 provides a brief summary on PPDL as a countermeasure against attacks.

We categorize the adversarial model in PPDL based on the adversary's behavior, adversary's power, and adversary's corruption types as shown in Fig. 14.1.

### 14.1.1 Adversarial Model Based on the Behavior

We categorize the adversarial model based on the behavior into honest-but-curious and malicious.

#### Honest-but-Curious

In an Honest-but-Curious (HbC) adversary model, all parties, including the corrupted party, follow the security protocol honestly. They do not pursue any malicious activity toward the system or other participants. However, the corrupted party tries to perform a "curious action" to learn sensitive information from the model or from the other participants. This model is the one most commonly used in PPDL.

Table 14.1: PPDL as Countermeasures Against Attacks on DL Models

| **PPDL Types** | **Cryptography -based** | **DP -based** | **Hardware -based** |
|---|---|---|---|
| Membership Inference Attack | O | $\Delta$ | X |
| Model Inversion Attack | O | $\Delta^*$ | X |
| Model Extraction Attack | O | N/A | N/A |

O : An effective countermeasure for the given attack.

X : An ineffective countermeasure for the given attack.

$\Delta$ : The effectiveness of the defense against the given attack has a trade-off with the privacy-preserving parameters of DL models.

$\Delta^*$ : This trade-off has been confirmed for non-DL models, and it is expected to be the same for DL models.

N/A: The adversarial assumption of the given attack is not applicable for the PPDL method.

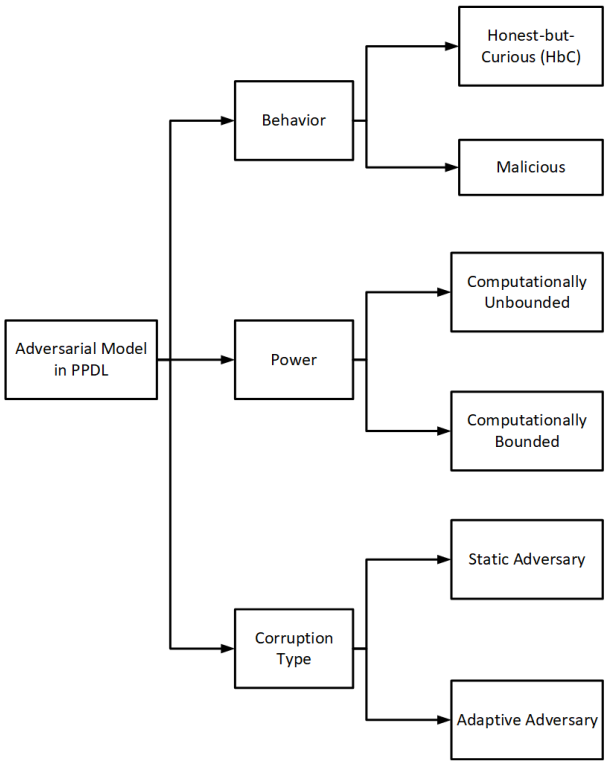


Figure 14.1: Adversarial Model in PPDL

**Malicious** This scenario is also known as the active adversary model because the corrupted parties will actively try to attack even if they must deviate from the existing security protocol. If the corrupted parties can prematurely halt their attacking process, sometimes the model is also recognized as a fail stop model.

### 14.1.2 Adversarial Model Based on the Power

We categorize adversarial model based on the behavior into computationally unbounded and computationally bounded.

**Computationally Unbounded** This means that the adversary has unlimited computational power. As a result, it is considered as the ideal adversary. It is usually used in theoretical information security field as it does not exist in real life.

**Computationally Bounded** This means that the adversary has limited computational power. Usually, it requires cryptographic assumption. The time assumption during the attack process is defined as the polynomial time.

### 14.1.3 Adversarial Model Based on Corruption Type

We categorize the adversarial model based on the corruption type into static adversary and adaptive adversary.

**Static Adversary** In this model, the corrupted parties are defined before the protocol starts. An honest party will always stay honest, and a corrupted party will always stay corrupted.

**Adaptive Adversary** In this model, an adversary will decide which party to corrupt based on the current situation. As a result, an honest party can become corrupted in the middle of protocol execution. However, in the adaptive model, an adversary can change the corrupted party such that the corrupted party can become honest again. This is classified as an adaptive-mobile adversary.

## 14.2 Attacks on DL Model and PPDL as the Countermeasure

### 14.2.1 Membership Inference Attack

Generally, membership inference means deciding whether given data were used for generating some aggregation of the data (or not). In the context of deep learning, a model itself (including the model parameters) can be regarded as the 'aggregation' of the training data. Therefore, membership inference attacks on DL models indicate attacks to decide whether given data belong to the training dataset (or not). Shokri *et al.* [129] provided one of the first suggestions of membership inference attacks.

Membership inference attacks are the attacks for the models violating the first security goal of PPDL. Stronger versions of membership inference attacks include extraction of some properties of sensitive training data or even recovery of the training data, which can be reduced to normal membership inference attacks. Usually, membership inference attacks harness overfitting during training, producing a difference in accuracy between the training data and the other data. Some defensive mechanisms dedicated to membership inference have been proposed including dropout [147] and adversarial regularization [148].

In cryptography-based PPDL models, the security against the membership inference attack can be reduced to the security of the underlying cryptosystems. In such models, the adversarial server cannot obtain model parameters in plaintext. Only if the model is public can the adversary have black-box access of the model, just like any outsider attacker. For HW-based models, the adversarial server owns white-box models, allowing the use of white-box membership inference attacks.

For DP-based models, the trade-off between the model accuracy and the performance of membership inference attacks according to the selection of the privacy parameter has been studied [149]. Appropriate choices of the privacy parameter result in moderate utility with low membership inference accuracy. However, further experiments are required for the extensibility of their analysis toward other types of tasks outside image classification.

### 14.2.2 Model Inversion Attack

As an attack toward the models does not satisfy the second security goal of PPDL, a model inversion attack is a prediction-phase attack introduced by Fredrikson *et al.* [150, 99]. Given the non-sensitive features of the original input data and their prediction results for a model, model inversion attacks aim to find the sensitive features of the input data.

In cryptography-based and HW-based PPDL models, we expect a similar advantage as that of membership inference attacks. For DP-based models, there has been limited research on the trade-off between the model accuracy and the attack performance, such as the analysis by Wang *et al.* [151] against regression models. Although a similar analysis for differentially private DL models remains for future work, we expect a similar trade-off.

### 14.2.3 Model Extraction Attack

Model extraction attacks [100], also known as model-stealing attacks, are attacks toward the third security goal of PPDL. When a black-box (target) model is given, the objective of model extraction attacks is to construct a model equivalent to the target model. Once an adversary succeeds with a model extraction attack, the adversary then accesses a white-box model. The attacker can take direct advantage of the model if the model owner sells the model access. The obtained model also becomes a "stepping stone" [152] toward further attacks utilizing white-box models.

Again, adversarial servers against cryptography-based DL models have negligible advantages over those of outsiders, excepting that of the model structure. Without the help of the client, the server cannot obtain the decrypted parameter values of the model.

Most differentially private models and hardware-based PPDL models do not fulfill the third security goal, as they reveal model parameters to the server. The extraction of such PPDL models is meaningless for the adversarial servers, as the servers already have the white-box access on the models, which is the purpose of the attack. Although the servers possess some part of the model parameters, it is relatively easy to extract the remaining parameters by observing the intermediate activation levels.

# Chapter 15. Concluding Remarks

To summarize the main trend, an annual roadmap that highlights the development of PPDL complemented with detailed comparisons of each PPDL work has been presented. Security goals and attack models on PPDL also have been discussed, with the possible countermeasures for each scenario. In brief, the trade-off between accuracy and complexity during the substitution process of the non-linear activation function is identified as the main challenge in PPDL.

The main challenge of PPDL is to ensure the PoC, PoM, and PoR simultaneously with two extra computations from the client's and model's perspectives while maintaining the computational performance. Last but not least, implementing PPDL based on federated learning will be an interesting topic. We believe that the future direction of PPDL is going to focus on combining federated learning and state-of-the-art PPDL to overcome the current privacy issues during data collection phase in MLaaS.

# Chapter 16. Summary and Open Problems

On the part I of this dissertation, we conclude that among FES, DFES, and mDFES; FES is the fastest, but has worst accuracy. mDFES is faster than DFES. Finally, the type of dataset affects the performance of deep learning method, as shown that mDFES is better than DFES for Malgenome dataset, while DFES is better than mDFES for Drebin dataset. Feature learning method is more suitable for non-binary dataset. Different type of dataset requires different deep learning method to achieve the best result. An open problem for future research in malware detection over Android is how to adapt feature learning method in binary dataset. Extracting substantial representation from binary dataset is a very challenging work. It will improve the performance of the model significantly.

On the part II of this dissertation, we have provided a complete review of state-of-the-art PPDL on MLaaS. Our discussion covers the classical PP method and the utilization of DL for PP. Our work also addresses the limitation of implementing novel DL techniques with PP, including the analysis of the original structure of NN and the modifications needed to use it in privacy-preserving environment. Furthermore, we have proposed a multi-scheme PPDL classification based on adversarial model, PP methods, and the challenges and weaknesses in state-of-the-art PPDL methods. An open problem for future research in PPDL is reducing computational burden. How to divide the burden between a client and a server optimally to achieve the best performance is a big challenge that needs to be addressed in the future.

# Bibliography


[1] R. Agrawal, V. Shah, S. Chavan, G. Gourshete, and N. Shaikh, "Android malware detection using machine learning," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE, 2020, pp. 1–4.

[2] E. M. S. Alkhateeb, "Dynamic malware detection using api similarity," in *2017 IEEE International Conference on Computer and Information Technology (CIT)*. IEEE, 2017, pp. 297–301.

[3] L. Han, C. Fu, D. Zou, C. Lee, and W. Jia, "Task-based behavior detection of illegal codes," *Mathematical and Computer Modelling*, vol. 55, no. 1-2, pp. 80–86, 2012.

[4] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 116–127.

[5] E. Hesamifard, H. Takabi, M. Ghasemi, and C. Jones, "Privacy-preserving machine learning in cloud," in *Proceedings of the 2017 on Cloud Computing Security Workshop*, 2017, pp. 39–43.

[6] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 123–142, 2018.

[7] V. K. Singh and A. K. Gupta, "From artificial to collective intelligence: perspectives and implications," in *2009 5th International Symposium on Applied Computational Intelligence and Informatics*. IEEE, 2009, pp. 545–550.

[8] J. Aycock, *Computer viruses and malware*. Springer Science & Business Media, 2006, vol. 22.

[9] K. Thompson, "Reflections on trusting trust," *Communications of the ACM*, vol. 27, no. 8, pp. 761–763, 1984.

[10] J. Parikka, *Digital contagions: A media archaeology of computer viruses*. Peter Lang, 2007, vol. 44.

[11] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 161–168.

[12] Z. Ghahramani, "Unsupervised learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 72–112.

[13] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[14] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.

[15] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of machine learning research*, vol. 11, no. 12, 2010.

[16] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–7.

[17] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[18] I. Firdausi, A. Erwin, A. S. Nugroho *et al.*, "Analysis of machine learning techniques used in behavior-based malware detection," in *2010 second international conference on advances in computing, control, and telecommunication technologies*. IEEE, 2010, pp. 201–203.

[19] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[20] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2016, pp. 577–582.

[21] O. E. David and N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[22] Z. Xu, S. Ray, P. Subramanyan, and S. Malik, "Malware detection using machine learning based analysis of virtual memory access patterns," in *Proceedings of the conference on design, automation & test in Europe*. European Design and Automation Association, 2017, pp. 169–174.

[23] L. Liu, B. Wang, B. Yu, and Q. Zhong, "Automatic malware classification and new malware detection using machine learning," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 9, pp. 1336–1347, 2017.

[24] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, "Malware detection using machine learning and deep learning," in *International Conference on Big Data Analytics*. Springer, 2018, pp. 402–411.

[25] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46 717–46 738, 2019.

[26] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, "Malware detection based on deep learning of behavior graphs," *Mathematical Problems in Engineering*, vol. 2019, 2019.

[27] W. Zhong and F. Gu, "A multi-level deep learning system for malware detection," *Expert Systems with Applications*, vol. 133, pp. 151–162, 2019.

[28] Y. Liu and Y. Wang, "A robust malware detection system using deep learning on api calls," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2019, pp. 1456–1460.

[29] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Android malware detection using deep learning on api method sequences," *arXiv preprint arXiv:1712.08996*, 2017.

[30] K. Raman *et al.*, "Selecting features to classify malware," *InfoSec Southwest*, vol. 2012, 2012.

[31] E. Gandotra, D. Bansal, and S. Sofat, "Zero-day malware detection," in *2016 Sixth International Symposium on Embedded Computing and System Design (ISED)*. IEEE, 2016, pp. 171–175.

[32] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 95–109.

[33] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[34] S. Y. Yerima and S. Sezer, "Droidfusion: a novel multilevel classifier fusion approach for android malware detection," *IEEE transactions on cybernetics*, vol. 49, no. 2, pp. 453–466, 2018.

[35] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.

[36] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[37] K. Kim and M. E. Aminanto, "Improving detection of wi-fi impersonation by fully unsupervised deep learning," in *International Workshop on Information Security Applications*. Springer, 2017, pp. 212–223.

[38] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature extraction: foundations and applications*. Springer, 2008, vol. 207.

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

[40] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996.

[41] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," *arXiv preprint arXiv:1412.6830*, 2014.

[42] X. Wang, Y. Wang, and L. Wang, "Improving fuzzy c-means clustering based on feature-weight learning," *Pattern recognition letters*, vol. 25, no. 10, pp. 1123–1132, 2004.

[43] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 94, 2018.

[44] S. Mishra, R. Prusty, and P. K. Hota, "Analysis of levenberg-marquardt and scaled conjugate gradient training algorithms for artificial neural network based ls and mmse estimated channel equalizers," in *2015 International Conference on Man and Machine Interfacing (MAMI)*. IEEE, 2015, pp. 1–7.

[45] J. Yu, X. Zheng, and S. Wang, "A deep autoencoder feature learning method for process pattern recognition," *Journal of Process Control*, vol. 79, pp. 1–15, 2019.

[46] O. Y. Al-Jarrah, O. Alhussein, P. D. Yoo, S. Muhaidat, K. Taha, and K. Kim, "Data randomization and cluster-based partitioning for botnet intrusion detection," *IEEE transactions on cybernetics*, vol. 46, no. 8, pp. 1796–1806, 2015.

[47] M. Masum and H. Shahriar, "Droid-nnet: Deep learning neural network for android malware detection," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 5789–5793.

[48] A. Hashem El Fiky, "Deep-droid: Deep learning for android malware detection," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, pp. 122–125, 2020.

[49] P. Samarati and L. Sweeney, "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression," 1998.

[50] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 3–es, 2007.

[51] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 2007, pp. 106–115.

[52] X. Xiao and Y. Tao, "M-invariance: towards privacy preserving re-publication of dynamic datasets," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 689–700.

[53] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of secure computation 4.11*, 1978, pp. 169–180.

[54] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[55] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Annual ACM on Symposium on Theory of Computing*. ACM, 2009, pp. 169–178.

[56] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun, "Batch fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 315–335.

[57] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 24–43.

[58] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[59] ——, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Advances in Cryptology–CRYPTO 2011*. Springer, 2011, pp. 505–524.

[60] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 75–92.

[61] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[62] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[63] M. Clear and C. M. Goldrick, "Attribute-based fully homomorphic encryption with a bounded number of inputs," *International Journal of Applied Cryptography*, vol. 3, no. 4, pp. 363–376, 2017.

[64] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *international conference on the theory and application of cryptology and information security*. Springer, 2016, pp. 3–33.

[65] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[66] S. Halevi and V. Shoup, "Algorithms in HElib," in *International Cryptology Conference*. Springer, 2014, pp. 554–571.

[67] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 617–640.

[68] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[69] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 457–473.

[70] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography Conference*. Springer, 2011, pp. 253–273.

[71] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of Cryptography Conference*. Springer, 2007, pp. 535–554.

[72] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Workshop on the theory and application of cryptographic techniques*. Springer, 1984, pp. 47–53.

[73] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*. Springer, 2001, pp. 213–229.

[74] B. Waters, "Efficient identity-based encryption without random oracles," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 114–127.

[75] C. Gentry, "Practical identity-based encryption without random oracles," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 445–464.

[76] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Annual ACM on Symposium on Theory of Computing*. ACM, 2008, pp. 197–206.

[77] R. Canetti, S. Halevi, and J. Katz, "A forward-secure public-key encryption scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 255–271.

[78] S. Agrawal, D. Boneh, and X. Boyen, "Efficient lattice (H) IBE in the standard model," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 553–572.

[79] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 89–98.

[80] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986, pp. 162–167.

[81] O. Goldreich, S. Micali, and A. Wigderson, "How to play ANY mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987, pp. 218–229.

[82] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[83] L. Qi, X. Zhang, S. Li, S. Wan, Y. Wen, and W. Gong, "Spatial-temporal data-driven service recommendation with privacy-preservation," *Information Sciences*, vol. 515, pp. 91–102, 2020.

[84] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *arXiv preprint arXiv:1803.05961*, 2018.

[85] R. Intel, "Software guard extensions programming reference," *Intel Corporation*, 2014.

[86] J. Doweck, W.-F. Kao, A. K.-y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation intel core: New microarchitecture code-named skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, 2017.

[87] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," *ACM Transactions on Computer Systems (TOCS)*, vol. 35, no. 4, pp. 1–32, 2018.

[88] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[89] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[90] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[91] Q. Zhang, L. T. Yang, A. Castiglione, Z. Chen, and P. Li, "Secure weighted possibilistic c-means algorithm on cloud for clustering big data," *Information Sciences*, vol. 479, pp. 515–525, 2019.

[92] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[93] W. Liu, F. Pan, X. A. Wang, Y. Cao, and D. Tang, "Privacy-preserving all convolutional net based on homomorphic encryption," in *International Conference on Network-Based Information Systems*. Springer, 2018, pp. 752–762.

[94] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[95] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," in *International Conference on Information Security and Cryptology*. Springer, 2012, pp. 1–21.

[96] M. Skurichina and R. P. Duin, "Bagging, boosting and the random subspace method for linear classifiers," *Pattern Analysis & Applications*, vol. 5, no. 2, pp. 121–135, 2002.

[97] S.-J. Kim, A. Magnani, and S. Boyd, "Robust fisher discriminant analysis," in *Advances in neural information processing systems*, 2006, pp. 659–666.

[98] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *arXiv preprint arXiv:1811.09953*, 2018.

[99] 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*, 2015, pp. 1322–1333.

[100] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 601–618.

[101] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[102] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[103] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network." *IACR Cryptology ePrint Archive*, vol. 2017, p. 35, 2017.

[104] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[105] A. Sanyal, M. J. Kusner, A. Gascon, and V. Kanade, "Tapas: Tricks to accelerate (encrypted) prediction as a service," *arXiv preprint arXiv:1806.03461*, 2018.

[106] M. Kim and P. Smaragdis, "Bitwise neural networks," *arXiv preprint arXiv:1601.06071*, 2016.

[107] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[108] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Annual International Cryptology Conference*. Springer, 2018, pp. 483–512.

[109] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1209–1222.

[110] H. Xue, Z. Huang, H. Lian, W. Qiu, J. Guo, S. Wang, and Z. Gong, "Distributed large scale privacy-preserving deep mining," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 418–422.

[111] R. Xu, J. B. Joshi, and C. Li, "Cryptonn: Training neural networks over encrypted data," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1199–1209.

[112] R. Krishnapuram and J. M. Keller, "The possibilistic c-means algorithm: insights and recommendations," *IEEE transactions on Fuzzy Systems*, vol. 4, no. 3, pp. 385–393, 1996.

[113] D. E. Gustafson and W. C. Kessel, "Fuzzy clustering with a fuzzy covariance matrix," in *1978 IEEE conference on decision and control including the 17th symposium on adaptive processes*. IEEE, 1979, pp. 761–766.

[114] P. Smyth, "Model selection for probabilistic clustering using cross-validated likelihood," *Statistics and computing*, vol. 10, no. 1, pp. 63–72, 2000.

[115] A. Boulemtafes, A. Derhab, and Y. Challal, "A review of privacy-preserving techniques for deep learning," *Neurocomputing*, vol. 384, pp. 21–45, 2020.

[116] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[117] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 619–631.

[118] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 35–52.

[119] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 707–721.

[120] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 26–49, 2019.

[121] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1651–1669.

[122] J. So, B. Guler, A. S. Avestimehr, and P. Mohassel, "Codedprivateml: A fast and privacy-preserving framework for distributed machine learning," *arXiv preprint arXiv:1902.00641*, 2019.

[123] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.

[124] Z. Bu, J. Dong, Q. Long, and W. J. Su, "Deep learning with gaussian differential privacy," *arXiv preprint arXiv:1911.11607*, 2019.

[125] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[126] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 991–1008.

[127] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, 2016, pp. 1–9.

[128] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 619–636.

[129] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 3–18.

[130] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 603–618.

[131] M. Chase, R. Gilad-Bachrach, K. Laine, K. E. Lauter, and P. Rindal, "Private collaborative neural network learning." *IACR Cryptology ePrint Archive*, vol. 2017, p. 762, 2017.

[132] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[133] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[134] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," *arXiv preprint arXiv:1909.07814*, 2019.

[135] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[136] J. Hamm, A. C. Champion, G. Chen, M. Belkin, and D. Xuan, "Crowd-ml: A privacy-preserving learning framework for a crowd of smart devices," in *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015, pp. 11–20.

[137] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[138] G. Song and W. Chai, "Collaborative learning for deep neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 1832–1841.

[139] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 1–11.

[140] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.

[141] V. Mugunthan, A. Peraire-Bueno, and L. Kagal, "Privacyfl: A simulator for privacy-preserving and secure federated learning," *arXiv preprint arXiv:2002.08423*, 2020.

[142] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can you really backdoor federated learning?" *arXiv preprint arXiv:1911.07963*, 2019.

[143] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2512–2520.

[144] I. J. Vergara-Laurens, L. G. Jaimes, and M. A. Labrador, "Privacy-preserving mechanisms for crowdsensing: Survey and research challenges," *IEEE Internet of Things Journal*, vol. 4, no. 4, pp. 855–869, 2016.

[145] L. Jiang, R. Tan, X. Lou, and G. Lin, "On lightweight privacy-preserving collaborative learning for internet-of-things objects," in *Proceedings of the International Conference on Internet of Things Design and Implementation*, 2019, pp. 70–81.

[146] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 360–384.

[147] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models," *arXiv preprint arXiv:1806.01246*, 2018.

[148] M. Nasr, R. Shokri, and A. Houmansadr, "Machine learning with membership privacy using adversarial regularization," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 634–646.

[149] M. A. Rahman, T. Rahman, R. Laganière, N. Mohammed, and Y. Wang, "Membership inference attack against differentially private deep learning model." *Transactions on Data Privacy*, vol. 11, no. 1, pp. 61–79, 2018.

[150] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 17–32.

[151] Y. Wang, C. Si, and X. Wu, "Regression model fitting under differential privacy and model inversion attack," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[152] J. Zhao, Y. Chen, and W. Zhang, "Differential privacy preservation in deep learning: Challenges, opportunities and solutions," *IEEE Access*, vol. 7, pp. 48 901–48 911, 2019.

# Acknowledgments in Korean

이 논문을 작성하기 위해 많은 분들의 도움을 받았습니다. 먼저 부족한 저를 잘 이끌어주시고 많은 지식뿐만 아니라 인생의 지혜도 알려주시며 항상 저를 믿어주신 김광조 교수님께 감사드립니다. 또한 바쁘신 와중 에도 저의 학위논문심사에 참여하셔서 진심어린 조언을 해주신 이기혁 교수님, 최호진 교수님, 임을규 교수님과 김익균 박사님께도 감사의 말씀을 드립니다.

연구실 친구 최락용, 이지은, Muhamad Erza Aminanto, Edwin Opare, 안형철, 김성숙, 최낙준, 한성호, 홍동연, 백승근 모두 감사합니다.

I am very grateful to my parents for their strong support and endless love. Thank you so much to my big family: my cousin, uncle, aunt, Lulu, and Ago that always cheer me during hard time. Thank you to my international friends in KAIST, members of Indonesian student association in KAIST: KAIST-INA; especially Erza for the hundreds of hours we spent playing PES to release our stress, Bivan for helping me a lot to endure the hardship during my qualifying exam time, Seli for accompanying me doing some random chat and random things, Monica (the kindest person in KAIST-INA) who remembers everyone's birthday, and Miko who has similar hobby and supporting same club: Chelsea FC. I also want to thank my badminton club mate at PUD for all those great time and support during these 4.5 years, especially Alvin, the person I played with most often and my personal coach; and Steven, my partner during KISA sports day and Perpikalympics (even though we were getting rekt as usual). Finally, I want to show a deep gratitude to Indonesia Endowment Fund for Education (LPDP) and my advisor, Prof. Kwangjo Kim, who have supported me financially during my time in KAIST. Without all of them, I will not be able to finish my degree, so let me express my gratitude sincerely from my heart.

# Curriculum Vitae in Korean


이 름: Harry Chandra Tanuwidjaja

생 년 월 일: 1991년 11월 18일

출 생 지: Semarang, Indonesia

주 소: KH Wahid Hasyim 6, Semarang, Indonesia


## 학 력

2009. 8. – 2013. 7. Institute Technology Bandung (ITB) (Bachelor)

2013. 8. – 2015. 3. Institute Technology Bandung (ITB) (Master)

2016. 8. – 2020 Korea Advanced Institute of Science and Technology (Ph.D)

## 연 구 업 적

### Book

1. Kwangjo Kim, Muhamad Erza Aminanto, and **Harry Chandra Tanuwidjaja**, "Network Intrusion Detection using Deep Learning: A Feature Learning Approach," ISBN 978-981-13-1443-8, 2018, Springer.
2. Kwangjo Kim and **Harry Chandra Tanuwidjaja**, "Privacy-Preserving Deep Learning: A Comprehensive Survey," 2020, Springer (to be published).

### International Refereed Journal

1. Muhamad Erza Aminanto, Rakyong Choi, **Harry Chandra Tanuwidjaja**, Paul D. Yoo, and Kwangjo Kim, "Deep Abstraction and Weighted Feature Selection for Wi-Fi Impersonation Detection," IEEE Trans. on Information Forensics and Security (IFS) Vol.13, No.3, pp.621-636.
2. **Harry Chandra Tanuwidjaja**, Rakyong Choi, and Kwangjo Kim, "Preserving Deep Learning on Machine Learning as a Service - A Comprehensive Survey," IEEE Access, 8, pp.167425-167447.

### International Refereed Conference

1. Muhamad Erza Aminanto, **Harry Chandra Tanuwidjaja**, Paul D. Yoo, and Kwangjo Kim, "Wi-Fi Intrusion Detection Using Weighted-Feature Selection for Neural Networks Classifier," International Workshop on Big Data and Information Security 2017 IWBIS 2017, Sep., 23-24, 2017, Jakarta, Indonesia.
2. **Harry Chandra Tanuwidjaja**, Rakyong Choi, and Kwangjo Kim, "A Survey on Deep Learning Techniques for Privacy-Preserving," Privacy-Preserving Machine Learning Workshop 2019, Aug.18. 2019, UCSB, USA

3. **Harry Chandra Tanuwidjaja**, Rakyong Choi, and Kwangjo Kim, "A Survey on Deep Learning Techniques for Privacy-Preserving," The Second International Conference on Machine Learning for Cyber Security (ML4CS 2019), Sep. 20-22, 2019, Xidian, China

**International Conference**

1. Muhamad Erza Aminanto, **Harry Chandra Tanuwidjaja**, Paul D. Yoo, and Kwangjo Kim, "Weighted Feature Selection Techniques for Detecting Impersonation Attack in Wi-Fi Networks," 2017 Symposium on Cryptography and Information Security, Session 2C2-4 (SCIS 2017), Jan., 24-27, 2017, Naha, Japan.
2. **Harry Chandra Tanuwidjaja**, Rakyong Choi, and Kwangjo Kim, "Limitations of Privacy-Preserving for Confidential Data Training by Deep Learning," 2019 Symposium on Cryptography and Information Security, Session 2C1-2 (SCIS 2019), Jan., 22-25, 2019, Otsu, Japan.
3. **Harry Chandra Tanuwidjaja** and Kwangjo Kim, "Enhancing Malware Detection by Modified Deep Abstraction and Weighted Feature Selection," 2020 Symposium on Cryptography and Information Security, Session 4F1-1 (SCIS 2020), Jan., 28-31, 2020, Kochi, Japan.

**Patent**

1. Muhamad Erza Aminanto, **Harry Chandra Tanuwidjaja**, 최락용, 김광조, "가중치 선택을 통한 무선 근거리 망의 침입탐지 방식 및 장치," 2018.01.18. 특허출원(제10-2018-0006369호).
2. Muhamad Erza Aminanto, **Harry Chandra Tanuwidjaja**, 최락용, 김광조, "무선 Wi-Fi 망에서 딥러닝을 이용한 위장 공격 특장점 분석 장치 및 도구," 2017.12.22. 특허출원(제10-2017-0178307호).

# Appendix and Source Code

**Dataset Pre-processing**

```
fid = fopen ('drebinwithoutheader.txt','r'); %change to MALG here
data=textscan(fid,repmat('%s ',1,216), 'delimiter','\t');

target=data{216};
i_end=size(target,1);

for i=1:i_end
    if strcmp(target{i,1},'B')
        newM(i,216)=1;
    else if strcmp(target{i,1},'S')
        newM(i,216)=2;
    else  newM5(i,155)=3;
        end
    end
end
jumlahkolom=215;
for p=1:jumlahkolom
    target=data{p};
    i_end=size(target,1);
    for i=1:i_end
        tmp9=target{i};
        tmp10=num2cell(tmp9);
        tmp7=cell2mat(tmp10);
        tmp8=str2double(tmp7);
        newM(i,p)=tmp8;
     end
end

csvwrite('norm_drebin_train.txt',newM); %change to MALG here
```

**FES-DREB/MALG-BAL/UNB**

```
raw_train=importdata('norm_drebin_train.txt'); %change to MALG here
raw_test=importdata('norm_drebin_test.txt'); %change to MALG here
tmpt1=raw_train(:,216)==1;%benign
tmpt3=raw_train(:,216)==2;%malware
tmpt2=raw_test(:,216)==1;%benign
tmpt4=raw_test(:,216)==2;%malware
train_IA=raw_train(tmpt3,:);
test_IA=raw_test(tmpt4,:);
train_N=raw_train(tmpt1,:);
test_N=raw_test(tmpt2,:);
rng(1);
n_train = size(train_N,1);
c_train = cvpartition(n_train,'Holdout',3/5); %for balancing modify here
train_N_reduced=train_N(test(c_train),:);
rng(1);
n_test = size(test_N,1);
c_test = cvpartition(n_test,'Holdout',3/5); %for balancing modify here
test_N_reduced=test_N(test(c_test),:);
raw_train=cat(1,train_N_reduced,train_IA); %merging reduced benign +
malware train
```

```
raw_test=cat(1,test_N_reduced,test_IA); %merging reduced benign + malware
test
csvwrite('bal_norm_drebin_train.txt',raw_train);
csvwrite('bal_norm_drebin_test.txt',raw_test);

x_train_all=raw_train(:,1:215);
x_train_all=x_train_all';
y_train_all_int=raw_train(:,216);
y_train_all_int=y_train_all_int';
y_train_all=dummyvar(y_train_all_int);
y_train_all=y_train_all';
x_test_all=raw_test(:,1:215);
x_test_all=x_test_all';
y_test_all_int=raw_test(:,216);
y_test_all_int=y_test_all_int';
y_test_all=dummyvar(y_test_all_int);
y_test_all=y_test_all';

rng('default');
hiddenSize1 = 100;
autoenc_saya = trainAutoencoder(x_train_all,hiddenSize1, ...
    'MaxEpochs',400, ...
    'L2WeightRegularization',0.004, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.15, ...
    'ScaleData', false);%
feat1_saya = encode(autoenc_saya,x_train_all);
hiddenSize2 = 50;
autoenc2_saya = trainAutoencoder(feat1_saya,hiddenSize2, ...
    'MaxEpochs',100, ...
    'L2WeightRegularization',0.002, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.1, ...
    'ScaleData', false);
feat2_saya = encode(autoenc2_saya,feat1_saya);
softnet_saya = trainSoftmaxLayer(feat2_saya,y_train_all,'MaxEpochs',400);
deepnet_saya = stack(autoenc_saya,autoenc2_saya,softnet_saya);
view(deepnet_saya)
y = deepnet_saya(x_test_all);
plotconfusion(y_test_all,y);
csvwrite('SAE_out_feat_TBD.csv',feat2_saya);

raw_train=importdata('bal_norm_drebin_train.txt');
extracted=importdata('SAE_out_feat_TBD.csv');
extracted=extracted';
y_train_all_int=raw_train(:,216);
feature=horzcat(extracted,y_train_all_int);
indeks_att=[1:1:51];
feature=vertcat(indeks_att,feature);
csvwrite('trainandextracted.csv',feature);

d_IA_train=importdata('trainandextracted.csv');
d_IA_train=d_IA_train(1:11245,:);
d_IA_train_WoT=d_IA_train(:,1:50);
d_IA_train_T=d_IA_train(:,51);
d_IA_train_T_dummy=dummyvar(d_IA_train_T);
Xtrain=d_IA_train_WoT;
Xtrain=Xtrain';
Ytrain=d_IA_train_T_dummy;
Ytrain=Ytrain';
```

```
tic
d_IA_test=importdata('trainandextracted.csv');
d_IA_test=d_IA_test(2:11246,:); d_IA_test_trans=d_IA_test';
selected_features=[33;37;45;8;29;39;11;10;30;15;42;19;1;13;23;26;17;28;3;12
;6;9;4;5;2];
ukuran_mat=size(selected_features);
jumlah_fitur=ukuran_mat(1,1);
indeks_fitur=[1:51];
for j=1:50
    for i=1:jumlah_fitur
        if indeks_fitur(1,j)==selected_features(i,1)
            indeks_fitur(2,j)=1;
            break;
        else indeks_fitur(2,j)=0;
        end
    end
end

indeks_fitur_aja=indeks_fitur(2,:);
indeks_fitur_aja=indeks_fitur_aja';
indeks_fitur_aja_kelas=indeks_fitur_aja

indeks_fitur_aja_kelas(51,1)=1;
indeks_fitur_aja_logical=logical(indeks_fitur_aja_kelas);

test_data=d_IA_test_trans;
test_data=test_data(indeks_fitur_aja_logical,:);
input_test=test_data(1:jumlah_fitur,:);
target_test=test_data((jumlah_fitur+1),:);
target_test_final=dummyvar(target_test);
target_test_final=target_test_final';
toc
```

**DFES-DREB/MALG-BAL/UNB**

```
raw_train=importdata('norm_drebin_train.txt'); %change to MALG here
raw_test=importdata('norm_drebin_test.txt'); %change to MALG here
tmpt1=raw_train(:,216)==1;%benign
tmpt3=raw_train(:,216)==2;%malware
tmpt2=raw_test(:,216)==1;%benign
tmpt4=raw_test(:,216)==2;%malware

train_IA=raw_train(tmpt3,:);
test_IA=raw_test(tmpt4,:);
train_N=raw_train(tmpt1,:);
test_N=raw_test(tmpt2,:);
rng(1);
n_train = size(train_N,1);
c_train = cvpartition(n_train,'Holdout',3/5);   %for balancing modify here
train_N_reduced=train_N(test(c_train),:);
rng(1);
n_test = size(test_N,1);%get #rows #benign test
c_test = cvpartition(n_test,'Holdout',3/5);   %for balancing modify here
test_N_reduced=test_N(test(c_test),:);

raw_train=cat(1,train_N_reduced,train_IA); %merging reduced benign +
malware train
```

```
raw_test=cat(1,test_N_reduced,test_IA); %merging reduced benign + malware
test
csvwrite('bal_norm_drebin_train.txt',raw_train);
csvwrite('bal_norm_drebin_test.txt',raw_test);

x_train_all=raw_train(:,1:215);
x_train_all=x_train_all';
y_train_all_int=raw_train(:,216);
y_train_all_int=y_train_all_int';
y_train_all=dummyvar(y_train_all_int);
y_train_all=y_train_all';
x_test_all=raw_test(:,1:215);
x_test_all=x_test_all';
y_test_all_int=raw_test(:,216);
y_test_all_int=y_test_all_int';
y_test_all=dummyvar(y_test_all_int);
y_test_all=y_test_all';

rng('default');
hiddenSize1 = 100;
autoenc_saya = trainAutoencoder(x_train_all,hiddenSize1, ...
    'MaxEpochs',400, ...
    'L2WeightRegularization',0.004, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.15, ...
    'ScaleData', false);%

feat1_saya = encode(autoenc_saya,x_train_all);
hiddenSize2 = 50;
autoenc2_saya = trainAutoencoder(feat1_saya,hiddenSize2, ...
    'MaxEpochs',100, ...
    'L2WeightRegularization',0.002, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.1, ...
    'ScaleData', false);
feat2_saya = encode(autoenc2_saya,feat1_saya);
softnet_saya = trainSoftmaxLayer(feat2_saya,y_train_all,'MaxEpochs',400);
deepnet_saya = stack(autoenc_saya,autoenc2_saya,softnet_saya);
view(deepnet_saya)
y = deepnet_saya(x_test_all);
plotconfusion(y_test_all,y);
csvwrite('SAE_out_feat_TBD.csv',feat2_saya);

raw_train=importdata('bal_norm_drebin_train.txt');
extracted=importdata('SAE_out_feat_TBD.csv');
extracted=extracted';
x_train_all=raw_train(:,1:215);
y_train_all_int=raw_train(:,216);
feature=horzcat(x_train_all,extracted);
feature=horzcat(feature,y_train_all_int);
indeks_att=[1:1:266];
feature=vertcat(indeks_att,feature);
csvwrite('trainandextracted.csv',feature);


d_IA_train=importdata('trainandextracted.csv');
d_IA_train=d_IA_train(2:11246,:);
d_IA_train_WoT=d_IA_train(:,1:265);
d_IA_train_T=d_IA_train(:,266);
d_IA_train_T_dummy=dummyvar(d_IA_train_T);
```

```
Xtrain=d_IA_train_WoT;
Xtrain=Xtrain';
Ytrain=d_IA_train_T_dummy;
Ytrain=Ytrain';

berat=net.IW{1,1};
indeks=net.inputs{1}.range;
for l=1:265
    range_indeks=indeks(l,2)-indeks(l,1);
    if range_indeks==0
        tmp(l,1)=0;
    else tmp(l,1)=1;
    end
end
tmp=logical(tmp);
indeks(:,1)=[1:265];
indeks2=indeks(tmp,:);
indeks_aja=indeks2(:,1);
indeks_aja=indeks_aja';

berat=abs(berat);
sizenya = size(berat);
col=sizenya(1,2);
row=sizenya(1,1);
berat_per_input=zeros(1,col);
berat_per_input(1,:)=indeks_aja;
for i=1:col
    berat_per_input(2,i)=0;
    for j=1:row
        berat_per_input(2,i)=berat_per_input(2,i)+berat(j,i);
    end
end
berat_per_input=berat_per_input';
berat_sort=sortrows(berat_per_input, 2);
tmp2=berat_sort(:,2)>=8.1;
selected_features=berat_sort(tmp2,1);

tic
d_IA_test=importdata('trainandextracted.csv');
d_IA_test=d_IA_test(2:11246,:);
d_IA_test_trans=d_IA_test';
selected_features=[27;53;63;48;147;37;29;36;245;114;231;217;18;59;177;186;1
65;172;31;19;194;91;83;259;80;233;211;16;239;39;208;70;60;64;183;129;118;15
7;173;182;123;110;124;6;7;102;185;138;54;94;9;207;111;35;197;52;105;109;33;
42;10];
ukuran_mat=size(selected_features);
jumlah_fitur=ukuran_mat(1,1);
indeks_fitur=[1:265];
for j=1:265
    for i=1:jumlah_fitur
        if indeks_fitur(1,j)==selected_features(i,1)
            indeks_fitur(2,j)=1;
            break;
        else indeks_fitur(2,j)=0;
        end
    end
end

indeks_fitur_aja=indeks_fitur(2,:);
indeks_fitur_aja=indeks_fitur_aja';
```

```
indeks_fitur_aja_kelas=indeks_fitur_aja;
indeks_fitur_aja_kelas(266,1)=1;
indeks_fitur_aja_logical=logical(indeks_fitur_aja_kelas);

test_data=d_IA_test_trans;
test_data=test_data(indeks_fitur_aja_logical,:);
input_test=test_data(1:jumlah_fitur,:);
target_test=test_data((jumlah_fitur+1),:);
target_test_final=dummyvar(target_test);
target_test_final=target_test_final';
toc
```

**MDFES-DREB/MALG-BAL/UNB**

```
raw_train=importdata('norm_drebin_train.txt'); %change to MALG here
raw_test=importdata('norm_drebin_test.txt'); %change to MALG here
tmpt1=raw_train(:,216)==1;%benign
tmpt3=raw_train(:,216)==2;%malware
tmpt2=raw_test(:,216)==1;%benign
tmpt4=raw_test(:,216)==2;%malware

train_IA=raw_train(tmpt3,:);
test_IA=raw_test(tmpt4,:);
train_N=raw_train(tmpt1,:);
test_N=raw_test(tmpt2,:);
rng(1);
n_train = size(train_N,1);
c_train = cvpartition(n_train,'Holdout',3/5);   %for balancing modify here
train_N_reduced=train_N(test(c_train),:);
rng(1);
n_test = size(test_N,1);
c_test = cvpartition(n_test,'Holdout',3/5);   %for balancing modify here
test_N_reduced=test_N(test(c_test),:);

raw_train=cat(1,train_N_reduced,train_IA); %merging reduced benign +
malware train
raw_test=cat(1,test_N_reduced,test_IA); %merging reduced benign + malware
test
csvwrite('bal_norm_drebin_train.txt',raw_train);
csvwrite('bal_norm_drebin_test.txt',raw_test);


x_train_all=raw_train(:,1:215);
x_train_all=x_train_all';
y_train_all_int=raw_train(:,216);
y_train_all_int=y_train_all_int';
y_train_all=dummyvar(y_train_all_int);
y_train_all=y_train_all';
x_test_all=raw_test(:,1:215);
x_test_all=x_test_all';
y_test_all_int=raw_test(:,216);
y_test_all_int=y_test_all_int';
y_test_all=dummyvar(y_test_all_int);
y_test_all=y_test_all';

rng('default');
hiddenSize1 = 100;
autoenc_saya = trainAutoencoder(x_train_all,hiddenSize1, ...
    'MaxEpochs',400, ...
```

```
    'L2WeightRegularization',0.004, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.15, ...
    'ScaleData', false);%
feat1_saya = encode(autoenc_saya,x_train_all);
hiddenSize2 = 50;
autoenc2_saya = trainAutoencoder(feat1_saya,hiddenSize2, ...
    'MaxEpochs',100, ...
    'L2WeightRegularization',0.002, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.1, ...
    'ScaleData', false);
feat2_saya = encode(autoenc2_saya,feat1_saya);
softnet_saya = trainSoftmaxLayer(feat2_saya,y_train_all,'MaxEpochs',400);
deepnet_saya = stack(autoenc_saya,autoenc2_saya,softnet_saya);
view(deepnet_saya)
y = deepnet_saya(x_test_all);
plotconfusion(y_test_all,y);
csvwrite('SAE_out_feat_TBD.csv',feat2_saya);


d_IA_train=importdata('bal_norm_drebin_train.txt');
d_IA_train=d_IA_train(1:11245,:);
d_IA_train_WoT=d_IA_train(:,1:215);
d_IA_train_T=d_IA_train(:,216);
d_IA_train_T_dummy=dummyvar(d_IA_train_T);
Xtrain=d_IA_train_WoT;
Xtrain=Xtrain';
Ytrain=d_IA_train_T_dummy;
Ytrain=Ytrain';

berat=net.IW{1,1};
indeks=net.inputs{1}.range;
for l=1:215
    range_indeks=indeks(l,2)-indeks(l,1);
    if range_indeks==0
        tmp(l,1)=0;
    else tmp(l,1)=1;
    end
end
tmp=logical(tmp);
indeks(:,1)=[1:215];
indeks2=indeks(tmp,:);
indeks_aja=indeks2(:,1);
indeks_aja=indeks_aja';

berat=abs(berat);
sizenya = size(berat);
col=sizenya(1,2);
row=sizenya(1,1);
berat_per_input=zeros(1,col);
berat_per_input(1,:)=indeks_aja;
for i=1:col
    berat_per_input(2,i)=0;
    for j=1:row
        berat_per_input(2,i)=berat_per_input(2,i)+berat(j,i);
    end
end
berat_per_input=berat_per_input';
berat_sort=sortrows(berat_per_input, 2);
tmp2=berat_sort(:,2)>=10.5;
```

```
selected_features=berat_sort(tmp2,1);

raw_train=importdata('bal_norm_drebin_train.txt');
extracted=importdata('SAE_out_feat_TBD.csv');
extracted=extracted';
x_train_all=raw_train(:,1:215);
y_train_all_int=raw_train(:,216);
feature=horzcat(x_train_all,extracted);
feature=horzcat(feature,y_train_all_int);
indeks_att=[1:1:266];
feature=vertcat(indeks_att,feature);
csvwrite('trainandextracted.csv',feature);

tic
d_IA_test=importdata('trainandextracted.csv');
d_IA_test=d_IA_test(2:11246,:);
d_IA_test_trans=d_IA_test';
selected_features=[165;89;3;152;116;209;71;31;92;32;18;106;147;213;101;182;
181;26;118;40;171;189;83;76;50;169;111;184;52;103;157;124;16;173;155;64;70;
42;53;197;105;183;11;107;9;109;110;186;33;7;216;217;218;219;220;221;222;223
;224;225;226;227;228;229;230;231;232;233;234;235;236;237;238;239;240;241;24
2;243;244;245;246;247;248;249;250;251;252;253;254;255;256;257;258;259;260;2
61;262;263;264;265];
ukuran_mat=size(selected_features);
jumlah_fitur=ukuran_mat(1,1);
indeks_fitur=[1:265];
for j=1:265
    for i=1:jumlah_fitur
        if indeks_fitur(1,j)==selected_features(i,1)
            indeks_fitur(2,j)=1;
            break;
        else indeks_fitur(2,j)=0;
        end
    end
end

indeks_fitur_aja=indeks_fitur(2,:);
indeks_fitur_aja=indeks_fitur_aja';
indeks_fitur_aja_kelas=indeks_fitur_aja;
indeks_fitur_aja_kelas(266,1)=1;
indeks_fitur_aja_logical=logical(indeks_fitur_aja_kelas);
test_data=d_IA_test_trans;
test_data=test_data(indeks_fitur_aja_logical,:);
input_test=test_data(1:jumlah_fitur,:);
target_test=test_data((jumlah_fitur+1),:);
target_test_final=dummyvar(target_test);
target_test_final=target_test_final';
toc
```

Feature details of Drebin and Malgenome Dataset

| Feature Name | Details |
|---|---|
| transact | API call signature |
| onServiceConnected | API call signature |
| bindService | API call signature |
| attachInterface | API call signature |

| | |
|---|---|
| ServiceConnection | API call signature |
| android.os.Binder | API call signature |
| SEND_SMS | Manifest Permission |
| Ljava.lang.Class.getCanonicalName | API call signature |
| Ljava.lang.Class.getMethods | API call signature |
| Ljava.lang.Class.cast | API call signature |
| Ljava.net.URLDecoder | API call signature |
| android.content.pm.Signature | API call signature |
| android.telephony.SmsManager | API call signature |
| READ_PHONE_STATE | Manifest Permission |
| getBinder | API call signature |
| ClassLoader | API call signature |
| Landroid.content.Context.registerReceiver | API call signature |
| Ljava.lang.Class.getField | API call signature |
| Landroid.content.Context.unregisterReceiver | API call signature |
| GET_ACCOUNTS | Manifest Permission |
| RECEIVE_SMS | Manifest Permission |
| Ljava.lang.Class.getDeclaredField | API call signature |
| READ_SMS | Manifest Permission |
| getCallingUid | API call signature |
| Ljavax.crypto.spec.SecretKeySpec | API call signature |
| android.intent.action.BOOT_COMPLETED | Intent |
| USE_CREDENTIALS | Manifest Permission |
| MANAGE_ACCOUNTS | Manifest Permission |
| android.content.pm.PackageInfo | API call signature |
| KeySpec | API call signature |
| TelephonyManager.getLine1Number | API call signature |
| DexClassLoader | API call signature |
| HttpGet.init | API call signature |
| SecretKey | API call signature |
| Ljava.lang.Class.getMethod | API call signature |
| System.loadLibrary | API call signature |
| android.intent.action.SEND | API call signature |
| Ljavax.crypto.Cipher | API call signature |
| WRITE_SMS | Manifest Permission |
| READ_SYNC_SETTINGS | Manifest Permission |
| AUTHENTICATE_ACCOUNTS | Manifest Permission |
| android.telephony.gsm.SmsManager | API call signature |
| WRITE_HISTORY_BOOKMARKS | Manifest Permission |
| TelephonyManager.getSubscriberId | API call signature |
| mount | Commands signature |
| INSTALL_PACKAGES | Manifest Permission |
| Runtime.getRuntime | API call signature |
| CAMERA | Manifest Permission |
| Ljava.lang.Object.getClass | API call signature |
| WRITE_SYNC_SETTINGS | Manifest Permission |
| READ_HISTORY_BOOKMARKS | Manifest Permission |
| Ljava.lang.Class.forName | API call signature |
| INTERNET | Manifest Permission |
| android.intent.action.PACKAGE_REPLACED | Intent |

| Binder | API call signature |
|---|---|
| android.intent.action.SEND_MULTIPLE | Intent |
| RECORD_AUDIO | Manifest Permission |
| IBinder | API call signature |
| android.os.IBinder | API call signature |
| createSubprocess | API call signature |
| NFC | Manifest Permission |
| ACCESS_LOCATION_EXTRA_COMMANDS | Manifest Permission |
| URLClassLoader | API call signature |
| WRITE_APN_SETTINGS | Manifest Permission |
| abortBroadcast | API call signature |
| BIND_REMOTEVIEWS | Manifest Permission |
| android.intent.action.TIME_SET | Intent |
| READ_PROFILE | Manifest Permission |
| TelephonyManager.getDeviceId | API call signature |
| MODIFY_AUDIO_SETTINGS | Manifest Permission |
| getCallingPid | API call signature |
| READ_SYNC_STATS | Manifest Permission |
| BROADCAST_STICKY | Manifest Permission |
| android.intent.action.PACKAGE_REMOVED | Intent |
| android.intent.action.TIMEZONE_CHANGED | Intent |
| WAKE_LOCK | Manifest Permission |
| RECEIVE_BOOT_COMPLETED | Manifest Permission |
| RESTART_PACKAGES | Manifest Permission |
| Ljava.lang.Class.getPackage | API call signature |
| chmod | Commands signature |
| Ljava.lang.Class.getDeclaredClasses | API call signature |
| android.intent.action.ACTION_POWER_DISCONNECTED | Intent |
| android.intent.action.PACKAGE_ADDED | Intent |
| PathClassLoader | API call signature |
| TelephonyManager.getSimSerialNumber | API call signature |
| Runtime.load | API call signature |
| TelephonyManager.getCallState | API call signature |
| BLUETOOTH | Manifest Permission |
| READ_CALENDAR | Manifest Permission |
| READ_CALL_LOG | Manifest Permission |
| SUBSCRIBED_FEEDS_WRITE | Manifest Permission |
| READ_EXTERNAL_STORAGE | Manifest Permission |
| TelephonyManager.getSimCountryIso | API call signature |
| sendMultipartTextMessage | API call signature |
| PackageInstaller | API call signature |
| VIBRATE | Manifest Permission |
| remount | Commands signature |
| android.intent.action.ACTION_SHUTDOWN | Intent |
| sendDataMessage | API call signature |
| ACCESS_NETWORK_STATE | Manifest Permission |
| chown | Commands signature |
| HttpPost.init | API call signature |
| Ljava.lang.Class.getClasses | API call signature |
| SUBSCRIBED_FEEDS_READ | Manifest Permission |

| | |
|---|---|
| TelephonyManager.isNetworkRoaming | API call signature |
| CHANGE_WIFI_MULTICAST_STATE | Manifest Permission |
| WRITE_CALENDAR | Manifest Permission |
| android.intent.action.PACKAGE_DATA_CLEARED | Intent |
| MASTER_CLEAR | Manifest Permission |
| HttpUriRequest | API call signature |
| UPDATE_DEVICE_STATS | Manifest Permission |
| WRITE_CALL_LOG | Manifest Permission |
| DELETE_PACKAGES | Manifest Permission |
| GET_TASKS | Manifest Permission |
| GLOBAL_SEARCH | Manifest Permission |
| DELETE_CACHE_FILES | Manifest Permission |
| WRITE_USER_DICTIONARY | Manifest Permission |
| android.intent.action.PACKAGE_CHANGED | Intent |
| android.intent.action.NEW_OUTGOING_CALL | Intent |
| REORDER_TASKS | Manifest Permission |
| WRITE_PROFILE | Manifest Permission |
| SET_WALLPAPER | Manifest Permission |
| BIND_INPUT_METHOD | Manifest Permission |
| divideMessage | API call signature |
| READ_SOCIAL_STREAM | Manifest Permission |
| READ_USER_DICTIONARY | Manifest Permission |
| PROCESS_OUTGOING_CALLS | Manifest Permission |
| CALL_PRIVILEGED | Manifest Permission |
| Runtime.exec | API call signature |
| BIND_WALLPAPER | Manifest Permission |
| RECEIVE_WAP_PUSH | Manifest Permission |
| DUMP | Manifest Permission |
| BATTERY_STATS | Manifest Permission |
| ACCESS_COARSE_LOCATION | Manifest Permission |
| SET_TIME | Manifest Permission |
| android.intent.action.SENDTO | Intent |
| WRITE_SOCIAL_STREAM | Manifest Permission |
| WRITE_SETTINGS | Manifest Permission |
| REBOOT | Manifest Permission |
| BLUETOOTH_ADMIN | Manifest Permission |
| TelephonyManager.getNetworkOperator | API call signature |
| /system/bin | Commands signature |
| MessengerService | API call signature |
| BIND_DEVICE_ADMIN | Manifest Permission |
| WRITE_GSERVICES | Manifest Permission |
| IRemoteService | API call signature |
| KILL_BACKGROUND_PROCESSES | Manifest Permission |
| SET_ALARM | API call signature |
| ACCOUNT_MANAGER | API call signature |
| /system/app | Commands signature |
| android.intent.action.CALL | Intent |
| STATUS_BAR | Manifest Permission |
| TelephonyManager.getSimOperator | API call signature |
| PERSISTENT_ACTIVITY | Manifest Permission |

| | |
|---|---|
| CHANGE_NETWORK_STATE | Manifest Permission |
| onBind | API call signature |
| Process.start | API call signature |
| android.intent.action.SCREEN_ON | Intent |
| Context.bindService | API call signature |
| RECEIVE_MMS | Manifest Permission |
| SET_TIME_ZONE | Manifest Permission |
| android.intent.action.BATTERY_OKAY | Intent |
| CONTROL_LOCATION_UPDATES | Manifest Permission |
| BROADCAST_WAP_PUSH | Manifest Permission |
| BIND_ACCESSIBILITY_SERVICE | Manifest Permission |
| ADD_VOICEMAIL | Manifest Permission |
| CALL_PHONE | Manifest Permission |
| ProcessBuilder | API call signature |
| BIND_APPWIDGET | Manifest Permission |
| FLASHLIGHT | Manifest Permission |
| READ_LOGS | Manifest Permission |
| Ljava.lang.Class.getResource | API call signature |
| defineClass | API call signature |
| SET_PROCESS_LIMIT | Manifest Permission |
| android.intent.action.PACKAGE_RESTARTED | Intent |
| MOUNT_UNMOUNT_FILESYSTEMS | Manifest Permission |
| BIND_TEXT_SERVICE | Manifest Permission |
| INSTALL_LOCATION_PROVIDER | Manifest Permission |
| android.intent.action.CALL_BUTTON | Intent |
| android.intent.action.SCREEN_OFF | Intent |
| findClass | API call signature |
| SYSTEM_ALERT_WINDOW | Manifest Permission |
| MOUNT_FORMAT_FILESYSTEMS | Manifest Permission |
| CHANGE_CONFIGURATION | Manifest Permission |
| CLEAR_APP_USER_DATA | Manifest Permission |
| intent.action.RUN | Intent |
| android.intent.action.SET_WALLPAPER | Intent |
| CHANGE_WIFI_STATE | Manifest Permission |
| READ_FRAME_BUFFER | Manifest Permission |
| ACCESS_SURFACE_FLINGER | Manifest Permission |
| Runtime.loadLibrary | API call signature |
| BROADCAST_SMS | Manifest Permission |
| EXPAND_STATUS_BAR | Manifest Permission |
| INTERNAL_SYSTEM_WINDOW | Manifest Permission |
| android.intent.action.BATTERY_LOW | Intent |
| SET_ACTIVITY_WATCHER | Manifest Permission |
| WRITE_CONTACTS | Manifest Permission |
| android.intent.action.ACTION_POWER_CONNECTED | Intent |
| BIND_VPN_SERVICE | Manifest Permission |
| DISABLE_KEYGUARD | Manifest Permission |
| ACCESS_MOCK_LOCATION | Manifest Permission |
| GET_PACKAGE_SIZE | Manifest Permission |
| MODIFY_PHONE_STATE | Manifest Permission |
| CHANGE_COMPONENT_ENABLED_STATE | Manifest Permission |

| | |
|---|---|
| CLEAR_APP_CACHE | Manifest Permission |
| SET_ORIENTATION | Manifest Permission |
| READ_CONTACTS | Manifest Permission |
| DEVICE_POWER | Manifest Permission |
| HARDWARE_TEST | Manifest Permission |
| ACCESS_WIFI_STATE | Manifest Permission |
| WRITE_EXTERNAL_STORAGE | Manifest Permission |
| ACCESS_FINE_LOCATION | Manifest Permission |
| SET_WALLPAPER_HINTS | Manifest Permission |
| SET_PREFERRED_APPLICATIONS | Manifest Permission |
| WRITE_SECURE_SETTINGS | Manifest Permission |
| class | B=Benign; S=Malware |