TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Sammy Alonso Calle Torres 214162IVCM

MACHINE LEARNING MULTIVIEW FOR MALWARE DETECTION ON ANDROID DEVICES

Master's Thesis

Supervisor: Hayretdin Bahsi
Phd

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sammy Alonso Calle Torres 214162IVCM

MASINÕPPE MULTIVAADE PAHAVARA TUVASTAMISEKS ANDROID-SEADMETES

Magistritöö

Juhendaja: Hayretdin Bahsi
Phd

Tallinn 2025

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Sammy Alonso Calle Torres

02.01.2025

Abstract

The increased use of Android mobile devices has led to a rise in mobile malware targeting these devices in recent years. Consequently, several studies are currently focusing on the application of machine learning for malware detection. While there are existing studies that utilize machine learning techniques, particularly supervised learning, the findings mainly emphasize the training and testing of models using these methods. In contrast, there is a limited number of studies exploring semi-supervised learning, specifically in the context of multiview learning for labeling samples and detecting malware.

This research will focus on implementing a multiview approach using the co-training algorithm, which is a semi-supervised learning technique. We will compare this approach with various machine learning algorithms to assess their performance against other techniques. The study will utilize non-stationary models with the aim of evaluating the models on novel malware while emulating a real-life scenario.

The main goal is to determine which algorithm and technique perform best on a dataset of novel malware. Specifically, we will use Random Forest and Multilayer Perceptron algorithms, along with different techniques for fitting the data into the models. The performance of these experiments will primarily be evaluated using the recall metric.

Annotatsioon

Masinõppe multivaade pahavara tuvastamiseks Android-seadmetes

Android-mobiilseadmete laialdasem kasutamine on viimastel aastatel toonud kaasa nendele seadmetele suunatud mobiilsete pahavarade arvu kasvu. Seetõttu keskenduvad mitmed uuringud praegu masinõppe rakendamisele pahavara tuvastamiseks. Kuigi on olemas uuringuid, mis kasutavad masinõppe tehnikaid, eriti juhendatud õppimist, rõhutavad tulemused peamiselt nende meetodite abil mudelite väljaõpet ja testimist. Seevastu on piiratud arv uuringuid, mis uurivad osaliselt juhendatud õppimist, eriti mitmevaatelise õppe kontekstis proovide märgistamiseks ja pahavara tuvastamiseks.

See uuring keskendub mitmevaatelise lähenemisviisi rakendamisele, kasutades kaastreeningu algoritmi, mis on pooleldi juhendatud õppetehnika. Võrdleme seda lähenemist erinevate masinõppe algoritmidega, et hinnata nende toimivust võrreldes teiste tehnikatega. Uuringus kasutatakse mittestatsionaarseid mudeleid, mille eesmärk on hinnata uudse pahavara mudeleid, jäljendades samal ajal reaalset stsenaariumi.

Peamine eesmärk on kindlaks teha, milline algoritm ja tehnika toimivad uudse pahavara andmekogumis kõige paremini. Täpsemalt kasutame Random Forest ja Multilayer Perceptron algoritme ning erinevaid tehnikaid andmete mudelitesse sobitamiseks. Nende katsete toimivust hinnatakse peamiselt tagasikutsumise mõõdiku abil.

List of Abbreviations and Terms

ML	Machine Learning
MLP	Multi Layer Perceptron
SSL	Semi Supervised Learning
SL	Supervised Learning
RF	Random Forest
MV	MultiView
CT	Co-training

Table of Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1. Introduction

Malware is software designed to cause abnormal and often harmful behavior in computer systems. It has been a persistent issue since the early days of computing, dating back to the creation of the first malware, the "Creeper Virus," [1] which emerged long before the internet was launched. While malware has primarily been associated with desktop computers and laptops, mobile devices have also become significant targets for malicious software, creating serious challenges.

To understand the significance of mobile malware, it’s crucial to consider the evolution of mobile devices and their current importance in people’s daily lives around the world. Mobile devices are now used for various tasks that were once done on personal computers (PCs), both online and offline.

The term "smartphone" was first used in 1997 by Ericsson [2] to describe devices that combined standard phone functions, such as making calls, with advanced computing capabilities. The IBM Simon, released in 1994 [3], marked the beginning of this new category. It was capable of sending emails, using a calendar, and operating a touchscreen. However, its high price and limited functionalities hindered widespread adoption.

The real revolution in smartphones began in 2007 [4] with the launch of Apple’s iPhone, a groundbreaking mobile phone that offered internet communication, email access comparable to desktops, web browsing, searching, maps. A year later, Apple introduced the App Store [5], a digital distribution platform for mobile apps on its iOS devices, primarily the iPhone and iPod Touch. The App Store quickly became a transformative force in the mobile app ecosystem by providing a centralized marketplace for developers to distribute their apps to a vast audience of iPhone users. The huge success of the iPhone in the market drew the attention of other tech giants regarding the potential of smartphones, including Google, which launched its mobile operating system, Android.

The launch of Android began with the HTC Dream in 2008 [6]. Subsequently, Google formed the "Open Handset Alliance," [7] a consortium of hardware, software, and telecommunications companies aimed at promoting the development and adoption of Android. Additionally, Android was designed with an open-source approach, meaning that any developer with the necessary knowledge could create applications for the operating system. These strategies led to Android’s rapid growth in the market, which allowed it to surpass

Apple's mobile OS in August 2012, capturing 71.77% of the total mobile operating system market share by December 2024 [8], as illustrated in Figure 1.

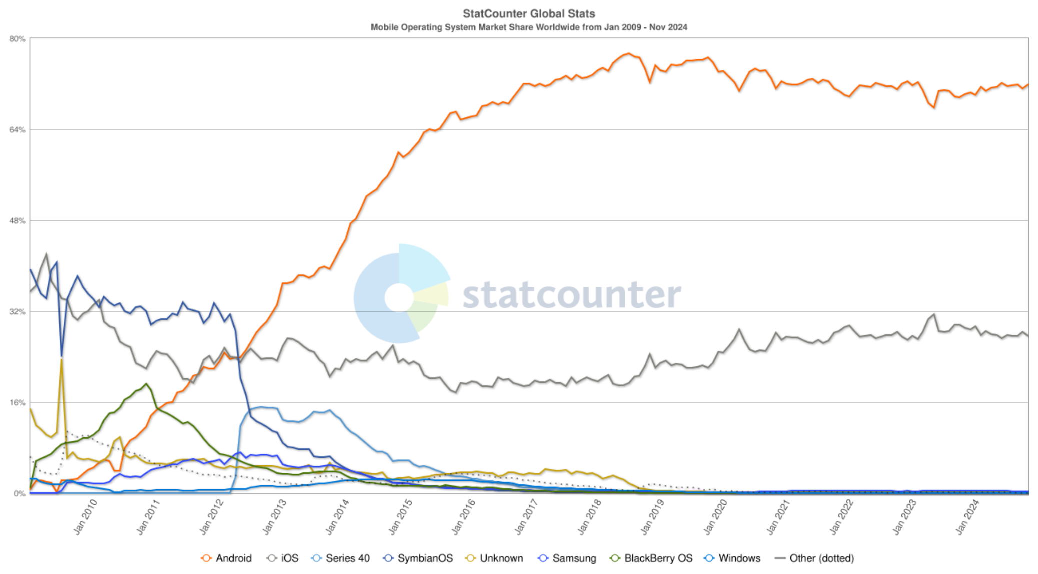

Figure 1. Mobile OS market from 2009-2024 share according to StatCounter [8]

From their inception, mobile platforms have undergone years of improvement and adaptation, allowing them to gain additional functionalities beyond merely providing access to email, calendars, and web browsing. The introduction and ongoing enhancement of sensors and additional hardware, such as cameras, have improved to the point where they can replace traditional amateur cameras. Furthermore, the inclusion of cameras has led to the development of biometric identification capabilities, such as facial recognition [2]. Fingerprint sensors, which are now widely used in smartphones, also serve as a form of user identification [9]. Additionally, features like NFC (Near Field Communication) have been added to mobile devices [10], allowing smartphones to begin replacing plastic bank cards. These advancements have enabled smartphones to support more significant applications, including mobile banking and government identification applications.

The functionalities of today's smartphones make them essential tools for individuals, but they also store a wealth of important and sensitive information, making them targets for malicious actors. Cybercriminals create malicious software aimed at infecting devices to steal or hijack information or, in some cases, damage the device itself. According to AV-Test Institute, over 1.1 million new malware variants emerged in 2024, alongside 3 million potentially unwanted applications released in the same year. The most common malware families include "Agent." The AV-Atlas [11] reports that by December 2024, the total amount of malware for the Android OS platform reached 35.5 million. Among the most frequently discovered malware families in the last two weeks of December 2024 were the "Agent" family and the "FakeApp" type of malware. Agent [12] malware operates in the background of a mobile device without the user's knowledge, waiting silently for commands from a Command and Control (C&C) server. These commands can range from stealing and transmitting personal information to executing DDoS attacks against targeted victims. Conversely, "FakeApp" [13] masquerades as a legitimate application claiming to perform various tasks, such as acting as an antivirus or updater. Its true intention is often to generate revenue through ad displays and redirecting users to install other apps — usually legitimate ones available on Google Play.

One of the main reasons Android devices are particularly vulnerable to malware is their open-source nature, which allows easy access to development knowledge for anyone interested in creating applications for the platform. Moreover, Android permits the installation of APKs (Android Packages) from unverified sources [14], meaning the Google Play Store is not the only outlet for applications. Malware applications can also be found in the Google Play Store [15] due to insufficient app filtering accuracy.

Given Android's dominant share of the mobile market, the increasing quantity of malware created each year, and Google's limited control over the software provided through its marketplace, this thesis will explore the application of Multiview to label and classify mobile malware effectively.

1.1 Problem Statement and Research Questions

Android is the most widely used operating system for mobile devices, holding over 70% of the market share. While its open-source nature contributes to its popularity, it also makes Android a prime target for malware. The ability to install applications from unauthorized sources, combined with limitations in Google Play Store's ability to fully verify uploaded applications, increases this vulnerability.

Even though there are strategies to mitigate the risk of malware such as avoiding untrusted software and protecting sensitive information detecting malicious software, like trojans and spyware, can be difficult or even impossible for the average user. Given these challenges, cybersecurity specialists have had to develop and improve malware detection techniques. Current techniques include signature-based and behavior-based detection methods [16]. In particular, machine learning (ML) has proven to be an effective technique for identifying malicious behavior [17], enabling the detection of novel malware and allowing for adaptations to the changing behaviors of malicious applications.

The process of machine learning for malware detection involves several steps: collecting and identifying data samples, preprocessing the dataset, training the model, testing and validating the model, and finally, deploying the model. Creating a dataset is a crucial step in this process and can be expensive, especially if it requires specialists to label unidentified samples before they can be used for training. Proper labeling of the dataset is critical for accurate malware classification; even the most powerful ML models will struggle to identify malware if the training is based on incorrectly labeled data or an insufficient amount of data due to resource constraints.

Identifying samples within a dataset has become increasingly challenging due to the constant influx of new applications. This situation raises costs for researchers and cybersecurity experts who need to label these newly identified applications. Moreover, as cybercriminals utilize various strategies to conceal malware, specialists require more time to classify applications as either malware or benign. This scenario underscores the growing importance of machine learning techniques, particularly semi-supervised learning [18].

In examining the use of semi-supervised learning for identifying Android malware, several research gaps become apparent. This thesis will investigate the effectiveness of the multiview approach in accurately labeling and identifying new malware using different machine learning (ML) algorithms. A comparison will be made between the multiview approach and other ML techniques, including supervised learning (SL) employing static features such as permissions, dynamic features like system calls, a combination of both, and single-view semi-supervised learning (SSL). This comparison will utilize various updating strategies to evaluate the impact of older data on malware identification and to determine whether the influence of that data differs across models. Finally, the effectiveness of the models will be assessed using synthetic datasets designed to emulate new malware, incorporating time-based features from the selected datasets.

This thesis aims to enhance the study of semi-supervised learning in Android malware. It addresses the following research questions:

- Does Multiview enable accurate labeling and prediction of malware?
- How does the amount of initial labeled data improve the identification of samples?
- How does the use of different machine learning algorithms impact detection?
- How does the balance of data classes impact the accurate identification of malware?

The aim of this research is to propose an innovative approach to using machine learning for the classification of Android malware by comparing various techniques. This study also examines different algorithms and their performance when applied to dynamic, static, and hybrid features. Additionally, it explores how the results vary when using balanced versus unbalanced datasets during the training of the models.

2. Background Information and Literature Review

This section aims to introduce the reader to the key concepts, theories, and technologies utilized in this thesis. Additionally, it will include a brief literature review of relevant concepts used during the research.

2.1 Machine Learning

Machine learning is a technique that utilizes statistical methods to predict or classify data based on previous samples that the respective algorithm has been trained on. This technique is widely used across various industries as it provides a certain level of confidence in predictions and classifications. Cybersecurity is one such field where machine learning and its various algorithms are already making an impact.

2.1.1 RandomForestClassifier

Random Forest (RF) is a well-known machine learning algorithm that belongs to the ensemble family of algorithms. It essentially involves the creation of several decision trees, where each tree makes a decision regarding a given data sample. The process begins with the Random Forest algorithm randomly generating several small datasets from the overall dataset; this process is known as bootstrapping [19]. Additionally, it randomly assigns features for each tree to consider, with the number of features being configurable through hyperparameters.

Once all the decision trees have been created, they each generate predictions based on the input sample. After this, all the predictions from the trees are combined through a majority vote, a process called aggregating. This method allows Random Forest to make classifications while being less sensitive to changes in the data, making it a great choice for model generalization.

Random Forest also allows users to configure various hyperparameters that can affect its performance. The first parameter is "n_estimators" which indicates the number of algorithms or in this case, the number of random trees that will be created [20]. This parameter is significant because more trees generally lead to more robust outputs, although they also increase the model's computational expense.

Next is the "n_features" parameter, which specifies the number of features to be used in each tree. The current options provided by the scikit-learn library include the square root of the total number of features, the logarithm base 2 of the total number of features, or the total number of features itself. Additionally, the "max_depth" parameter determines how deep the trees can grow, while the "max_leaf_nodes" parameter indicates how many nodes the tree can split into. This last parameter can also influence the depth of the trees.

2.1.2 MultiLayerPerceptron

A Multi-Layer Perceptron (MLP) is an artificial neural network that is part of the deep learning family of algorithms. The main architecture of the MLP consists of an input layer, which specifies the number of features in the input data; one or more hidden layers, which can be configured differently depending on the requirements of the model being trained; and an output layer, which provides the prediction outcome [21].

Each layer in the neural network has a specific type. In this research, the type used is known as a Dense layer. Its main characteristic is that it is fully connected, meaning that every neuron in the layer receives input from all the neurons in the preceding layer. Additionally, each layer must have a specified number of neurons.

A neuron is the fundamental computational unit of the network, essentially acting as a mathematical function that processes information and transmits data to other elements. The components of a neuron include:

1. The signal received by the neuron, which can come from another neuron or external data.
2. A value that is multiplied by the input; each input has an associated weight.
3. The neuron calculates the sum of all the inputs multiplied by their respective weights.
4. A value added to the summation.
5. A function that transforms the summation value based on its type; this value becomes the output of the neuron.

There are several activation functions [22], but the most common ones are ReLU, which is widely used in the hidden layers of the MLP algorithm and outputs the maximum value between 0 and the input value; Softmax, which is used for multi-class classification in the output layer; and the Sigmoid function, which is used for binary classification.

Finally, the term "epoch" refers to the process of fitting the data into the model and updating parameters such as weights and biases. The number of epochs for training a model can be configured as needed.

2.1.3 Class Imbalance

Class imbalance is a common issue in machine learning where one class has a significantly larger number of samples than the other class. This imbalance can lead to a bias toward the majority class, making it more challenging to accurately classify the minority class.

To address issues of class imbalance, several techniques can be employed:

1. This technique involves reducing the number of samples from the majority class.
2. In this approach, additional samples for the minority class are generated randomly based on the existing samples.
3. Both undersampling and oversampling techniques are applied together.

These methods help create a more balanced dataset, improving the performance of machine learning models.

2.1.4 Metrics

In the field of machine learning, particularly regarding classification tasks, evaluating model performance is essential. A variety of metrics offer valuable insights into a model's predictive abilities. Among these, accuracy, precision, recall, and the F1 score are fundamental measures to consider.

- Accuracy: reflects the overall correctness of a model's predictions. It is calculated as the ratio of correctly predicted instances to the total number of instances. While this metric appears straightforward, it can be misleading in cases where class distributions are imbalanced. For example, in a dataset with 95% of instances belonging to class A and only 5% to class B, a naive model that always predicts class A would achieve 95% accuracy, even though it fails to identify any instances of class B.
- Precision: also known as positive predictive value, quantifies the proportion of true positive predictions among all positive predictions made by the model. It answers the question: "Of all the instances the model labeled as positive, how many were actually positive?" High precision indicates that the model is reliable in its positive predictions, minimizing false positives.
- Recall: often referred to as sensitivity or the true positive rate, measures the model's ability to identify all actual positive instances. It addresses the question: "Of all

the actual positive instances, how many did the model correctly identify?" High recall signifies that the model effectively captures the majority of positive cases, minimizing false negatives.

- F1-score: is the harmonic mean of precision and recall, providing a single metric that balances both aspects. This score is especially useful in scenarios where both precision and recall are important. An F1 score of 1 indicates perfect precision and recall, while a score of 0 suggests that there is either perfect precision or perfect recall, but not both.

The choice of metrics depends significantly on the specific application and the relative importance of precision and recall. In malware detection, a low recall can indicate that a significant amount of malware is being missed, as it is incorrectly labeled as negative. This means the model has a high number of false negatives. Therefore, it is critical to select the metrics that hold greater importance for the research accurately.

2.2 Machine Learning Techniques

Machine learning has different types of techniques that aim to solve various kinds of problems.

2.2.1 Supervised Learning

Supervised learning is a technique that works with labeled data. This means that the learning process is guided by data that has been previously labeled by experts or individuals with sufficient knowledge, such as in image recognition tasks involving the identification of dogs and cats.

The objective of the algorithm is to learn the underlying relationship between inputs and outputs, enabling it to make accurate predictions on new, unseen data. The main characteristics of supervised learning include:

- Labeled Data: The availability of labeled data is crucial for supervised learning. Each data point in the training set is associated with a known output or target variable.
- Predictive Modeling: The primary goal of supervised learning is to build predictive models that can accurately map inputs to outputs.
- Learning from Examples: The algorithm learns by iteratively adjusting its internal parameters based on the errors observed between its predictions and the actual labels in the training data.

Supervised learning is widely used in classification and regression problems. While it can be highly accurate, one of its biggest challenges is that it can be expensive to implement. This is primarily because it requires labeled data, which can be costly to obtain. In some cases, labeling must be done by experts in specific fields, such as cybersecurity or medicine.

2.2.2 Semi Supervised Learning

This technique primarily utilizes information from labeled data to predict or assign labels to unlabeled data. Once the unlabeled data is labeled, it can be used to improve the performance of supervised learning models, enabling them to learn from a larger dataset. Its main characteristics are:

1. Combination of Data Types: It capitalizes on both labeled and unlabeled data, addressing the limitations of purely supervised and unsupervised learning methods.
2. Cost-Effectiveness: By reducing the need for large amounts of labeled data, which can be expensive and time-consuming to collect, this technique allows for more efficient data usage.
3. Improved Model Performance: The inclusion of additional labeled data through labeling the unlabeled examples can lead to better generalization and improved accuracy of machine learning models.
4. Flexibility: This approach can be adapted for a variety of tasks, including classification and regression, across different domains.

Co-Training

Co-training is a well-known SSL technique that relies on the assumption that there are multiple, conditionally independent views of the data. Each view, which comprises a distinct set of features, should be sufficient for accurate classification on its own [23].

The co-training algorithm starts by training two classifiers, each utilizing a different view of the labeled data. After this initial training, the classifiers iteratively label the unlabeled instances. In each iteration, each classifier predicts the class probabilities for all unlabeled samples. The predictions with the highest confidence—those with the highest probabilities are then selected and added to the labeled dataset. Concurrently, the pool of unlabeled instances is updated by removing these newly labeled samples. This iterative process continues until either the pool of unlabeled instances is exhausted or a predefined convergence criterion is satisfied, effectively providing the two trained classifiers with new labeled data [24].

By taking advantage of the complementary information found in multiple views, co-training can effectively utilize unlabeled data to improve model performance, especially when labeled data is limited. This technique has shown success across various domains, including text classification, image recognition, and web page categorization. the main consideration of the co-training are:

- View Independence: The conditional independence between views is critical for the success of co-training. If this assumption is violated, the performance may decline.
- View Sufficiency: Each view should contain enough information to classify instances accurately on its own.
- Label Noise: Carefully selecting confidently labeled instances is essential. If instances are incorrectly labeled, this can introduce noise and degrade the model's performance.

2.3 Related Work

Semi-supervised learning has emerged as a promising approach for malware detection, particularly in scenarios where labeled data is limited. Numerous studies have investigated its application in this field, emphasizing different malware types and various detection methods.

Gao et al. [25] were among the first to apply co-training for classifying image-based malware. By converting binary files into grayscale images and using the Local Binary Patterns (LBP) operator for feature extraction, they implemented Random Forest, KNN, and logistic regression as base classifiers. Abdelmonem et al. [26] they enhanced this method by introducing the Model, a semi-supervised deep neural network. They integrated the Model with CNN architectures like ResNet50 and LeNet8 to enhance performance.

Niu et al. [27] proposed ADESSA, a semi-supervised learning algorithm for detecting network traffic anomalies. They used the KDD'99 dataset for initial classifier training and collected unlabeled data from a cyber-physical system (CPS) network. ADESSA employed a co-training-like approach, utilizing three base classifiers to generate labels and probabilities. The prediction with the highest probability was selected, mimicking co-training without requiring different views.

Appice et al. proposed Clustering-Aided Multi-View Classification (CAMVC) [28] for Android malware detection. This method involved partitioning training samples into distinct clusters and using classifiers trained on one cluster to label data in other clusters. Stacking was then used to combine the predictions. Kamal et al. [29] introduced NELMA, another co-training-based approach for Android malware detection. They utilized the Android Malware Dataset (AMD) and validated benign APK files using VirusTotal. Features such as permissions and API calls were employed, while classifiers like SVM, decision trees, Random Forest, KNN, and Naive Bayes were used. Logistic regression was applied for knowledge integration.

The experiments assessed the performance of the proposed methods. Common evaluation metrics included accuracy, precision, recall, and F1-score, with comparisons made to both supervised learning baselines and other semi-supervised learning techniques.

Semi-supervised learning has demonstrated considerable promise for malware detection, especially when there is a lack of labeled data. Future research could focus on developing new semi-supervised learning algorithms, exploring the integration of domain-specific knowledge, and tackling the challenges posed by evolving malware threats.

The literature review suggests that previous studies have implemented the Multiview approach using various algorithms to compare results. However, a gap appears in the literature when it comes to comparing Multiview with other machine learning techniques, which justifies the focus of this thesis.

3. Methodology

This section will show the methods, procedures, and tools used to conduct the research. It does include the selection of the dataset and how it was created to create all the necessary environments for the execution of the training experiment and all the other variants that will presented later in the document.

The thesis uses a quantitative research approach since the main focus of machine learning is to analyze data to uncover patterns, trends and relationships.

3.1 Data

Machine learning necessitates intensive use of data to train and evaluate algorithms. The quality and nature of this data significantly influence the algorithm's performance.

For this research, the selected dataset must meet several specific criteria.Firstly, the dataset should be divided into distinct subsets based on views. This means that it must be capable of being partitioned into two or more sets of features, where each set is conditionally independent from the others. In other words, the sets should represent different aspects of the data, complementing each other without being redundant or conveying the same information. Secondly, the dataset should contain a feature indicating the date or time the samples were collected. This is crucial, as one of the primary objectives of the investigation is to explore methods capable of detecting novel malware. Finally, the dataset should be accurately labelled, as one of the aims of this study is to compare the efficacy of semi-supervised learning with that of fully supervised learning.

3.1.1 Dataset Selection

The dataset selected for the research is the KronoDroid [30] Dataset Real Devices variant because the dataset meets all the requirements needed for the research. This can be seen in its characteristics:

- The dataset includes a set of dynamic features known as system calls.
- The dataset includes a feature set known as permissions, which falls under the category of static features.
- The samples include two timestamp features: "EarliestModDate," representing the

earliest modification found within the application's inner file, and "HighestMod-Date," indicating the latest modification timestamp across all files that make up the application.

- Contains a total of 78,137 samples that are labelled in a binary way as needed in the experiment.
- The dataset samples cover the time period from 2008 to 2020.

The system calls feature set includes 288 numeric features, while the permissions feature set consists of 166 binary features that indicate whether the standard Android permissions were requested by the app. Additionally, the dataset is organized using various types of timestamps for research purposes. The "HighestModDate" feature will be used to sort the dataset by date and time, as it has the highest number of valid timestamps for both malware and benign applications. Finally, the "Malware" column tells if the sample has been labelled malware. This can be seen as 1 for positive malware or 0 for negative malware.

3.1.2 Dataset Processing

To apply co-training in a non-stationary environment, the dataset must first be properly encoded and transformed. The initial step involves cleaning the dataset to remove samples with incorrect dates; specifically, some samples mistakenly contain the timestamp "1980," which needs to be eliminated. Next, any columns that are not part of the feature set related to system call permissions must be removed from the dataset.

Once the data has been successfully cleaned, we should create scenarios for the various experiments. The evaluation period will span from the beginning of 2012 to the end of 2018, resulting in the creation of seven datasets one for each year. Datasets from 2012 to 2017 will be used to train the models, while datasets from 2013 to 2018 will be designated for testing the previously created models.

3.1.3 Dataset Balance

The dataset has an issue with the balance between the "Malware" and "Non-Malware" classes. The distribution of these classes is uneven, creating an imbalance that may result in bias toward the majority class. This can lead to overfitting in the majority class and a failure to accurately classify the minority class. This imbalance is illustrated in the table below. 1 where a significant difference between positive and negative classes exists, with the positive or "Malware" class being the majority class.To address this issue during the experiment,

	Dataset	Non-Malware	Malware
0	2012	342	7567
1	2013	489	7490
2	2014	632	8015
3	2015	721	1450
4	2016	743	2485
5	2017	652	4861
6	2018	780	4184

Table 1. Class Balance

we used an oversampling technique known as SMOTE [31] (Synthetic Minority Oversampling Technique). This process involves creating new samples for the minority class using the k-nearest neighbors algorithm. For each sample in the minority class, a synthetic point is generated by connecting it to its nearest neighbor. This method has been applied to all six datasets that will be used for training the models in this experiment.

3.2 Experiments

For this research, we developed three types of scenarios to determine how the data will fit into the model. In each scenario, a new model is created for each year using data from 2012 to 2018. These models are then tested on the data from the following year. For example, the model created from the 2012 data is applied to the 2013 data, and this process continues for each subsequent year.

3.2.1 Normal Fit Scenario

What sets this scenario apart from others is that the first model, created in 2012, was developed using the same dataset as the models in the other scenarios. In contrast, the models developed between 2013 and 2017 were trained on a smaller dataset because this scenario does not incorporate data from previous years for model development. This distinction is significant for the study, as it emphasizes the impact of using a smaller dataset for model training compared to models trained on larger datasets.

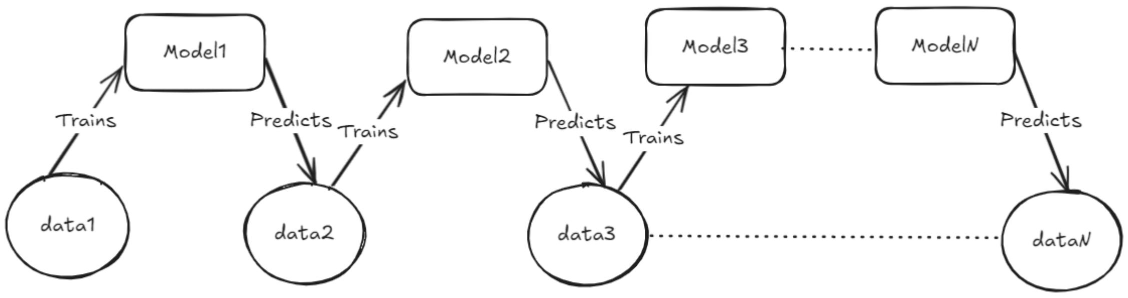

Figure 2. Normal Fit Scenario description

In the figure 2, "data1" represents the dataset from the year 2012, which will be used to create "model1." This model will then be tested on the following year's data, referred to as "data2," which corresponds to 2013. The process will continue with "data2" being used to train "model2," and this pattern will persist until reaching "dataN," representing the data from the year 2018. In this experiment, "modelN" will be tested using "dataN," which was developed from "dataN-1," although the latter is not shown in the image. It is important to note this progression throughout the experiment.

3.2.2 Batch Fit Scenario

The importance of the batch fit scenarios lies in the fact that each model created will utilize data from both the current year and previous years for training. This means that in the first year, the model will only have access to data from 2012, while in the final year, it will incorporate data from 2012 through 2017. This scenario allows researchers to investigate the advantages of using a larger dataset with the various techniques and models that will be examined in the study.

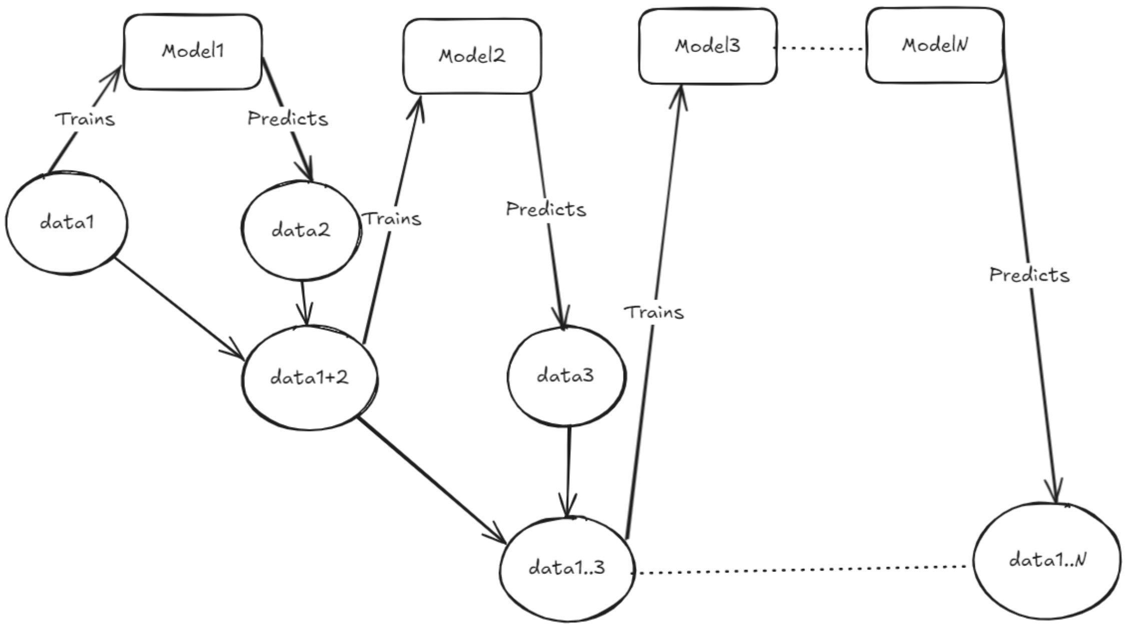

Figure 3. Batch Fit Scenario description

The figure 3 illustrates the fitting and testing process for models in the Batch scenario. Initially, the model will be fitted only with data from the year 2012 and tested with data from 2013. However, in subsequent models, unlike in the Normal fit scenario, the training process changes.

In the image, "data1+2" refers to the training data for "Model 2," which is designed for the year 2013. This model will be fitted using data from both 2012 and 2013, effectively combining these two years into a single dataset for training.

This pattern will continue until reaching "Model N," which will incorporate all data from the previous years. In this particular experiment, that means Model N will use data from 2012 through 2017 and will be tested with data from the year 2018.

3.2.3 Online Fit Scenario

The online fit scenario for co-training involves developing a single model that is updated annually with new data. In this study, an initial model will be created using data from the year 2012. This model will then be tested with data from 2013. For subsequent experiments, no new model will be created; instead, the model from 2012 will be updated using the data from 2013. The significance of this scenario is similar to that of the batch method, but instead of creating a new model each year, the initial model will be updated with new data. This approach allows the study to explore how a model behaves when exposed to a consistent stream of data.

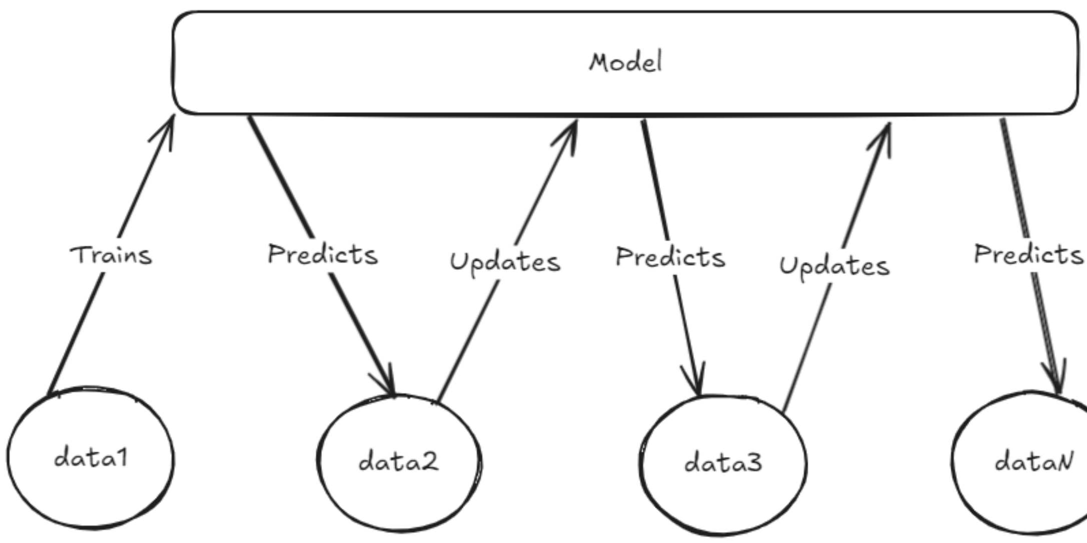

Figure 4. Online Fit Scenario description

As shown in Figure 4, the initial model is created using "data1," which represents data from the year 2012. This model is then tested on "data2," which corresponds to the year 2013. In this online scenario, the model will not be retrained each year; instead, its parameters will be updated. Therefore, the figure illustrates the model as one that persists over time and undergoes updates annually.

3.3 Machine Learning Techniques

Machine learning encompasses various techniques that improve model prediction accuracy and allow models to perform tasks beyond data prediction and classification. For this research, two techniques of machine learning have been utilized and will be explained next.

3.3.1 Supervised Learning

Supervised Learning involves using labeled data to train new models, emphasizing the importance of such data. This study applies various approaches to Supervised Learning. Specifically, we will test the dataset SL technique using 25%, 50%, and 75% of the data to evaluate how different volumes of data affect model performance. For each of these data volumes, we will utilize different features.

We will start by examining Permissions, which include the static set of features from the dataset. Next, we will analyze System Calls, which represent the dynamic set of features. Finally, we will explore a Hybrid approach that combines both Permissions and System

Calls. This comparison is essential as it will help determine which perspective offers better model performance and how these two views complement each other in the Hybrid approach.

3.3.2 Semi Supervised Learning

Semi-Supervised Learning is a technique that utilizes a small amount of labeled data to assign labels to a larger set of unlabeled data. The process begins by creating one or more classifiers using the available labeled data and selecting relevant features for the task.

Once the initial classifiers are established, the algorithm iteratively processes batches of unlabeled data. In each iteration, the classifiers label a portion of the unlabeled samples. This portion can be set as a parameter in the algorithm, or if left at the default, it will select both positive and negative samples to be labeled. The samples chosen for labeling are those with the highest prediction probabilities; that is, the model assigns each sample a class and a confidence percentage for that prediction. The samples with the highest confidence for both the positive and negative classes are picked during each iteration.

After labeling the samples, the classifiers are updated or retrained with the newly labeled data. This cycle continues until either all unlabeled data has been labeled or a specified iteration limit is reached.

In this research, we will implement a single-view approach to semi-supervised learning, as well as a multi-view algorithm known as co-training.

Single View Semi-Supervised Learning

As mentioned earlier, semi-supervised learning (SSL) relies on labeled data to assign labels to previously unlabeled samples. In this research, we implemented single-view SSL in two ways. The first method utilizes only system calls for labeling the samples. This means that samples with the highest confidence levels—determined by the classifier created from the system calls—will be labeled and subsequently added to the updated classifier. The same process will be applied to permissions.

This step in the research is crucial because it enables us to compare the performance of SSL using different views or sets of features. By doing so, we can better understand which approach has a greater impact on data labeling. The outcome of this algorithm will be two classifiers, each trained with the new labeled data generated by the algorithm that used a single view.

Co-Training : Multi-View

Co-training is a semi-supervised learning (SSL) algorithm that facilitates the identification and labeling of samples using multiple sets of features. In this study, the features used are permissions and system calls. Two classifiers will be created, one for each set of features. After the classifiers are established, each will select the prediction with the highest level of reliability for both classes.

In this SSL approach, the newly labeled data will be incorporated to update the classifiers, allowing the process of labeling data to continue iteratively. The multi-view algorithm will enable us to assess the performance of a model that considers different perspectives for labeling the data. The outcome of this algorithm will be two classifiers trained with the newly labeled data generated from both sets of features.

3.4 Machine Learning Models

Choosing the right machine learning models for the experiment is crucial. The selected models should be capable of implementing both supervised and semi-supervised learning techniques. It's important that they are non-linear models, as this characteristic allows them to effectively tackle complex problems. Additionally, the models should provide probabilities indicating the likelihood that a sample belongs to class 0 or class 1. Finally, using algorithms from different families within machine learning will offer a more comprehensive view of the results.

3.4.1 RandomForest

The Random Forest classifier (RF) [32] is a non-linear model that belongs to the ensemble learning family of algorithms. It is chosen for its ability to perform both supervised and semi-supervised learning, with the Random Forest library being used for implementation. The scikit-learn library [20] offers an option to obtain the probability of a sample belonging to class 0 or class 1. Additionally, to achieve the best possible results when training this model, we tuned the model's hyperparameters.

3.4.2 Multilayer Perceptron

The Multi-Layer Perceptron (MLP) model, which belongs to the Neural Networks family, is a non-linear model capable of performing supervised and semi-supervised learning. The model has been optimized in terms of both the number of epochs and the number of hidden layers. This will be discussed further in the results section. The Multi-Layer Perceptron model has been implemented using the Keras library [33].

3.5 Optimization

3.5.1 Random Forest

The Random Forest model was optimized using the GridSearchCV technique [34]. This optimization was conducted on separate views of the dataset as well as on the dataset as a whole, reflecting the study's focus on exploring the performance of permissions, system calls, and a combination of both. As a result, multiple tree variants were created, each with different hyperparameters. The GridSearchCV algorithm employs a dictionary containing lists of various hyperparameter values for each type of hyperparameter. This can be observed in the code referenced in Algorithm 1.

The results from the GridSearchCV revealed the following optimized hyperparameters:

- For the System Calls model, the best hyperparameter found was "n_estimators," which indicates the number of trees created. This parameter was set to 200. Additionally, the "max_features" parameter was set to "None," meaning that the entire set of features available for System Calls would be utilized in the training of each tree.
- For the Permissions model, the "n_estimators" parameter was set to 50, while "max_features" was set to "log2." This means that the number of features used for each created tree corresponds to the logarithm base 2 of the total number of features available, with features being randomly assigned for each trained tree.
- For hybrid case, the "n_estimators" parameter was set to 150, and the "ma_features" parameter was also set to its default value, which means the total number of features would be used in the training of each created tree as well.

Algorithm 1: GridSearchCV Algorithm

```
Data: $X\_train$ : DataFrame, $y\_train$ : Array
Result: $best\_hyperparameters$ : String
$param\_grid = \{$
'n_estimators': [25, 50, 100, 150, 200, 250, 300, 600],
'max_features': ['sqrt', 'log2', None],
'max_depth': [None, 3, 6, 9],
'max_leaf_nodes': [None, 3, 6, 9, 12, 15],
}
$grid\_search = GridSearchCV(estimator = RandomForestClassifier(),$
$param\_grid = param\_grid)$
$grid\_search.fit(X\_train, y\_train)$
$best\_hyperparameters = str(grid\_search.best\_params)$
return $best\_hyperparameters$
```

3.5.2 Adaptative Random Forest

The Adaptive Random Forest (ARF) comes from the River library, and even though it behaves similarly to the Random Forest, it requires its own model optimization. Optimization for the ARF has been conducted for different tree variants: only permissions, only system calls, and both system calls and permissions. The goal of this optimization was to identify the best number of estimators for each variation of the model. This approach was chosen because using other hyperparameters made model creation excessively costly. The outcome of it is:

- For the System Calls model, the best hyperparameter found was "n_estimators," which indicates the number of trees created. This parameter was set to 20.
- For the Permissions model, the "n_estimators" parameter was set to 30.
- For hybrid case, the "n_estimators" parameter was set to 20.

3.5.3 MultiLayer Perceptron

The MultiLayer Perceptron (MLP) model was optimized through two main steps. First, we tested various combinations of hidden layers during the model's compilation to identify the best configuration. This involved creating models with different numbers of Dense layers, each having an incremental number of neurons. We continued adding neurons until we reached the maximum limit. For this optimization, we used limits of 1024, 512, 256, 128, 64, 16, and 8. The number of neurons was doubled when increasing and divided by two when decreasing from the previous layer until we reached the output layer.

After determining the optimal combination of layers, we evaluated the number of epochs necessary to achieve the best results. This evaluation was conducted separately for permissions, system calls, and the combination of both perspectives.

The optimized configuration for system calls is presented in Algorithm 1, while the optimization for permissions can be found in Algorithm 2. The combined optimization for both categories is detailed in Algorithm 3.

```
model = keras.Sequential(
    [
        Dense(52, activation='relu', input_dim=287),
        Dense(32, activation='relu'),
        Dense(64, activation='relu'),
        Dense(128, activation='relu'),
        Dense(256, activation='relu'),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(16, activation='relu'),
        Dense(8, activation='relu'),
        Dense(1, activation='sigmoid')
    ]
)

model.compile(loss='binary_crossentropy', optimizer='adam')

model.fit(X_train_resampled, y_train_resampled, epochs=20)
```

Listing 1. Systems Calls Optimized MultiLayer Perceptron

```
model = keras.Sequential(
    [
        Dense(52, activation='relu', input_dim=166),
        Dense(8, activation='relu'),
        Dense(1, activation='sigmoid')
    ]
)

model.compile(loss='binary_crossentropy', optimizer='adam')

model.fit(X_train_resampled, y_train_resampled, epochs=10)
```

Listing 2. Permissions Optimized MultiLayer Perceptron

```
model = keras.Sequential(
    [
        Dense(52, activation='relu', input_dim=453),
        Dense(32, activation='relu'),
        Dense(64, activation='relu'),
        Dense(16, activation='relu'),
        Dense(8, activation='relu'),
        Dense(1, activation='sigmoid')
    ]
)

model.compile(loss='binary_crossentropy', optimizer='adam')

model.fit(X_train_resampled, y_train_resampled, epochs=30)
```

Listing 3. Permissions + System Calls Optimized MultiLayer Perceptron

3.5.4 Co-Training

The co-training algorithm has parameters that can be optimized for three scenarios: Normal, Batch, and Online fitting. This optimization process will be conducted systematically. First, we will determine the optimal value for "k," which represents the number of iterations for labeling the unlabeled samples. Once we identify the best value for "k," we will proceed to test the parameters "u," "p," and "n." These parameters correspond to the unlabeled pool, the number of positive samples labeled per iteration, and the number of negative samples labeled per iteration, respectively.

The results of all attempted variations for optimizing the co-algorithm are presented in Table 2. It was observed that adjusting the hyperparameters does not significantly impact the variation in model metrics. Therefore, in the following experiments, the default settings of the algorithm will be used.

Parameters (k, U, p, n)	Accuracy	Precision	Recall	F1-score
30, 74, 2, 2	0.991235	0.995318	0.987175	0.991230
60, 74, 2, 2	0.991512	0.995431	0.987617	0.991509
30, 74, 10, 10	0.991124	0.995207	0.987065	0.991119
90, 74, 2, 2	0.991290	0.995429	0.987175	0.991285
120, 74, 2, 2	0.990902	0.995315	0.986512	0.990894
20, 74, 2, 2	0.991346	0.995651	0.987065	0.991339
10, 74, 2, 2	0.991235	0.995429	0.987065	0.991229
5, 74, 2, 2	0.991179	0.995539	0.986844	0.991172
90, 160, 40, 40	0.991179	0.995207	0.987175	0.991175
90, 160, 70, 70	0.991734	0.995765	0.987728	0.991730
60, 74, 10, 10	0.991179	0.995097	0.987286	0.991176
60, 74, 5, 5	0.991124	0.994986	0.987286	0.991121
60, 74, 3, 3	0.991401	0.995430	0.987396	0.991397
60, 74, 10, 10	0.991457	0.995652	0.987286	0.991451

Table 2. Co-training optimization results

3.6 Evaluation of Results

The CoTraining technique involves a preliminary step before evaluating the outcomes using metrics. In both experiments, Random Forest (RF) and Multi-Layer Perceptron (MLP), the classification of a sample will be determined based on the agreement between their respective Permissions classifier and System Calls classifier, which are derived from the views regarding the predicted class. However, if there is a disagreement between the classifiers, the probabilities assigned to each class will be summed, and the class with the highest total will be selected for labeling.

The evaluation metrics used in this study are accuracy, recall, precision, and F1-score. Among these metrics, recall is the most important, as it indicates the number of true positives identified by the model. Precision, which measures the true negatives classified by the model, is also crucial. Both metrics are significant because the study aims to improve malware detection; a higher recall means that more malware is being detected, while precision is equally vital since a high number of false positives can be problematic.

Accuracy provides a general overview of the algorithm's performance in classification, whereas the F1-score assesses the balance between recall and precision. A higher F1-score indicates a better balance between these two metrics, which is another desired outcome.

3.7 Project Repository

All the code of the project can be accessed from GitHub [35].

4. Results

In this section, we present the results obtained from various experiments, organized in tables. Each table displays the results of either the Random Forest (RF) or Multi-Layer Perceptron (MLP) methods, trained with either balanced or unbalanced data across different scenarios.

The tables include the following rows:

- Permissions Learning [25, 50, 75]% it refers to the Supervised Learning (SL) applied to the respective 25%, 50%, and 75% subsets of the Permission data.
- System Calls Learning [25, 50, 75]% it indicates the SL applied to the 25%, 50%, and 75% subsets of the System Calls data.
- Hybrid Learning [25, 50, 75]% this represents the SL applied to the combined 25%, 50%, and 75% subsets of both Permissions and System Calls.
- Co-training Multiview this row shows the results of the Multiview algorithm.
- Semi-Supervised System Calls it displays the results from the SSL algorithm, where System Calls were used for labeling unlabeled samples.
- Semi-Supervised Permissions which indicates the results from another SSL algorithm, where Permissions were utilized for the labeling process.

All of these results are crucial for evaluating the outcomes and drawing meaningful conclusions.

4.1 Random Forest with Balanced Data

The table **??** presents the results of the Random Forest (RF) model trained with balanced data across three different scenarios: Normal, Batch, and Online.

In the Normal scenario, the "Hybrid Learning 75%" experiment achieved the best overall metrics, with a recall of 89.52%, accuracy of 90.30%, and an F1-score of 93.79%. Notably, the highest precision in this scenario was recorded by the "CoTraining Multiview" experiment, which had a value of 98.97%. Additionally, the overall performance of the Supervised Learning (SL) experiments suggests that the hybrid approaches are the most effective, particularly when analyzing recall. In contrast, the System Calls experiments exhibited the lowest recall, with the "System Calls Learning 50%" experiment achieving a value of only 67.86%. For the Semi-Supervised Learning (SSL) experiments, the Co-training option demonstrated the best performance.

In the Batch scenario, "Hybrid Learning 75%" again produced the best results, achieving a precision of 99.20%, an accuracy of 94.71%, and an F1-score of 96.75%. The highest recall in this scenario was attained by the "CoTraining Multiview" experiment, which reached a recall of 97.76%. Once again, the hybrid experiments outperformed the others in the SL category, while the System Calls experiments exhibited the lowest performance. The "CoTraining Multiview" experiment once more showcased the best recall among the SSL experiments.

In the Online scenario, the highest metrics were produced by the "System Calls Learning 25%" experiment, which achieved a remarkable recall of 98.93%. The "Hybrid Learning 25%" experiment achieved an accuracy of 89.27% and an F1-score of 93.01%. Meanwhile, the best precision was recorded by the "Semi-Supervised Permissions" experiment, reaching a value of 99.97%. In this scenario, it is difficult to determine which SL experiment performed best based on recall. However, when sorted by precision, the hybrid SL techniques consistently performed better. For the SSL techniques, the results show a decline compared to the Batch scenario but are somewhat improved compared to the Normal scenario.

When comparing the three scenarios, it is evident that the Batch scenario consistently achieves the best overall results, as indicated by its higher F1-score and accuracy compared to the others. Interestingly, the Online scenario demonstrates strong recall, particularly in the "System Calls Learning 25%" experiment, which was conducted with a smaller dataset. Moreover, all SL experiments performed exceptionally well in the Online scenario, with results surpassing 97%.

Finally, regarding the MultiView results, the highest recall of 97.76% was achieved in the Batch scenario. This scenario also yielded the best F1-score and accuracy for the MultiView algorithm, while the highest precision was recorded in the Normal scenario.

ML Technique	Accuracy	Precision	Recall	F1 Score
Normal				
Permissions Learning 25%	82.70%	97.82%	81.61%	88.36%
System Calls Learning 25%	72.61%	96.96%	68.05%	79.18%
Hybrid Learning 25%	88.32%	98.72%	87.30%	92.44%
Permissions Learning 50%	86.48%	97.81%	85.81%	91.13%
System Calls Learning 50%	72.61%	97.15%	67.86%	79.01%
Hybrid Learning 50%	89.89%	98.76%	89.04%	93.52%
Permissions Learning 75%	82.22%	98.09%	80.94%	87.96%
System Calls Learning 75%	74.37%	97.23%	70.11%	80.71%
Hybrid Learning 75%	**90.30**%	98.74%	**89.52**%	**93.79**%
CoTraining MultiView	86.56%	**98.97**%	84.93%	90.85%
SemiSupervised SystemCalls	85.26%	98.38%	82.98%	89.84%
SemiSupervised Permisions	83.90%	98.41%	81.33%	88.83%
Batch				
Permissions Learning 25%	87.68%	97.81%	87.50%	92.04%
System Calls Learning 25%	81.24%	97.62%	78.60%	86.58%
Hybrid Learning 25%	92.97%	98.89%	92.76%	95.64%
Permissions Learning 50%	89.40%	98.14%	89.21%	93.22%
System Calls Learning 50%	83.72%	98.01%	81.34%	88.51%
Hybrid Learning 50%	94.47%	99.03%	94.34%	96.60%
Permissions Learning 75%	89.69%	98.28%	89.39%	93.42%
System Calls Learning 75%	85.76%	97.73%	84.10%	90.14%
Hybrid Learning 75%	**94.71**%	**99.20**%	94.51%	**96.75**%
CoTraining MultiView	91.97%	92.50%	**97.76**%	94.98%
SemiSupervised SystemCalls	91.89%	93.14%	96.80%	94.88%
SemiSupervised Permisions	92.09%	93.40%	96.75%	95.00%
Online				
Permissions Learning 25%	81.49%	79.89%	98.11%	87.63%
System Calls Learning 25%	63.48%	55.62%	**98.93**%	69.48%
Hybrid Learning 25%	**89.27**%	88.46%	98.29%	**93.01**%
Permissions Learning 50%	85.86%	84.74%	97.91%	90.75%
System Calls Learning 50%	68.87%	62.98%	98.39%	75.74%
Hybrid Learning 50%	86.00%	84.89%	98.65%	91.03%
Permissions Learning 75%	81.07%	79.40%	97.99%	87.32%
System Calls Learning 75%	68.13%	62.02%	98.36%	74.98%
Hybrid Learning 75%	88.86%	87.85%	98.63%	92.81%
CoTraining MultiView	87.34%	98.29%	87.70%	92.53%
SemiSupervised SystemCalls	86.24%	99.87%	85.85%	92.10%
SemiSupervised Permisions	83.85%	**99.97**%	83.86%	90.91%

Table 3. Random Forest Performance Metrics Balanced Data

4.2 Random Forest with Unbalanced Data

The table 4 presents the results of the Random Forest (RF) model, which was trained using unbalanced data across three different scenarios: Normal, Batch, and Online.

In the Normal scenario, the "Hybrid Learning 75%" approach achieved the best accuracy and F1-score, recording values of 91.72% and 94.75%, respectively. The "Semi-Supervised System Calls" experiment obtained the highest precision at 98.60%, while the "Permissions Learning 25%" experiment achieved the best recall, with a value of 91.94%. The results indicate that, in this scenario, training with Hybrid data performs slightly better than training with Permissions data, while System Calls data yielded the lowest performance. For the semi-supervised learning (SSL) technique experiments, results were nearly identical, with the Permissions experiment emerging as the top performer, achieving a recall of 83.72%.

In the Batch scenario, the "Hybrid Learning 50%" experiment produced the best results for both accuracy (95.41%) and F1-score (97.19%). The "Hybrid Learning 75%" experiment attained the highest precision at 98.91%, while the "Co-Training Multiview" experiment recorded the best recall at 98.57%. Within the supervised learning (SL) technique, the Hybrid data performed the best, while both Permissions and System Calls data showed lower performance, with values so close that they were difficult to differentiate. For the SSL technique experiments, all performed nearly equally well, but "Co-Training Multiview" had a slight edge with a recall of 98.57%.

In the Online scenario, the "Hybrid Learning 75%" experiment achieved the highest metrics, with an accuracy of 92.22%, precision of 92.83%, recall of 97.37%, and an F1-score of 94.99%. In this scenario, the Hybrid approach was clearly the best performer for the SL technique. For the SSL technique experiments, the differences in performance were minimal, but "Co-Training Multiview" stood out with a recall of 95.87%.

Comparing the three scenarios, it is clear that the Batch scenario consistently achieves the best overall results, with higher accuracy, precision, recall, and F1-score compared to the other scenarios. The Normal scenario shows the lowest performance overall, surpassing the Online scenario only in accuracy, with a value of 98.60% compared to 92.83%. The Online scenario, however, produced notable results, with the "Hybrid Learning 75%" experiment demonstrating the best metrics across all categories within that scenario.

Lastly, regarding the MultiView results, the highest recall of 98.57% was achieved in the Batch scenario. This scenario also yielded the best F1-score and accuracy for the MultiView algorithm, while the highest precision was recorded in the Normal scenario.

ML Technique	Accuracy	Precision	Recall	F1 Score
Normal				
Permissions Learning 25%	91.03%	96.80%	**91.94**%	94.26%
System Calls Learning 25%	81.27%	94.38%	81.55%	87.36%
Hybrid Learning 25%	91.10%	97.87%	91.18%	94.33%
Permissions Learning 50%	89.83%	96.38%	90.99%	93.50%
System Calls Learning 50%	81.30%	95.76%	79.99%	86.92%
Hybrid Learning 50%	91.25%	98.36%	91.01%	94.45%
Permissions Learning 75%	88.43%	97.48%	88.45%	92.59%
System Calls Learning 75%	83.07%	96.00%	81.79%	88.15%
Hybrid Learning 75%	**91.72**%	98.42%	91.46%	**94.75**%
CoTraining MultiView	84.05%	98.13%	81.74%	88.98%
SemiSupervised SystemCalls	85.34%	**98.60**%	83.08%	90.01%
SemiSupervised Permisions	85.85%	98.50%	83.72%	90.36%
Batch				
Permissions Learning 25%	90.75%	97.32%	91.38%	94.16%
System Calls Learning 25%	87.89%	95.49%	88.89%	92.01%
Hybrid Learning 25%	94.43%	98.34%	94.99%	96.60%
Permissions Learning 50%	90.49%	97.66%	90.74%	93.94%
System Calls Learning 50%	89.59%	96.33%	90.24%	93.14%
Hybrid Learning 50%	**95.41**%	98.57%	95.90%	**97.19**%
Permissions Learning 75%	89.38%	97.94%	89.39%	93.22%
System Calls Learning 75%	89.87%	96.61%	90.30%	93.29%
Hybrid Learning 75%	95.17%	**98.91**%	95.35%	97.06%
CoTraining MultiView	91.69%	91.67%	**98.57**%	94.91%
SemiSupervised SystemCalls	91.78%	92.50%	97.47%	94.87%
SemiSupervised Permisions	91.37%	92.17%	97.42%	94.65%
Online				
Permissions Learning 25%	90.72%	91.86%	96.09%	93.88%
System Calls Learning 25%	83.41%	87.53%	90.83%	89.07%
Hybrid Learning 25%	91.39%	91.88%	97.33%	94.47%
Permissions Learning 50%	90.80%	92.63%	95.45%	93.98%
System Calls Learning 50%	81.34%	83.43%	92.59%	87.67%
Hybrid Learning 50%	91.75%	91.89%	96.81%	94.20%
Permissions Learning 75%	91.53%	92.41%	96.61%	94.41%
System Calls Learning 75%	82.28%	85.04%	91.97%	88.29%
Hybrid Learning 75%	**92.22**%	**92.83**%	**97.37**%	**94.99**%
CoTraining MultiView	87.46%	88.40%	95.87%	91.92%
SemiSupervised SystemCalls	87.43%	89.27%	94.98%	91.99%
SemiSupervised Permisions	72.91%	71.20%	92.88%	80.17%

Table 4. Random Forest Performance Metrics No Balanced Data

4.3 Multilayer Perceptron with balanced Data

The table 5 presents the results of the MLP model trained using balanced data across three different scenarios: Normal, Batch, and Online.

In the Normal scenario, the "Permissions Learning 25%" approach achieved the highest accuracy at 92.21% and the best F1-score at 94.89%. Meanwhile, the "Hybrid Learning 25%" approach obtained the highest recall, with a value of 99.97%, while "Hybrid Learning 75%" recorded the highest precision at 98.50%.

In the Batch scenario, the "Permissions Learning 50%" method achieved the highest accuracy at 94.23% and an F1-score of 96.44%. The "System Calls Learning 25%" experiment recorded the best recall at 100%, while the highest precision was obtained from the "Permissions Learning 75

In the Online scenario, the "Permissions Learning 50%" experiment yielded the best overall metrics, achieving an accuracy of 94.40%, a precision of 98.83%, and an F1-score of 95.25%. Notably, the recall for the experiments "System Calls Learning 25%", "System Calls Learning 50%", and "System Calls Learning 75%" all tied at 100%.

When comparing the three scenarios, it is difficult to determine a clear winner, as the F1-scores across all scenarios are quite similar. Additionally, the 100% recall obtained in both the Batch and Online scenarios may indicate a symptom of overfitting in those cases.

Finally, regarding the MultiView results, the highest recall of 96.23% was achieved in the Batch scenario. However, for all other metrics, the Online scenario demonstrated better performance.

ML Technique	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Normal				
Permissions Learning 25%	84.87%	98.67%	83.27%	90.14%
System Calls Learning 25%	61.37%	95.98%	54.30%	68.24%
Hybrid Learning 25%	65.75%	98.18%	60.51%	74.04%
Permissions Learning 50%	**91.30**%	98.63%	**90.57**%	**94.41**%
System Calls Learning 50%	75.68%	98.10%	71.60%	82.49%
Hybrid Learning 50%	75.78%	99.21%	71.39%	83.83%
Permissions Learning 75%	85.10%	99.12%	83.30%	90.26%
System Calls Learning 75%	73.17%	97.44%	69.13%	80.49%
Hybrid Learning 75%	76.03%	**99.32**%	71.84%	83.15%
CoTraining MultiView	87.29%	97.68%	86.69%	91.66%
SemiSupervised SystemCalls	89.44%	97.71%	89.01%	93.07%
SemiSupervised Permisions	84.51%	97.49%	83.67%	89.76%
Batch				
Permissions Learning 25%	86.29%	98.46%	85.34%	91.20%
System Calls Learning 25%	75.00%	97.40%	71.20%	81.98%
Hybrid Learning 25%	86.16%	96.68%	86.20%	91.07%
Permissions Learning 50%	90.29%	**99.09**%	89.28%	93.86%
System Calls Learning 50%	82.66%	97.80%	80.52%	88.18%
Hybrid Learning 50%	90.48%	97.60%	90.40%	93.82%
Permissions Learning 75%	**90.70**%	98.55%	90.17%	**94.12**%
System Calls Learning 75%	81.70%	97.06%	79.87%	87.50%
Hybrid Learning 75%	90.62%	97.08%	91.08%	93.96%
CoTraining MultiView	86.82%	90.27%	93.24%	91.65%
SemiSupervised SystemCalls	88.75%	90.44%	**95.96**%	93.04%
SemiSupervised Permisions	89.75%	92.57%	94.56%	93.46%
Online				
Permissions Learning 25%	91.35%	98.62%	90.46%	94.26%
System Calls Learning 25%	81.45%	95.04%	80.79%	87.14%
Hybrid Learning 25%	88.19%	98.17%	86.89%	92.07%
Permissions Learning 50%	**93.49**%	**98.91**%	**92.75**%	**95.66**%
System Calls Learning 50%	84.31%	94.26%	85.71%	89.65%
Hybrid Learning 50%	91.30%	97.91%	91.00%	94.29%
Permissions Learning 75%	92.10%	98.85%	91.22%	94.81%
System Calls Learning 75%	86.11%	95.56%	86.37%	90.65%
Hybrid Learning 75%	89.19%	98.26%	88.02%	92.77%
CoTraining MultiView	87.34%	97.38%	86.29%	91.35%
SemiSupervised SystemCalls	87.37%	97.14%	86.56%	91.38%
SemiSupervised Permisions	91.59%	97.17%	92.05%	94.50%

Table 5. MultiLayer Perceptron Performance Metrics Balanced Data

6

4.4 Multilayer Perceptron with unbalanced Data

The table 6 presents the results of the MLP model trained using unbalanced data across three different scenarios: Normal, Batch, and Online.

In the Normal scenario, the "Permissions Learning 25%" approach achieved the best accuracy at 92.21% and f1-score with a value of 94.89%, the highest recall has being achieve by the "Hybrid Learning 25%" experiment with a value of 99.97% , the highest precision is 98.50% by the "Hybrid Learning 75%" experiment.

In the Batch scenario, the "Permissions Learning 75%" method achieved the best accuracy at 90.70%. The best recall, measuring 95.96%, was obtained from the "Semi-Supervised System Calls" experiment, while the highest precision was recorded by the "Permissions Learning 50%" approach at 99.09%.

In the Online scenario, the "Permissions Learning 50%" experiment yielded the highest overall metrics, achieving an accuracy of 93.49%, precision of 98.91%, recall of 92.75%, and an F1-score of 95.66%.

When comparing the three scenarios, it is challenging to identify a clear winner, as the F1-scores across all scenarios are quite similar. However, if we prioritize recall—which is the most significant metric for this study—the Batch scenario stands out with the highest value.

Finally, concerning the MultiView results, the highest recall of 93.24% was achieved in the Batch scenario. Nevertheless, for all other metrics, the Normal scenario demonstrated better performance.

ML Technique	Accuracy	Precision	Recall	F1 Score
Normal				
Permissions Learning 25%	**92.21**%	97.57%	92.41%	**94.89**%
System Calls Learning 25%	83.72%	83.83%	99.83%	90.83%
Hybrid Learning 25%	83.81%	83.92%	**99.97**%	90.89%
Permissions Learning 50%	85.90%	97.49%	85.52%	90.78%
System Calls Learning 50%	83.61%	95.30%	83.69%	88.98%
Hybrid Learning 50%	91.56%	98.53%	91.22%	94.64%
Permissions Learning 75%	87.88%	96.91%	88.38%	92.22%
System Calls Learning 75%	83.23%	96.05%	81.97%	88.25%
Hybrid Learning 75%	91.49%	**98.50**%	91.15%	94.61%
CoTraining MultiView	85.97%	97.65%	84.86%	90.63%
SemiSupervised SystemCalls	86.97%	97.74%	86.43%	91.54%
SemiSupervised Permisions	84.65%	97.69%	83.45%	89.73%
Batch				
Permissions Learning 25%	93.71%	96.55%	95.36%	95.94%
System Calls Learning 25%	83.80%	83.80%	**100.00**%	90.89%
Hybrid Learning 25%	83.98%	84.94%	98.11%	90.80%
Permissions Learning 50%	**94.23**%	95.17%	97.83%	**96.44**%
System Calls Learning 50%	83.72%	83.79%	99.91%	90.84%
Hybrid Learning 50%	88.75%	96.66%	89.25%	92.78%
Permissions Learning 75%	91.84%	**96.72**%	93.21%	94.87%
System Calls Learning 75%	82.98%	83.75%	98.87%	90.38%
Hybrid Learning 75%	92.19%	93.15%	97.40%	95.17%
CoTraining MultiView	88.66%	90.30%	96.23%	93.06%
SemiSupervised SystemCalls	89.42%	89.82%	97.88%	93.56%
SemiSupervised Permisions	89.48%	90.31%	97.23%	93.53%
Online				
Permissions Learning 25%	92.38%	97.35%	92.82%	94.98%
System Calls Learning 25%	83.80%	83.80%	**100.00**%	90.89%
Hybrid Learning 25%	90.13%	95.48%	91.80%	93.59%
Permissions Learning 50%	**94.40**%	**98.83**%	93.90%	**96.25**%
System Calls Learning 50%	83.80%	83.80%	**100.00**%	90.89%
Hybrid Learning 50%	92.73%	98.30%	92.46%	95.26%
Permissions Learning 75%	94.25%	98.40%	94.10%	96.16%
System Calls Learning 75%	83.80%	83.80%	**100.00**%	90.89%
Hybrid Learning 75%	92.67%	96.96%	93.70%	95.28%
CoTraining MultiView	90.09%	97.83%	89.79%	93.57%
SemiSupervised SystemCalls	90.69%	97.23%	90.54%	93.66%
SemiSupervised Permisions	87.52%	97.58%	87.12%	91.90%

Table 6. Multi Layer Perceptron Performance Metrics No Balanced Data

5. Discussion

This chapter will do an analysis of the obtained results during the research, the point is to describe the results , point out limitations and future research options.

5.1 General observations

Throughout this research, we conducted a series of experiments that focused on recall as the primary metric for evaluating the success of our models. Recall is essential for accurately identifying true positives, which is critical for effective malware detection.

An interesting observation from our study is the inherent trade-off between recall and precision. Typically, models that achieve higher recall rates tend to have lower precision, whereas those with high precision often sacrifice some level of recall. This pattern has been noted in the results mentioned earlier.

Furthermore, we found that all experiments achieving the highest accuracy also had the highest F1-score, suggesting a direct dependency between these two metrics.

5.2 Random Forest vs Multi Layer Perceptron

The initial takeaway from comparing the two algorithms is that, in the case of Supervised Learning (SL), the Random Forest algorithm performs better than the MultiLayer Perceptron (MLP). This can be seen in the batch scenario with balanced data, where the highest recall values for Random Forest are 92.76% for the Hybrid data at 25%, 94.34% at 50%, and 94.51% at 75%. In contrast, the corresponding recall values for MLP are 86.20%, 90.40%, and 91.08%, respectively.

In terms of Semi-Supervised Learning (SSL), the results again suggest that Random Forest outperforms MLP. For instance, when comparing recall values for Random Forest in the balanced data variant within the batch scenario, it achieves 97.76% for Multiview, 96.80% for System Calls, and 96.75% overall. In contrast, MLP under the same conditions yields recall values of 93.24% for Multiview, 95.06% for System Calls, and 94.56% for Permissions. These results clearly indicate that Random Forest consistently outperforms MLP in this context.

A final note is that it is not feasible to make comparisons in an unbalanced data variant because MLP tends to be significantly affected by the bias toward the majority class—in this experiment, the Malware sample—which results in a 100% average recall for System Calls.

5.3 Normal vs Batch vs Online

One key finding from the analysis of the experiments conducted on different scenario types is that for co-training, the "Batch" scenario is the most effective for semi-supervised learning models. All the tree models tested across various scenarios and variants performed best in the "Batch" scenario. It is important to note that this performance increase is significant, with improvements of at least 10% observed when comparing the "Normal" and "Online" scenarios to the "Batch" scenario in several experiments. This expected since Batch scenario hold more data meaning that the model will have more knowledge but at the same will be more expensive to train.

Additionally, it was found that after the "Batch" scenario , the "Online" scenario demonstrated the next best performance. This suggests that leveraging previous experience or knowledge may enhance the outcomes of semi-supervised learning algorithms. Furthermore, it raises the possibility that a larger initial amount of labeled data could potentially yield even better results for these algorithms. This is an intriguing point that warrants further investigation, particularly considering the concept of the initial amount of labeled data as a hyperparameter for semi-supervised learning algorithms.

5.4 Future work and found limitations

The thesis explores the use of co-training techniques for malware detection and highlights important limitations and potential future work arising from this research.

First, it is essential to consider the possibility of constructing a dataset with multiple views. This would allow for a comparison of whether co-training techniques can improve results when more views are utilized for labeling. Currently, no dataset exists with this feature, which represents a limitation. A dataset that includes a set of features related to API calls would provide an interesting additional view.

The duration of the experiments conducted in this research was one year. However, it may be beneficial to either extend or shorten this timeframe to examine its impact on the effectiveness of co-training compared to other models.

Additionally, there is potential to investigate the application of co-training not only for binary classification but also for multi-class classification. This is particularly relevant for distinguishing between different malware families, which can currently be accomplished using the Kronodroid dataset.

6. Conclusion

Malware is constantly evolving, leading to an increase in its variations and making detection more complex. This presents two main challenges: the initial cost and effort required to label data for classifiers, and the fact that malware is designed to evade detection. This thesis contributes to the study of semi-supervised learning for malware detection by evaluating the effectiveness of the co-training technique for labeling. It compares this method with supervised learning approaches and other variations of semi-supervised learning techniques, using different training scenarios. The results are analyzed in the context of both balanced and unbalanced data.

For this research, the Kronodroid dataset was selected and transformed into one-year periods. It was then divided into three categories: Permissions, System Calls, and a combination of both. This division was made to facilitate different experiments that represent various timeframes.

Furthermore, two machine learning algorithms were employed in the study of malware detection: Multi-Layer Perceptron (MLP) and Random Forest (RF). Both MLP and RF were compared using different variations of SL and SSL techniques.

The research concludes that a batch scenario is the most suitable approach for applying co-training as a data labeling technique. It was found that algorithms like Multi-Layer Perceptron (MLP) require additional investment in data balancing to provide a fair comparison with Random Forest (RF). Ultimately, the findings suggest that Random Forest is the best choice for co-training. However, when co-training is compared to algorithms like MLP, it may lose its effectiveness, as better results can be achieved with more cost-effective models.

References

[1] Renas Rajab Asaad. “Implementation of a Virus with Treatment and Protection Methods”. In: *ICONTECH INTERNATIONAL JOURNAL* 4.2 (2020), pp. 28–34.

[2] Marina Proske, Erik Poppe, and Melanie Jaeger-Erben. “„The smartphone evolution-an analysis of the design evolution and environmental impact of smartphones “”. In: *Fraunhofer-Institut für Zuverlässigkeit und Mikrointegration* (2020).

[3] Payton Lo. “A Brief History on Mobile Apps and Their Market Impact”. In: (2022).

[4] Dylan Whitesel, Lisa Whitesel, and English Grade. “The History of the iPhone”. In: *History* 6 (2022), p. 23.

[5] Emiliano Miluzzo et al. “Research in the app store era: Experiences from the CenceMe app deployment on the iPhone”. In: *First Workshop on Research in the Large at UbiComp*. 2010.

[6] Ali Almasli. “Evolution of Portable Devices”. In: (2022).

[7] Harish Sharma. “Android Open Source Project: A study of contribution analysis & degree of openness in governance”. In: (2021).

[8] Statcounter. *Mobile Operating System Market Share Worldwide*. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-202411.

[9] Yirong Yu et al. “A review of fingerprint sensors: Mechanism, characteristics, and applications”. In: *Micromachines* 14.6 (2023), p. 1253.

[10] Karnpimon Krorakai et al. “Smartphone-based NFC potentiostat for wireless electrochemical sensing”. In: *Applied Sciences* 11.1 (2021), p. 392.

[11] AV-ATLAS. *Malware Statistics Trends*. URL: https://portal.av-atlas.org/malware/statistics.

[12] Malwarebytes. *Trojan.Agent - Malwarebytes*. URL: https://www.malwarebytes.com/blog/detections/android-trojan-agent.

[13] Malwarebytes. *Android/FakeApp | Malwarebytes Labs*. URL: https://www.malwarebytes.com/blog/detections/android-fakeapp.

[14] Saket Acharya, Umashankar Rawat, and Roheet Bhatnagar. “[Retracted] A Comprehensive Review of Android Security: Threats, Vulnerabilities, Malware Detection, and Analysis”. In: *Security and Communication Networks* 2022.1 (2022), p. 7775917.

[15] Andrea Di Sorbo and Sebastiano Panichella. “Exposed! A case study on the vulnerability-proneness of google play apps”. In: *Empirical Software Engineering* 26.4 (2021), p. 78.

[16] Hee-Yong Kwon, Taesic Kim, and Mun-Kyu Lee. “Advanced intrusion detection combining signature-based and behavior-based detection methods”. In: *Electronics* 11.6 (2022), p. 867.

[17] Vasileios Kouliaridis and Georgios Kambourakis. “A comprehensive survey on machine learning techniques for android malware detection”. In: *Information* 12.5 (2021), p. 185.

[18] Mariam Memon et al. “Feature-based semi-supervised learning approach to android malware detection”. In: *Engineering Proceedings* 32.1 (2023), p. 6.

[19] Gérard Biau and Erwan Scornet. “A random forest guided tour”. In: *Test* 25 (2016), pp. 197–227.

[20] scikit-learn developers. *scikit-learn — machine learning in Python — RandomForestClassifier*. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[21] Luis B Almeida. “Multilayer perceptrons”. In: *Handbook of Neural Computation*. CRC Press, 2020, pp. C1–2.

[22] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. “Activation functions in deep learning: A comprehensive survey and benchmark”. In: *Neurocomputing* 503 (2022), pp. 92–108.

[23] Xiaojin Zhu. *Semi-Supervised Learning Literature Survey*. Technical Report 1530. Computer Sciences, University of Wisconsin-Madison, 2005.

[24] Xiaojin Zhu and Andrew B. Goldberg. *Introduction to Semi-Supervised Learning*. Springer Cham, 2009. DOI: https://doi.org/10.1007/978-3-031-01548-9.

[25] Tan Gao, Xudong Li, and Wen Chen. “Co-training for image-based malware classification”. In: *2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*. IEEE. 2021, pp. 568–572.

[26] Salma Abdelmonem et al. “Enhancing Image-Based Malware Classification Using Semi-Supervised Learning”. In: *2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. IEEE. 2021, pp. 125–128.

[27] Zequn Niu et al. “A novel anomaly detection approach based on ensemble semi-supervised active learning (ADESSA)”. In: *Computers & Security* 129 (2023), p. 103190.

[28] Annalisa Appice, Giuseppina Andresini, and Donato Malerba. "Clustering-aided multi-view classification: a case study on android malware detection". In: *Journal of intelligent information systems* 55.1 (2020), pp. 1–26.

[29] Moumita Kamal and Douglas A Talbert. "Toward Never-Ending Learner for Malware Analysis (NELMA)". In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE. 2020, pp. 2291–2298.

[30] Alejandro Guerra-Manzanares, Hayretdin Bahsi, and Sven Nõmm. "KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization". In: *Computers Security* 110 (2021), p. 102399. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102399. URL: https://www.sciencedirect.com/science/article/pii/S0167404821002236.

[31] Neda Abdelhamid et al. "Data Imbalance in Autism Pre-Diagnosis Classification Systems: An Experimental Study". In: *Journal of Information & Knowledge Management* 19.01 (2020), p. 2040014. DOI: 10.1142/S0219649220400146. eprint: https://doi.org/10.1142/S0219649220400146. URL: https://doi.org/10.1142/S0219649220400146.

[32] Naomi Altman and Martin Krzywinski. "Ensemble methods: bagging and random forests". In: *Nature Methods* 14.10 (2017), pp. 933–935.

[33] Keras Team. *Keras API*. Accessed: 2023-12-28. 2024. URL: https://keras.io/api/.

[34] Luqman Hakim et al. "Optimizing Android Program Malware Classification Using GridSearchCV Optimized Random Forest". In: *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control* (2024), pp. 173–180.

[35] Sammy Calle. *co_training*. 2024. URL: https://github.com/SammyCalle/co_training.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student’s application for restriction on access to the graduation thesis that has been signed by the school’s dean, except in case of the university’s right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.