Android Malware Detection Using Machine Learning: An Approach for Classifying Malicious Applications

Submitted by
Ahnaf Tahmeed Araf
201-16-497
Department of Computing and Information System
Daffodil International University

Under the Guidance of
Mr. Md. Sarwar Hossain Mollah
Associate Professor and Head
Department of Computing and Information System
Daffodil International University

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computing and Information System

Department of Computing and Information System
Daffodil International University
Birulia, Savar, Dhaka-1216

APPROVAL

This Project titled **"Android Malware Detection Using Machine Learning: An Approach for Classifying Malicious Applications"**, Submitted by **Ahnaf Tahmeed Araf** , ID No: 201-16-497 to the Department of Computing & Information System, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computing & Information System and approved as to its style and contents. The presentation has been held on 29-09-2024.

BOARD OF EXAMINERS

Md Sarwar Hossain Mollah — **Chairman**
Associate Professor and Head
Department of Computing & Information System
Faculty of Science & Information Technology
Daffodil International University

Md. Nasimul Kader — **Internal Examiner**
Assistant Professor
Department of Computing & Information System
Faculty of Science & Information Technology
Daffodil International University

Md. Mehedi Hassan — **Internal Examiner**
Lecturer
Department of Computing & Information System
Faculty of Science & Information Technology
Daffodil International University

Md. Miftah Uddin — **External Examiner**
Head of SBU, Brain Station 23

Declaration

I hereby declare that this thesis (Android Malware Detection Using Machine Learning: An Approach for Classifying Malicious Applications) has been done by me, Ahnaf Tahmeed Araf under the supervision of Mr. Md. Sarwar Hossain Mollah, Associate Professor and Head, Department of Computing and Information System (CIS) of Daffodil International University. I am also declaring that this thesis or any part of it has never been submitted anywhere else for the award of any educational degree like a B.Sc., M.Sc., diploma, or other qualifications.

Supervised By

29/07/24

Mr. Md. Sarwar Hossain Mollah
Associate Professor and Head
Department of CIS
Daffodil International University

Submitted By

Name: Ahnaf Tahmeed Araf
ID: 201-16-497
Department of CIS
Daffodil International University

Acknowledgement

Firstly, I would like to express my sincere gratitude to the almighty God for allowing me to accomplish this B.Sc. study successfully. I wish to acknowledge my gratitude to my thesis supervisor, Md. Sarwar Hossain Mollah, Associate Professor and Head, Department of Computing and Information System, Daffodil International University, for his valuable suggestions, continuous help, long discussion, motivation, countless times, and guidance from the beginning to the end of the thesis work.

Besides, I would like to thank the rest of my thesis mentors: Md. Nasimul Kader, Assistant Professor. Md. Mehedi Hassan, lecturer, for their encouragement, insightful comments, and helpful mind.

Dedication

I dedicate this thesis to my beloved parents, whose unconditional love, encouragement, and unwavering support have been the foundation of my journey. Your belief in me has always been my greatest strength, and I owe all my accomplishments to your guidance and sacrifices.

To my family members, thank you for your constant understanding, patience, and encouragement throughout this challenging endeavor. Your presence has been a source of comfort and motivation, and for that, I am forever grateful.

Abstract

The problem of Android malware detection in imperatively demanding conditions (i.e., an increasing number of applications are being developed at a fast pace as well as ever-growing threats) is investigated. In this article, we suggest a machine learning approach to the problem including data acquisition, pre-processing, feature engineering methods and models. We can see that Bagging Classifier achieves Top Performance metrics bagging the same with Precision, Recall and F1-score of 0.84, thus leading to Overall Accuracy being (0.84), though Random Forest astonishing too having metrics as 0.81 Experimental evaluation against existing techniques shows that our ensembles can detect intricate malware patterns, robustly addressing the limitations of metadata-based detection strategies. We also integrate Explainable AI methods to improve the interpretability and understandability of the decisions made by our model, making it easier for trust in cybersecurity implementations. The work underscores the necessity for evolving learning schemes to encompass new threats and future investigations aiming to exploit deep learning techniques, cross-platform detection strategies, and implications regarding user privacy and ethical concerns. This work gives a wealthier gainful binary to envision portable security look into field and in malware detection and avoidance specifically.

Table of Contents

Abstract **i**

Chapter 1 Introduction **1**

1.1 Overview 1

1.2 Motivation 3

1.3 Problem Statement 5

1.4 Organization of the Project 7

Chapter 2 Related Works **9**

2.1 Related Work 9

2.2 Summary 13

Chapter 3 Methodology **15**

3.1 Methodology 15

3.1.1 Data Collection 15

3.1.2 Data Preprocessing 16

3.1.2.1 Data Cleaning 16

3.1.2.2 Handling Categorical Variables 17

3.1.2.3 Data Balancing 17

3.1.2.4 Splitting the Data 17

3.1.3 Feature Engineering 18

3.1.4 Model Building and Model Testing 18

3.1.4.1 Model Building 18

3.1.4.2 Model Testing 18
3.1.5 Performance Evaluation 19
3.1.5.1 Accuracy 19
3.1.5.2 Precision 19
3.1.5.3 Recall 19
3.1.5.4 F1-score 19
3.1.6 Explainable AI (XAI) 20
3.1.7 Decision Making 21
3.2 Summary 21

Chapter 4 Experiment and Result Analysis **22**
4.1 Experiment Environment Setup 22
4.1.1 Software and Tools 22
4.1.2 Libraries 23
4.2 Dataset Collection and Preparation 23
4.2.1 Summary of the Dataset 24
4.2.2 Label Distribution 24
4.2.3 Data Loading and Initial Exploration 25
4.2.4 Data Cleaning 25
4.2.5 Handling Categorical Variables 26
4.2.6 Checking for Missing Values 26
4.2.7 Data Balancing 26
4.2.8 Splitting the Data 27
4.2.9 Feature Importance 27
4.3 Exploratory Analysis of the Dataset 28
4.4 Result Analysis 35
4.4.1 Feature Importance Analysis 35
4.4.2 Model Performance Analysis 37
4.4.2.1 Performance Analysis of Classifiers 39
4.4.3 Comparative Analysis 41

4.5 Discussion 43
4.6 Summary 44

Chapter 5 Conclusions and Future Work **47**
5.1 Summary of the Dissertation 47
5.2 Future Research Directions 48

Bibliography **49**

A Appendix **54**
A.1 Snapshot of Dataset 54

B Appendix **55**
B.1 Snapshot of Dataset 55

List of Figures

3.1 Overall processing phases for Android malware detection using machine learning 16

4.1 Class label distribution in the dataset. 29
4.2 Imbalance class ration with respect to Benign class. 30
4.3 Visualization of columns with null values in the dataset used for this study, aiding in assessing missing data across features. 31
4.4 Distribution of Flow Durations for Different Classes. 33
4.5 Distribution of Packet Length Variance for Different Classes. 34
4.6 Visualizing pairwise correlations between features to identify highly correlated and potentially redundant features for better feature selection in model building. 36
4.7 Visualization of the top features contributing to Android malware detection. 38
4.8 Comparative analysis of the different classifiers. 42

List of Tables

2.1 Summary table of the related works on Android malware detection using machine learning, highlighting their approaches and limitations. 12

4.1 Performance Metrics of Classifiers 40

4.2 Comparative Analysis of Malware Detection Approaches 46

Chapter 1

Introduction

This chapter kicks the research off by providing a background of Android malware and how it is used — one that has come to grow, adapt, and change with time. The following explains the particular goals, zeroing in on how well machine-learning techniques work for malware detection versus traditional methods. The drive underscores a necessity for strong detection mechanisms and basically the fact that the current models may not be sufficient, especially with respect to traditional permission-based security. Finally, the problem statement discusses difficulties in exemplifying malware instances, obfuscation methods and requirement of real-time detection.

1.1 Overview

This study aims at improving the Android malware detection using a systematic approach with machine learning algorithms. The main goal is to provide a model that can generalize well and detect known and even unknown malware with high accuracy, whilst remaining interpretable for user transparency. The study starts with the architecture of the model, highlighting how these different parts contribute to having a better method for malware detection. It breaks down into several key phases;

1. **Datasets**: Multiple sources — App Store and Malware Repository are utilized to get a comprehensive dataset. This dataset contains information regarding different applications, their corresponding features and metadata as to whether the application is benign or malicious.

2. **Pre-processing**: It is a very important stage where the following activities are needed to be performed on the data (cleaning, handling the categorical variables, balancing class imbalances and splitting the data into train and test dataset) These steps are essentially required to get the data in shape for making effective models.

3. **Feature Engineering**: Features take under consideration for model performance! It starts from scaling, normalization and creating new features which can capture the complex relations within the data.

4. **Building the Model and Testing:**: Machine learning algorithms, such as a Decision Tree or Random Forest, or SVC used to construct a forecasting model. The effectiveness of these models is then evaluated using a different testing dataset.

5. **Performance Review:**: The study uses confidence and performance metrics (accuracy, precision, recall and F1-score) to evaluate the appropriateness and effectiveness of the models. The evaluation is important to know about the limitations and features of them.

6. **Explainable AI (XAI)**: Uses techniques like SHAP, LIME to improve model transparency. They give the insight into how model makes decision and well as reasoning which helps in building a trust on your model and debugging.

7. **Decision Making**: Then, based on how well the model does and its interpretability, one makes decisions about deployment. This refers to testing for robustness, how easy it is to integrate and the degree of compliance with quality standards. The proposed research provides great potential to contribute positively to the Android malware detection field, especially in the context on ongoing changes and adaptations in a variety of malware type, and regarding transparency models (i.e., which are critical for cyber security).

1.2 Motivation

Recently, there has been a flurry of interest in the research community on using machine learning based detection approaches for Android malware as the traditional forms of detecting these have started to become obsolete due to significantly higher sophistication level exhibited by these malwares. Machine learning (ML) methods, especially deep learning (DL), have demonstrated improvement of detection accuracy and effectiveness from many field-scale datasets in application for different features.

The salient feature attributed to the ML-based malware detection is its potential in utilizing different features extracted from Android applications. For instance, Kim et al. highlight the Android Manifest permissions it defines. xml files, can be treated as strong signals for potential malicious software behavior that are hard to obfuscate [1]. Similarly, Onwuzurike et al. emphasize the importance of behavior modeling, where a sequence of API calls is captured to detect malware patterns that evolve over time [2]. This is supported by Li et al, who propose to use a set of opcode sequences that are analyzed via convolutional neural networks (CNNs) to evaluate the dynamical behavior of an application for maliciousness [3]. Static analysis and dynamic analysis are better integrated to give a more powerful view of an application behaviour [4].

Deeper applications of deep learning have further revolutionized the field of malware detection. Yuan et al. The solution is simply that signature-based approach will not be enough to capture the variety of Android malware and that feature extraction and ML methods need to get more sophisticated too [4]. The work of Niu et al. [5] further proves that deep learning models are able to properly encode the temporal aspect of pipeline inspections when some knowledge is available for inspection, e.g.; from the static features and more importantly by combining both methods above renders high classification accuracy. In contrast, the hybrid models suggested by Lu et al. [6] demonstrate the ability of combining multiple deep learning architectures to improve detection performance.

Even though it has been a great evolution, still malware detection has some

challenges. Hefter et al. showed that the metadata-based inspection has limitations as the shown sophisticated malware could change the metadata attributes to avoid detection, therefore generating false negatives [7]. Bakır et al. employed auto-encoder feature extraction to introduce DroidEncoder, but it performs poorly on the complexity of modern malware behaviors [8]. Iqbal et al. integrated with proposed RThreatDroid, which uses multiple data modalities for ransomware detection, however also adds computational overhead [9]. Chow et al., on the other hand, addressed concept drift in malware classifiers as well [10], while Balikcioglu et al. tested the use of LSTM models for feature extraction, with potential missed important behavioral differences [11]. Alam et al. introduced MORPH (A concept drift adaptation technique for neural networks) [12]; Muzaffar et al. proposed an active learning method for Android malware detection and they also required high quality labeled data [13].

Due to this, the detection methodologies needs to advance continuously as well due to the challenges faced by obfuscation techniques and rapid evolution of malware. For instance, Maray et al. present an analysis of the effectiveness of dynamic analysis on various obfuscated, encrypted or packed malicious features (i.e., crypter algorithms), and then demonstrate the benefit that ML and DL can bring towards learning novel threats [14]. A detailed review on Android malware detection via a set of machines learning based ensemble methods was presented as well by [15] where it has been mentioned that among other techniques, the ensemble approach improves classification accuracy and reliability. These challenges need to be addressed by further research, to the end of incrassation the efficiency of machine learning systems occurrence in Android malware detection.

Several machine learning methods can be used to tackle these problems. The first is practically to improve the robustness of malware classifiers using ensemble learning methods like boosting and bagging that combine the strength of multiple models to lessen false negatives; hence helping to mitigate detection rates [16]. In addition, semi-supervised and unsupervised learning approaches can remove the req-

uisition of a large amount of labeled data by enabling models to take advantage of unlabeled data for improved generalization in order to adapt more effectively against new malware behaviors [17]. In a similar manner, an anomaly detection algorithm like Isolation Forest or One-Class SVM could be employed to detect new types of malware by detecting deviations in the behavior of the application which have not been seen before [16]. Online learning algorithms have been used to solve the problem with concept drift, which is that as malware techniques change over time, old classifiers might become invalid and won´t make good predictions without adapting themselves to their changing environment [17]. Lastly, reducing the complexity of current malware detection may be possible by leveraging feature selection methods for dimensionality reduction and focusing on the most meaningful features that in turn can help to have more interpretable decision models—overall enhancing model interpretation and performance. Further investigation on these techniques will be necessary to improve the performance of machine learning-driven malware detection systems.

1.3 Problem Statement

The incredible growth of Android devices has caused them to become a key target for malware, which creates major security issues for both users and developers. This is because advances in traditional security protection techniques are not keeping pace with the increasing sophistication of malware, such as polymorphic and obfuscated variants, thus rendering conventional detection methods inadequate [18]. Traditional method like signature-based approaches are unable to detect the new arising malware that could bypass varied evasion techniques [7].

The latest studies proved that machine learning (ML) became a great solution to improve the efficacy of Android malware detection [19]. Many current models have some limitations, such as using labelled datasets that could be incomplete and not representing all the possible variants of malicious software behaviors (due to which they may introduce an bias at their classifications) [13]. Further, the expensive

nature and complexity of deep learning models due to their enormous number of parameters make them hard to employ in real-time systems [20].

In addition, there is a severe urgency for ML models used in cybersecurity to make the decision-makings processes transparent. A lot of models have black box predictions and provide little in terms of interpretable performance indicators that could guide some interventions to address false positives and negatives [12]. Unfortunately, the lack of interpretability of DL models does not only hinder user trust but also tremendously restrict its practical deployment in dynamic threat landscapes [21].

Therefore, the purposes of this research are:

- Create a machine learning approach to Android malware detection model incorporating the changing landscape of threats.
- Increase the explainability of the detection model to make it more transparent and help users and stakeholders trust it.
- Performance evaluation and comparison with existing methods to demonstrate that our approach is better performing in a real-world application.
- The goal of this research is investigating the trade-offs due to these challenges and provide a better understanding of approaches used in Android malware detection which will contribute to existing anti-malware solutions, enhancing a secured Android environment.

The spread of malicious apps in the Android operating system is a growing concern as it proliferates and evolves with an ever-increasing number of new applications built to attack it. To this end, several methods are researched which can broadly be grouped under static, dynamic and hybrid analysis techniques. Static analysis works by analyzing application code without executing the code, while dynamic analysis observes the behavior of an application during runtime. Hybrids use these two together to improve detection shelter and sinteny accuracy.

Recent research has shown that the Android malware composition change is quite drastic, reaching expected 6300 new examples every day [22]. This disturbing figure further afield illustrates that there is an incredibly high-demand for the need of efficient detection methods. As the traditional permission-based security model in Android has failed special attention needs to be paid to detection strategies, due the apparent ease that attacker have bypassing these controls [23, 24].

According to the research, machine learning (ML) techniques are found to be suitable for detecting malware as they can learn from huge dataset and change according to new threats unlike signature-based methods [25, 26]. For example, hybrid models (where ML is combined with static and dynamic analysis) have reached the detection accuracies of 98.69% [27]. Further, the control flow graphs and even opcode sequences have also been studied to improve the detection of malware capabilities [28, 29].

Detecting obfuscated malware Obfuscation by the malicious entities renders a significant challenge to detection, since many of the malware are designed under behavioral deception techniques [30, 31]. This has led to the evolution of sophisticated detection techniques, from ensemble learning to deep neural networks (DNNs) for more accurate labelling of the obfuscated malware [26]. When used with static features that can be derived from applications code, Data-Mining approaches based on Machine Learning (ML) showed detection rates of more than 96% while maintaining a low false positive rate [19].

1.4 Organization of the Project

The dissertation is organized as follows:

- • **Chapter 1 Introduction.** This chapter has provided a brief overview, motivation and problem statement for the following chapters. The dissertation then addresses the contribution.

- • **Chapter 2 Literature Review and Background of the Study.** This chapter deals about purpose and significance of the project and related literatures reviews. It would then clearly address the limitations of existing work since these are the issues that this is meant to focus on.

- • **Chapter 3 Methodology.** This chapter describes the methodology adopted for the solution to problems which have been discussed in this dissertation. First, we briefly introduce the structure of the proposed model. Next, it talks about important modules of the model in detail with relevant pseudocodes.

- • **Chapter 4 Experimental Results and Analysis.** In This chapter we describe the experiment and results which we want to achieve in this thesis. We first show the experimental process in dataset collection, preparation, and analysis among other things. This is followed by the analysis of the results from the experiment.

- • **Chapter 5 Conclusion and Future Work.** Last, this chapter summarizes the dissertation indicating the limitations and future works.

Chapter 2

Related Works

This chapter discusses various works in malware detection using machine learning approaches.

2.1 Related Work

Machine learning for android malware detection is a trending research area. Shifu et al. proposed Hindroid, which applies neither feature selection nor neural networks, but merely standard multi-kernel learning with SVM for Android malware detection [32]. Though Hindroid is functional, it seems to be based in some degree on SVMs and more traditional machine learning methods that are less suitable to the constantly shifting malware landscapes and complex habits of behavior.

A call flow graph along with one-class SVM were also used to detect Android malware by Sahs and Khan [33], providing resistance to the some extent against known malware behavioral patterns. But this technique can be difficult to catch new and zero days patterned malwares, since new viruses do not follow the call flow graph patterns.

Studies have shown that machine learning is a promising method of detecting Android malware, specifically for the recognition of currently unknown malwares [18] While this may promise, the dependence on labeled datasets for supervised learning is a limitation as it might not necessarily reflect the diverse set of Android malware behaviors. Additionally, the combination of machine learning and deep learning and clustering has been suggested to improve Android malware detection models [34]. Deep learning models are great for untangling the complex relationships, but they

can also be very resource-hungry and might require tons of computational power on training and inference.

Moreover, ensemble machine learning methods have been employed to improve the performance of Android malware detection [19]. Nevertheless, merging multiple classifiers and feature sets might raise complexity and model overhead — two obstacles that could impair real-time detection capacities. Literally, they (e.g., SVM, Random Forest and J48) were employed for classifying Android applications as malware or benign in a lot of papers [19]. These algorithms are powerful but may not necessarily be able implicitly to give the reasons for their decisions, which is crucial in relation to false positives and negatives while detecting malware.

Machine Learning Detection of Android Malware: Android malware detection has made machine learning-related strides, but each comes with its own challenges and tradeoffs. Hefter et al. The skills of metadata-based detection in Android security were already amply demonstrated by past investigations into detection using attributes like permissions and API calls [7]. On the other hand, metadata-based approaches might be at risk with to evasions that modify or hide metadata fields.

Bakır et al. [8] leverages auto-encoder based feature extraction to improve the resilience of malware detection systems. First, while the auto-encoder model was highly effective in capturing latent features, it may suffer from diversity and complexity of modern malware behaviors which can potentially restrict on detection capabilities for novel evolving adversaries.

Iqbal et al. [35]proposed a hybrid ransomware detection approach combining image analysis, text mining and code inspection. This holistic method improves threat assessment but incurs computational overhead, affecting the scalability and real-time capabilities for detection.

Chow et al. Concept drift was considered, e.g., in malware classifiers by N. Revett Jr. [10] to develop frameworks for changing malware behaviors. Continuous retraining and concept drift adaptation in real-time environments can consume significant computational resources, which very often precludes practical deployment

in any resource-constrained setting.

Balikcioglu et al. [11] the use of LSTM models for feature extraction was employed to examine the effect of disassembly methods and data characteristics on malware detection accuracy. Although LSTM models do a good job capturing sequences, they may miss important very small changes needed for accurate detection.

Alam et al. MORPH, a concept drift adaptation technique for neural networks in the field of malware detection was introduced by [12]. Although adaptive its organic nature may struggle with the fast-changing malware landscapes and over time this could result in a diminished detection performance.

Muzaffar et al. Active Learning for Streaming Android Malware Detection [13]. Although effective in refining models, active learning effectiveness comes at the expense of broader requirements for high-quality and representative labeled data that may be expensive or difficult to produce on-the-fly settings.

Vidal et al. Another research [21] was based on behavioral analysis, suspicious boot sequence is analyzed for malware detection. Of course, it is difficult to rely on behavioral analysis which increases the chances of false positives since this largely depends on behavioral anomalies.

Kumar et al. [36] proposed as well a MIL-based hardware solution called RT-HMD, for malware detection. However, hardware-based approaches can encounter compatibility problems with different device architectures whilst improving real-time capabilities.

Chaieb et al. Bertroid: A bert-based model for malware detection [20]. While it is resilient to changes in the threat landscape, the large computational requirements and sophisticated training process of BERTroid may reduce its scalability within resource-constraint environments.

Table 2.1: Summary table of the related works on Android malware detection using machine learning, highlighting their approaches and limitations.

Study	**Approach and Methods**	**Limitations**
Shifu et al. [32]	Multi-kernel SVM with Hindroid	Limited adaptability to rapidly evolving malware landscapes
Sahs and Khan [33]	Call flow graphs with one-class SVM	Difficulty in detecting novel and zero-day malware
Research Review [18]	General promise of ML in Android malware detection	Reliance on labeled datasets for supervised learning
Rathore et al. [34]	Integration of ML, DL, and clustering	High computational resources required
Yerima et al. [19]	Ensemble learning techniques	Increased complexity and model overhead
Hefter et al. [7]	Metadata-based detection	Vulnerable to evasion techniques using obfuscated metadata
Bakır et al. [8]	Auto-encoder based feature extraction	Difficulty with diverse and complex malware behaviors
Iqbal et al. [35]	Hybrid approach with image analysis, text mining, and inspection	Computational overhead impacting scalability
Chow et al. [10]	Frameworks for concept drift adaptation	Resource-intensive retraining in real-time environments
Balikcioglu et al. [11]	LSTM models for feature extraction	Limited ability to capture subtle variations in malware behavior

Continued on next page

Study	Approach and Methods	Limitations
Alam et al. [12]	MORPH: Concept drift adaptation for neural networks	Challenges in rapidly evolving malware landscapes
Muzaffar et al. [13]	Active learning framework	Dependency on high-quality labeled data for real-time settings
Vidal et al. [21]	Behavioral analysis through boot sequence analysis	Increased false positives from behavioral anomalies
Kumar et al. [36]	Hardware-based MIL formulations	Compatibility issues across different device architectures
Chaieb et al. [20]	BERTroid: Malware detection with BERT architecture	Computational complexity and training requirements

2.2 Summary

It also delves into a number of ways Android malware detection can be done, and discusses situations wherein these techniques may suffer. It opens by analyzing classical machine learning methods for malware analysis like the multi kernel SVM model from Shifu et al., showing that this kind of solutions tend to perform well but have trouble keeping up with new trends on malware development. Although the one-class SVM with Sahs, et al., call flow graph method reported by Khan 45 is robust to known malware, it lacks efficacy in distinguishing new threats.

In short, machine learning definitely has very large promises when it comes to detecting unknown malware — but it is also heavily dependent on the datasets that are supplied, and an unknown classification will be much more likely when compared to Android malware behaviors as a whole. Rathore et al. offer a way to counter

this by combining all of the data in such a way that — if perfect blanks, correct blanks and incorrect matching are simultaneously aimed for optimal performance, respectively — the best ratio is going to come out. suggest the idea of unifying deep learning (DL) and clustering, but these models need plenty of computational resources. However, ensemble learning is complex because it combines several models, and as published by Yerima et al., it has high accuracy which leads to low-time processing for object detection in real-time fashion.

Recent work, e.g., Hefter et al. using metadata-based detection In contrast, the approach by Bakır et al., which leverages features of minification and obfuscation resistance (in a domain invariant ways), feature type obfuscation and autoencoders offer novel techniques but are still susceptible to evasion or have difficulty generalizing across malware variations. A few other approaches are notable such as Hybrid approach, active learning and drift adaption of behaviors changes in malware. All of these methods help in improving the malware detection process to some extent, but each one has its trade-offs such as increased computational overhead, false positives or the need for frequent retraining.

In sum, literature accumulated presents some historical challenges and improvements on Android malware detection strategies to preserve the trade-off between **accuracy** of malicious sampling discovery, **generalizability** for an open threat environment with diversified nature of malware samples generating daily, and **real-time mobile security** that imposes an immediate overhead on a resource-constrained setting, as shown int Table 2.1.

Chapter 3

Methodology

In this chapter, we describe the proposed methodology and how this work is accomplished. This chapter presents the main contribution of the project. First of all, we figure out and discuss the architecture of the proposed model. Then, we discuss some preliminary steps and overview behind that methodology. Then, important modules of the model are discussed elaborately with necessary pseudocodes.

3.1 Methodology

The methodology for detecting Android malware using machine learning involves a structured and systematic approach to ensure the accuracy, reliability, and interpretability of the developed model. The process begins with the collection of a comprehensive dataset of Android applications, encompassing both benign and malicious samples. Subsequent steps include rigorous data preprocessing, feature engineering, model building and testing, performance evaluation, and the application of explainable AI techniques to elucidate the model's decision-making process. Each step is designed to enhance the model's performance while maintaining transparency and robustness, as shown in Figure 3.1.

3.1.1 Data Collection

The most basic and initial phase to know Android malware detection using machine learning is the data collection phase. This consists of creating diverse datasets that serve as the foundation for the analyses on Android applications from different sources like app stores, malicious databases, and online resources. Benign and

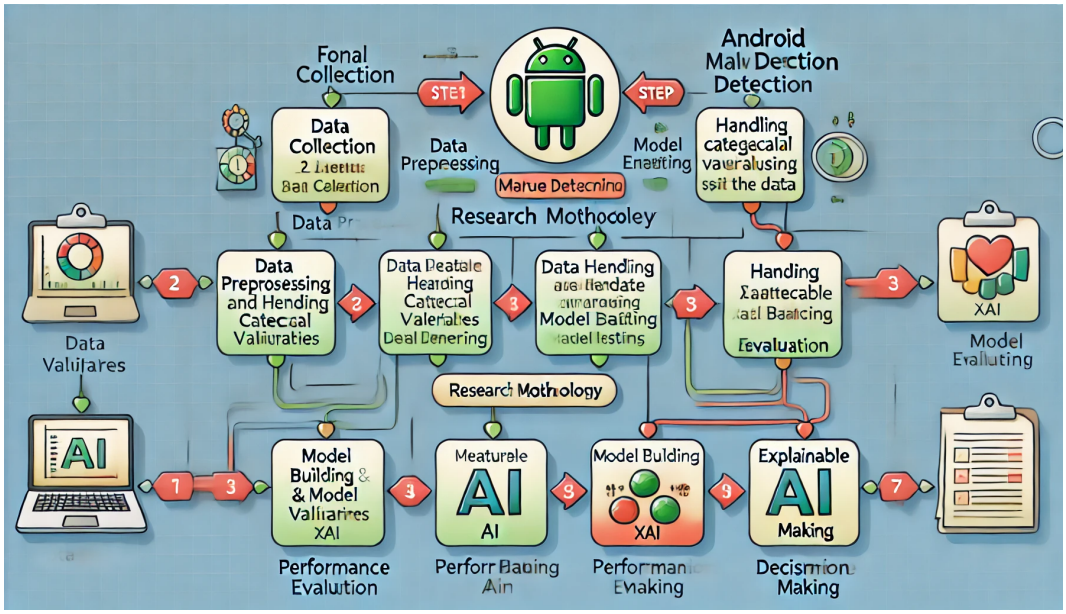

Figure 3.1: Overall processing phases for Android malware detection using machine learning

Malicious applications, with features, behaviours and their metadata.

$$\text{Dataset} = \{(x_i, y_i)\}_{i=1}^{n}$$

Where x_i represents the features of the i-th application, and y_i represents its label (1 for malware, 0 for benign).

3.1.2 Data Preprocessing

Data preprocessing is one of the crucial and indispensable steps in making sense of the raw data. This is a three step process consisting of:-

3.1.2.1 Data Cleaning

Transformation: Cleaning data involves deleting duplicates, treating the missing values, dealing with garbage and sometimes irrelevant information. This step will

clean the data to ensure that there are no inconsistent data and noise due to an imbalance in the dataset.

$$x_i' = f_{\text{clean}}(x_i)$$

Where f_{clean} is the cleaning function applied to the features x_i.

3.1.2.2 Handling Categorical Variables

For example, categorical feature (permission types or API categories) are converted to numerical representation such as one hot encoding. This step is needed to standardize these features for easier consumption by machine learning algorithms.

$$x_i'' = f_{\text{encode}}(x_i')$$

Where f_{encode} is the encoding function.

3.1.2.3 Data Balancing

In many cases, high bias is caused by the imbalanced nature of malware datasets where benign samples outnumber malicious ones. Also, to balance the dataset, we use techniques like SMOTE (Synthetic Minority Oversampling Technique) that would ensure model doesn't become biased toward majority class.

$$\text{Dataset}_{\text{balanced}} = \text{SMOTE}(\text{Dataset})$$

3.1.2.4 Splitting the Data

The balanced dataset is divided between training and testing datasets. That is, the model is fitted on the training set and then it is evaluated using a testing set.

$$\text{Dataset}_{\text{train}}, \text{Dataset}_{\text{test}} = \text{split}(\text{Dataset}_{\text{balanced}}, \text{ratio})$$

3.1.3 Feature Engineering

Which means choosing and changing certain features, in order to improve the performance of a model. Scaling, normalization, as well as potentially other new features with poly transformations or interactions between features.

$$x_i''' = f_{\text{feature}}(x_i'')$$

Where f_{feature} represents various feature engineering techniques.

3.1.4 Model Building and Model Testing

3.1.4.1 Model Building

Predictive models are constructed using different machine learning algorithms. Below written are few of the popular algorithms for malware detection:

- Decision Trees and Random Forests
- Support Vector Machines (SVM)
- Neural Networks

The models are trained using the training dataset.

$$\text{Model} = \text{train}(\text{Model}_{\text{type}}, \text{Dataset}_{\text{train}})$$

3.1.4.2 Model Testing

Test the model: after training the models are tested on some external testing dataset to see how well they do. Predictions are made, Performance metrics are calculated.

$$\text{Predictions} = \text{Model}(\text{Dataset}_{\text{test}})$$

3.1.5 Performance Evaluation

To be sure about the accuracy and reliability, several metrics are used to evaluate the performance of the models. These metrics include:

3.1.5.1 Accuracy

The proportion of correctly classified instances.

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

3.1.5.2 Precision

The proportion of true positive instances among the instances classified as positive.

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

3.1.5.3 Recall

The proportion of true positive instances among the actual positive instances.

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

3.1.5.4 F1-score

The harmonic mean of precision and recall.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where TP, TN, FP, and FN represent True Positives, True Negatives, False Positives, and False Negatives, respectively.

3.1.6 Explainable AI (XAI)

Therefore, we use Explainable AI — a group of techniques to interpret the decision of the model — this helps us to understand and analyze how our model differentiate between benign and malicious apps. This part for providing insights of the model's decision making process, SHAP (SHapley Additive exPlanations) and LIME(Local Interpretable Model-agnostic Explanations) are some techniques used.

$$\text{Explanation} = \text{XAI}(\text{Model}, x_i)$$

Explainable AI (XAI) is an indispensable component to build machine learning models for Android malware detection. Even complex traditional machine learning models like deep neural networks are capable of providing high accuracy, they often work as a "black box" which does not allow any explainability on the working decision-making process. The opaque nature of these technologies is highly problematic, especially in the context of cybersecurity, where it is vital that we can provide an explanation as to why a model made a certain decision so that we may trust those that implement them, hold them accountable for their results, and take further action where necessary. Benefits of this approach includes:

- Transparency: XAI techniques are helpful to understand how machine's decision-making. This transparency is crucial for security professionals to know why some applications are being flagged as malicious, and this information can be useful to enhance the model and fix any biases.
- Trust: Users and stakeholders are less likely to trust a model if they do not think they understand how it operates. Explanations for predictions contribute towards building a trust in the system that it would elicit accurate and reliable results over its predictions.
- Debugging and Improving Models: By analyzing which features the model learns to consider important, developers can recognize problems in the training data or model building process. We constantly keep our engines well

maintained and calibrated, to be sure they deliver the required accuracy and robustness.

3.1.7 Decision Making

This leads to the deployment of the model in real-world applications based on how well it performs and its interpretability; This means, the model works as expected and can able to use for real world problems with tradeoffs between fulfill requirements such as accuracy or explainability. Each of these encompass their decision-making processes:

- Assessing the robustness and dependability of the model.
- Integration into existing systems.
- Ensuring compliance with regulations as well as ethics.

3.2 Summary

In this chapter, first we discuss about some definition which related with our work. Next are some various kinds of meme. Afterwards, we need to design the architecture and look for it extensively. In the last section, we present our algorithm to effectively recognize religious abuse on Twitter and to track it.

Chapter 4

Experiment and Result Analysis

This chapter presents experiment and results that have been targeted to accomplish in this dissertation. First, the experimental procedures including dataset collection, preparation and analysis are demonstrated. Since the proposed model is based on a supervised approach for detecting and tracking religious abuse in social media using data set based on a real scenario, so the experiment is mainly focused on designing abuse detection model. Then the results obtained from experiment are analyzed.

4.1 Experiment Environment Setup

Establishing the experimental environment is essential to make both the analysis and model training efficient and reproducible. Here is the brief representation of software and libraries used for this malware detection project.

4.1.1 Software and Tools

- **Google Colab**: This is an online Jupyter notebook environment provided by Google. Question 8: Online Python Question 9: What is an online compiler to execute the written python code in our browser, which helps us writing collaborative projects and also help it accessing fast cloud-based GPUs.

- **Python**: The programming language being used in the project. Its vast array of libraries make python very good for data science and machine learning problems.

4.1.2 Libraries

- **Pandas**: Its an open-source data manipulation tool and it is use to perform data manipulation tasks. It offers data structures and functions you need to manipulate structured data, and makes manipulating textual tables easy.
- **NumPy**: Used for numerical computing. Offers support for arrays and matrices, as well as mathematical functions to operate on these data structures.
- **Matplotlib and Seaborn**: Used for Data visualization Matplotlib — is used to create static, animated, and interactive visualizations in Python Seaborn — built on top of Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics in Python.
- **Scikit-Learn**: a rich ML library, Tools for Data Mining and Data Analysis — It has simple, efficient tools for data mining and data analysis, accessible to everybody, and reusable in various contexts. It includes implementations for a number of classical machine learning algorithms like classification, regression, clustering and dimensionality reduction.
- **Warnings**: It is used for warning dealing with multiple warnings which can happen while training and evaluating of machine learning model without collapsing any analysis flow.

4.2 Dataset Collection and Preparation

The dataset choose to use in this project is Android Malware Detection, from kaggle. Dataset Description The dataset consists of many feature vectors, each with 9 features extracted from 10,000 malicious and 10,000 benign APKs. This method of tracking network activity for bad behavior is a key indicator that the malware could be present. Machine learning models are really good for them because they can look at incoming data in the system and make predictions about what something new might be—that it smells like Malware, even though the individual patterns that

comprise it have never been seen before. This dataset also includes features like requested permissions, API calls used and network activity.

4.2.1 Summary of the Dataset

The dataset includes features that assist in detecting and classifying various types of malicious applications on Android devices. This dataset consists of:

- **Number of Entries**: The dataset contains 355,630 instances (rows).
- **Number of Features**: There are 85 columns representing various features of the applications, including network activity, permissions requested, API calls made, and more.
- **Labels**: The dataset includes four labels, each representing a different category of applications:
  - *Android_Adware*: Applications that display unwanted advertisements.
  - *Android_Scareware*: Applications that scare users into purchasing unnecessary services or software.
  - *Android_SMS_Malware*: Applications that send unauthorized SMS messages.
  - *Benign*: Legitimate, non-malicious applications.

4.2.2 Label Distribution

It is also a very unbalanced dataset in terms of labels, a key point for model training and validation. First of all, let's analyze the distribution of the labels as follows:

- **Android_Adware**: 147,443 instances
- **Android_Scareware**: 117,082 instances
- **Android_SMS_Malware**: 67,397 instances

- **Benign**: 23,708 instances

This dataset is a slight variation of the 'Cobranded' partion retrieved from CIC repository which offers wide range of datasets for cybersecurity studies. Because of the more number of infected instances over non-infected (boring) ones dataset is very imbalanced and that's why you have to use data balancing methods to prevent your machine learning model from biased towards those majority classes.

4.2.3 Data Loading and Initial Exploration

The first step in the pipeline was to convert the data into a pandas DataFrame so it could be easily manipulated and analyzed. This step was to load the data from a CSV file. In my case the first step in exploration of the dataset was to check what is inside it, I mean number and types of all features and distribution of y(reply) variable as well.

- **Loading the Data**: We load our data using pandas' read_csv function that reads this CSV file into a Data Frame. Which you can then manipulate more easily and process with pandas awesome data processing framework.

- **Initial Inspection**: The data was printed on the top 5 rows of dataset for initial checking. Check the structure of the dataset.(column names, data types and some sample values.)

4.2.4 Data Cleaning

It is very important to get the data in optimum shape before any analysis and to do so, Data cleaning is one of most impotent steps. This is comprised of deleting data that is inaccurate or inconsistent.

- **Removing Irrelevant Columns**: Drop the `Timestamp` column as it has no use for Analysis. Standardized features Selection: This step ensures that only relevant features are included in the dataset.

- **Handling Missing Values**: Most machine learning models cannot work with missing values. The missing values were checked in the dataset and solutions like filling the with proper substitutes or deleting rows having the missing values were done.

4.2.5 Handling Categorical Variables

Since the machine learning models work with numeric input, therefore we need to convert these categorical variables into a numerical value.

- **Label Encoding**: The above feature engineering techniques required converting Categorical variables into Numerical, where Label Encoding was used for the same. For this, I decided to have integer mappings for the unique category values. Required for models that cannot take in categorical directly.

- **Ensuring Completeness**: After encoding all the labels, we did a check to see that there were no NA s introduced into our data, due to label encoding. All matters were deal with in order to preserve the dataset enforceability.

4.2.6 Checking for Missing Values

Missing values were thoroughly checked even after initial data cleaning stage for making sure it is an immaculate dataset.

- **Sum of Missing Values**: It returned the total count of null values in each column. Identified and improve columns with missing values appropriately.

- **Handling Strategies**: Imputation with the column mean or median, drop rows or columns including a high percentage of missing values.

4.2.7 Data Balancing

The class imbalance will result in the biasing of the model towards one majority class, which give rise to poor performance on minority classes.

- **Resampling Technique**: Oversampling the minority class using the SMOTE method: For example, over-sampling the examples of rare output to make it equal with the number of common outputs. For this, we used the resample function from sklearn's utils module.

- **Ensuring Balance**: All the classes were checked in the balanced dataset for an equal count of samples.btnSave This step is essential in order for the model to be trained on a relatively uniformly distributed selection of all classes.

4.2.8 Splitting the Data

It is a standard practice to split the dataset between testing and training sets in order to validate the model on unseen data.

- **Train-Test Split**: Used the train test split function from sklearn to split the dataset into two sets, one for training and another for testing. In general, 80% of data used for training and 20% of it using for testing.

- **Shuffling Data**: A few intervening steps has to be undertaken such as: The data had been shuffled prior to splicing to insure that each set would be representative of the entire structure.

- **Verifying Splits**: Sizes of train and test sets are verified to make sure everything was split correctly in the first place. The dimension of the feature matrices and target vectors were printed to verify this.

4.2.9 Feature Importance

It is essential to know which of the features are important for our target variable so that we can interpret and optimize our model.

- **RandomForestClassifier**: We calculated feature importance using a RandomForestClassifier. It returns a list of important features from most to least importance in terms of contribution towards the prediction.

- **Feature Ranking**: The selection of features were ranked by importance scores. It helps us to identify the best N features that are most important for the model to do the ranking.

- **Visualization**: The list of became a horizontal bar chart displaying the feature importances This visualization helps to spot the important features In no time.

Now that we have a prepared dataset, next thing is we need to train different types of machine learning models using the same for detecting malware effectively. This means testing different models, hyperparameter tuning, and using metrics to measure performance of a model — like accuracy, precision breakout and f1-score. The end goal is to build a strong model which can classify whether Android applications are benign or malicious based on behavior and feature associations.

4.3 Exploratory Analysis of the Dataset

The Fig. 4.1 illustrates how the class labels are distributed in our dataset, for each of the Android malware classes (VirusShare.A and Genome respectively) along with benign samples. The samples are mostly dominated by the Android Adware class, and then followed by the categories of Android Scareware, and Android SMS Malware. This imbalance stresses the importance of ensuring that when training a model, it does not become biased towards simply predicting that most/all individuals are non-recidivist.

Moreover, the class distribution indicates a severe level of imbalance especially for the minority class ' Benign'. The Android Adware class is the major class (with an imbalance ratio of 6.22), as it has around 6.22 times more instances compared to the minority class (as Fig. 4.2). Similar to the above mentioned example, the Android Scareware class in this family has the imbalance ratio of 4.94 which means that nearly 4.94 times more instances of majority than minority classes are there in it. Additionally, the Android SMS Malware class remains significantly imbalanced

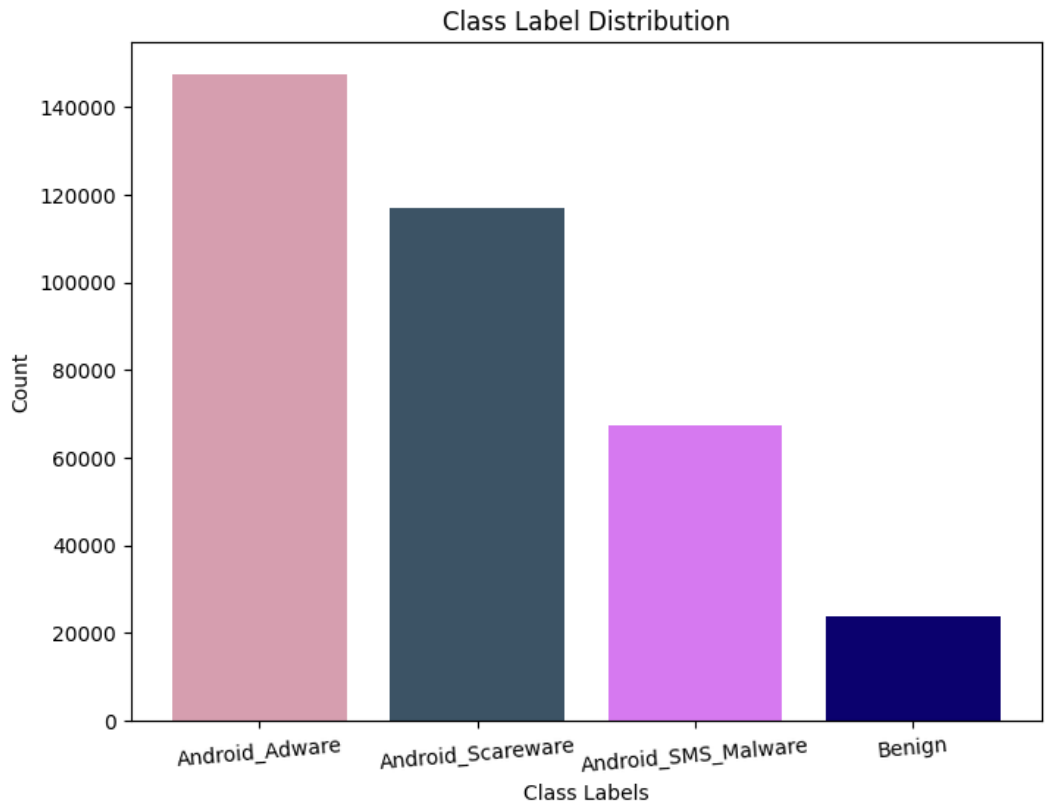

Figure 4.1: Class label distribution in the dataset.

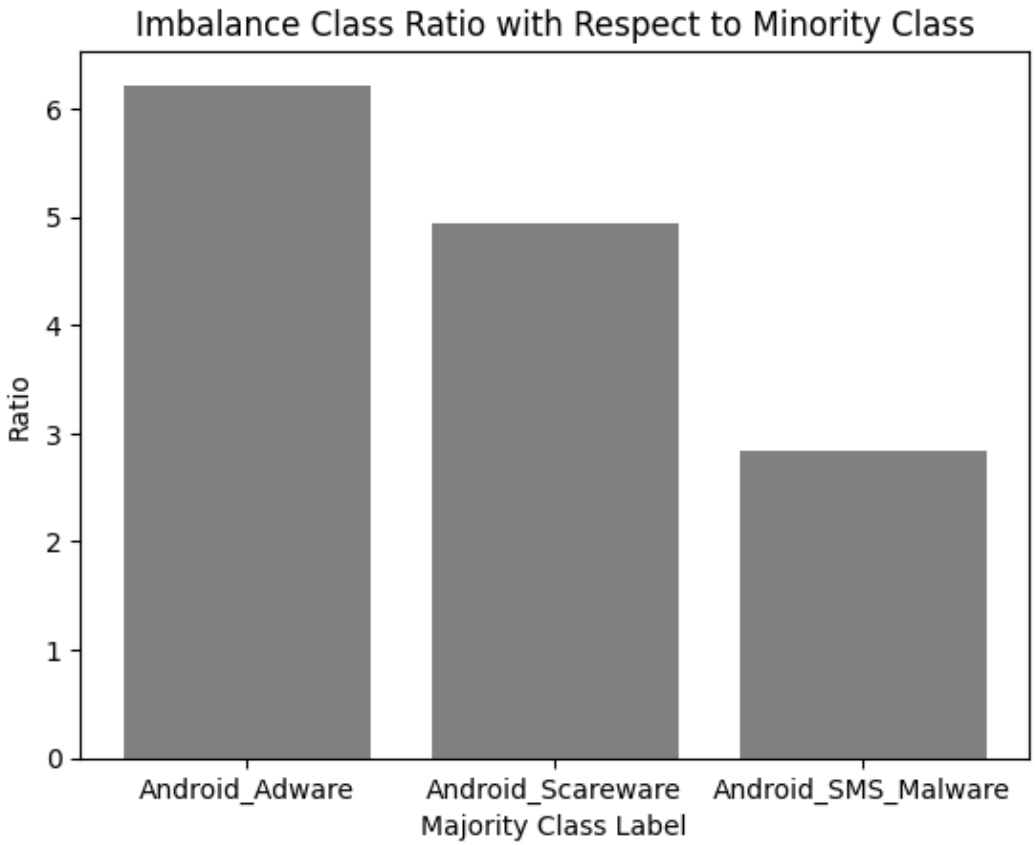

Figure 4.2: Imbalance class ration with respect to Benign class.

with a ratio of unbalancing close to 2.84 for the minority over the majority class. The ratios demonstrate the vast disparity between the majority classes and the minority class, highlighting the cassification.

The dataset sample used in this study consist of 42 different attributes with missing values. Fig. Visualization and null value counts for columns that have missing values in our dataset are shown in the image 4.3. All the bars in the plot represent all null values of a column and its height indicates how many null entries each columns have. In the x-axis, we have x-tick labels as names of the columns which are having missing values and for y-axis it shows count of na values in that particular column. This visualization enables a quick look at how much data we are missing in different features of our dataset. The amount of columns that have null

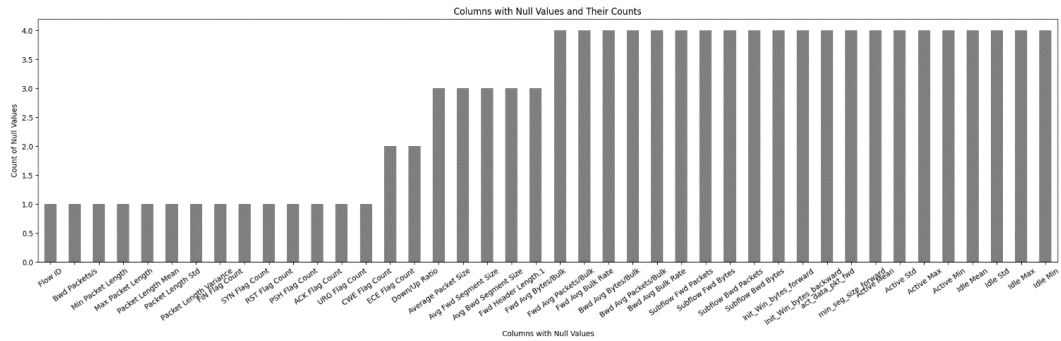

Figure 4.3: Visualization of columns with null values in the dataset used for this study, aiding in assessing missing data across features.

values are also shown under the graph.

This could potentially take a little while to pinpoint what the key features for detecting malware in android are, but hey that is why we experiment and analyse! Nevertheless, there are features that are typically seen as more useful as they may help to identify malicious behavior. Following are few common features that most android malware detection consider:

- **Source IP and Destination IP**: These can indicate suspicious communication patterns, especially if the source IP is associated with known malicious servers or if the destination IP is an unusual or suspicious address.
- **Source Port and Destination Port**: Unusual port usage or connections to well-known malware-related ports can be indicative of malicious activity.
- **Protocol**: Certain protocols may be more commonly used by malware (e.g., UDP for command and control communication), so analyzing protocol usage can be insightful.
- **Flow Duration**: The duration of a flow can indicate whether the communication pattern is typical or anomalous.
- **Packet Length Statistics (Max, Min, Mean, Std)**: Anomalies in packet

length distributions may suggest malicious activity, such as encryption or obfuscation techniques used by malware.

- **Flow Bytes/s and Flow Packets/s**: Unusually high or low rates of flow bytes and packets may indicate suspicious activity, such as a DDoS attack or data exfiltration.

- **Flag Counts (e.g., FIN, SYN, RST, PSH, ACK, URG)**: Anomalous flag counts or combinations of flags can be indicative of specific types of attacks or malware behavior.

- **Packet Length Variance**: High variance in packet lengths may indicate abnormal packet fragmentation or manipulation, which could be a sign of malicious activity.

- **Init_Win_bytes_forward and Init_Win_bytes_backward**: Window size information can be relevant for detecting certain types of attacks, such as TCP-based attacks.

- **Active and Idle Times (Mean, Std, Max, Min)**: Anomalies in active and idle times can indicate abnormal behavior, such as prolonged periods of activity or inactivity associated with malware execution or evasion techniques.

- **Total Fwd Packets and Total Backward Packets**: Unusually high numbers of packets in either direction may suggest suspicious activity, such as data exfiltration or reconnaissance.

- **Label**: This is the ground truth indicating whether the flow is associated with malware. While not a feature to be used in the detection model itself, it is crucial for training and evaluation.

The graph (as shown in Fig. 4.4) is used to express the flow duration patterns and features of flows for each flow classes, which can be used to analyze and compare the two in turn. On the Horizontal Axis (Flow Duration) we can visualize how long flows

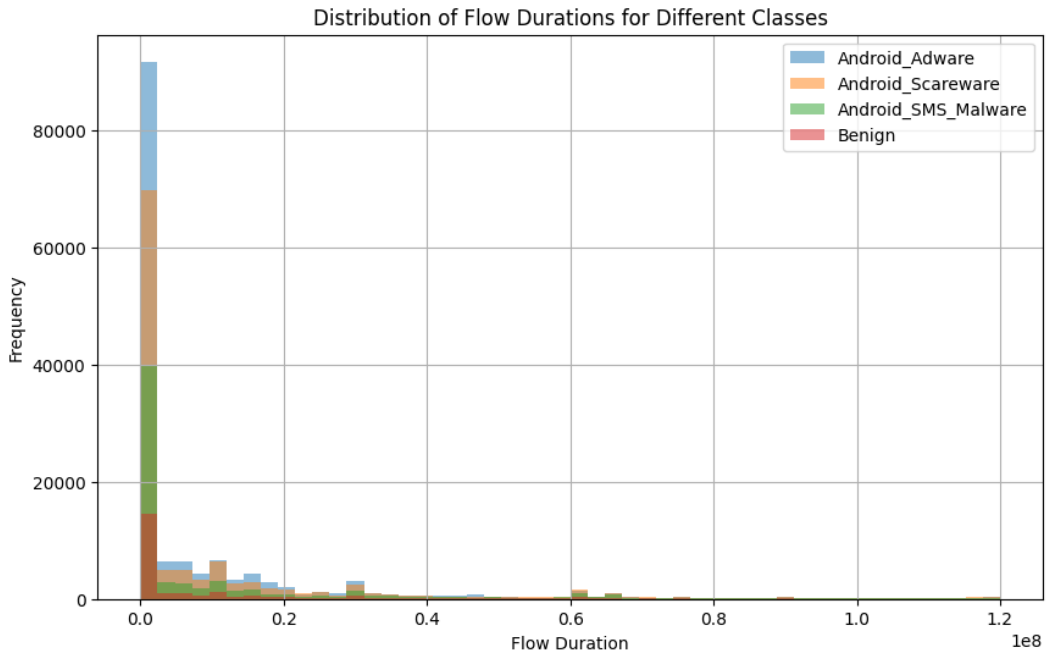

Figure 4.4: Distribution of Flow Durations for Different Classes.

stays on average and in each bin represents a different mean flow duration. Figure 2: Vertical Axis (Frequency) vs. On the other hand of flow duration falling within the range on the Horizontal Axis Ø Number of flows in each interval [Frequency axis] Fig. Also, the histogram bars represent how long flows take to complete for Android Adware, Andrew Scareware, and Android SMS Malware classes as well as Benign flows. The height of the bar represents how often flow durations occur within each respective range on the Horizontal Axis, which can then be compared with other classes of flows.

The graph (as shown in Fig. 4.5) variances that are given across flows in diverse classes, looking for indications of aberrations or methodologies regarding to malevolent network ways. For instance, a packet length variance of 0 means different possible things depending on flow specifics and networking context. In some cases, it may be typical behavior, but in others (relatively rare) behavior within your service is an abnormality and a threat that needs to be investigated further.

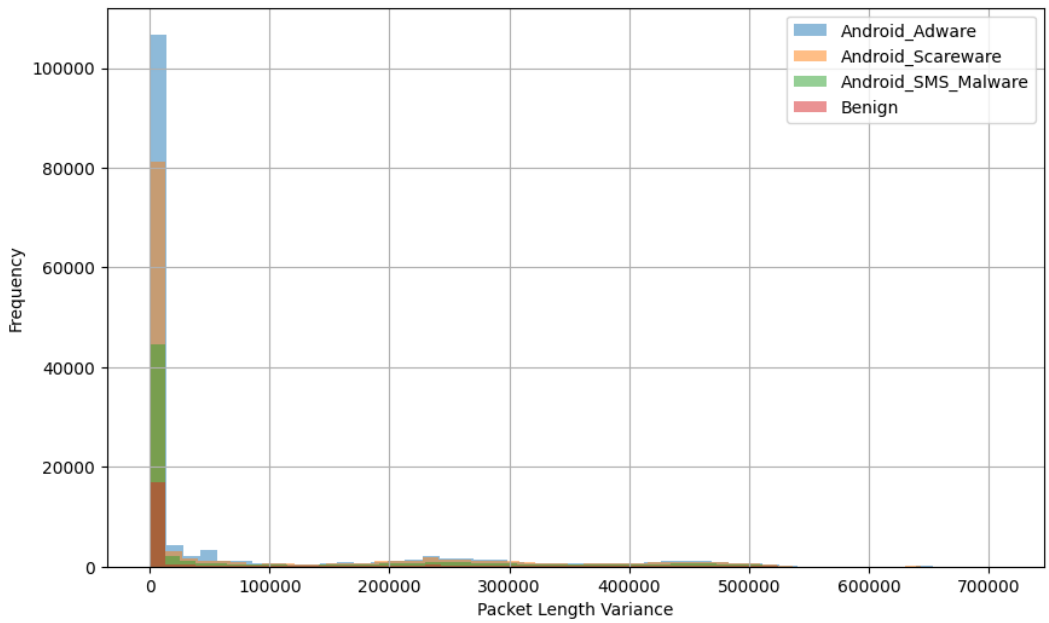

Figure 4.5: Distribution of Packet Length Variance for Different Classes.

The Figure 4 shows the distribution of packet length variance for different classes. This is crucial for Feature selection, model building and so on. Figure: The figure below shows a correlation heatmap between to visualize the pairwise correlations between features in our dataset. The heatmap (in Fig. This figure (4.6) offers a way of looking at all the features in relation to each other. In the heatmap, each cell is the correlation coefficient between two features as a value from -1 to 1. If the value is close to 1 it means then there is a strong positive correlation, if it is near -1 this tells us that there is a strong negative correlation. A value of 0 would indicate no correlation. High positive correlation is represented by yellow and high negative correlation is blue, which make these relationships easy to identify due to the color gradient from dark blue to yellow. The heatmap show that some features are highly correlated with just one another. A lot of these features do seem to cluster together (in particular, all the various packet length and count related features appear in a couple different groups) suggesting that they are measuring similar things about network traffic. It is important to find these highly correlated

feature because they could be redundant. This will generate the exact error that we are trying to avoid: overfitting with lots of redundant features, and a huge corpus that actually worsens our model. Do not allow registered user model performance. Along with the correlation heatmap, histograms for all features were plotted to comprehend the distributions. These histograms provided valuable insights into the data, including the skewness in the data, existence of outlying values and the overall distribution of values. Visualizations were used to understand how features are distributed and whether they need transformation or normalization before passed into the machine learning algorithms.

4.4 Result Analysis

This section represents the result analysis of machine learning approaches used in this work.

4.4.1 Feature Importance Analysis

Random Forest model is trained resulting as a 99%accuracy of features that play most important roles in classifying the Android malware. The importance scores from the model gave us an added perspective into which features help in making accurate predictions. The relative importance scores of all features were visualized in an explanatory plot as a bar chart (Figure 4.7). This is the visualization that allows us to interpret which features mainly explain the predictive power of the model. Top features are listed by importance from most to least important, making it easy to identify the top contributors. Feature importance: Analysis in the way of visualization is critical for not only selection but also understanding which features are driving your model. This can help us to build a simpler model that will be computationally fast and we could help the performance of our model by removing noisy or redundant features. As we analyze the top features in Figure 4.7 are:

1. **Fwd Avg Bytes/Bulk**: 0.2501

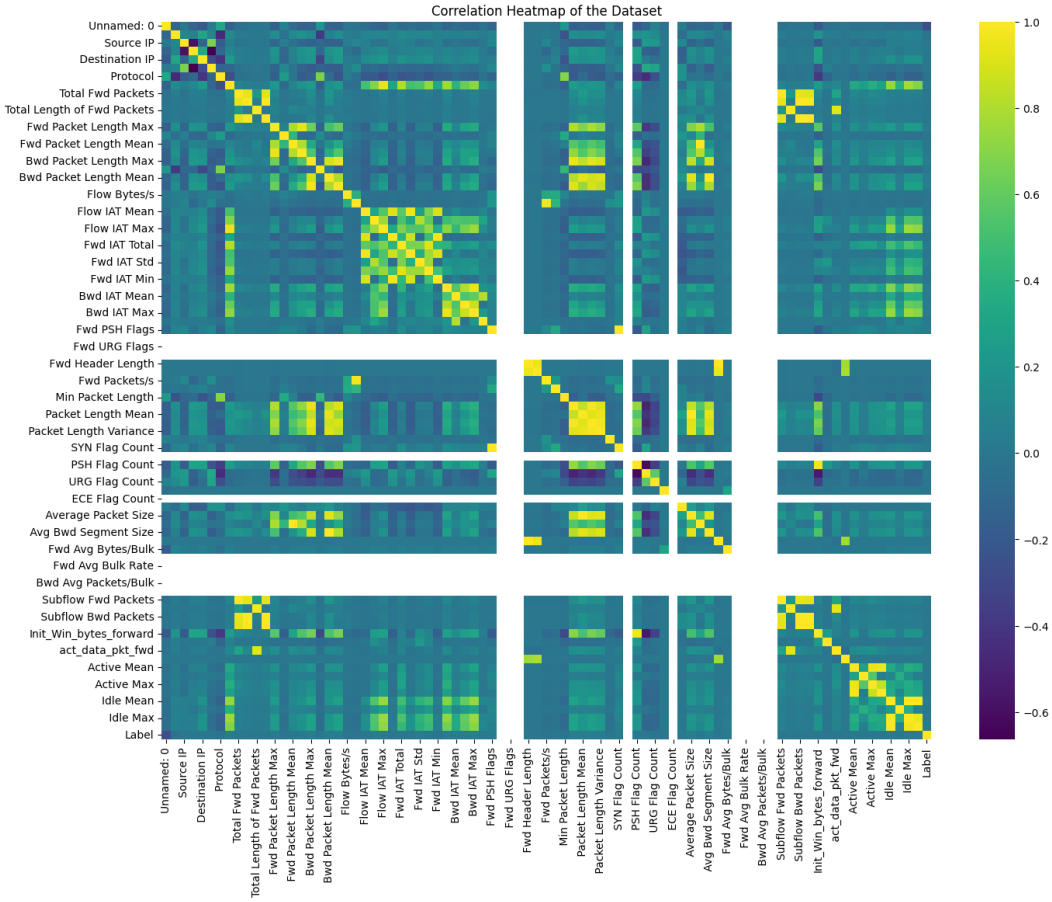

Figure 4.6: Visualizing pairwise correlations between features to identify highly correlated and potentially redundant features for better feature selection in model building.

2. **Unnamed: 0**: 0.0941
3. **Flow ID**: 0.0547
4. **Source Port**: 0.0368
5. **Destination IP**: 0.0315
6. **Source IP**: 0.0300
7. **Flow IAT Min**: 0.0261
8. **Flow IAT Max**: 0.0255
9. **Flow Duration**: 0.0247
10. **Fwd Packets/s**: 0.0244

The features identified as the most important for Classifier of choice Random Forest The most important functionality is the "Fwd Avg Bytes/Bulk" with a large margin to many of the others. That is to say, an important judgment dimension of whether it is malicious or not lies in the average number of bytes per bulk forward transmission in Android applications. Furthermore, the characteristics from network flow (e.g., "Flow ID," "Source Port" and "Destination IP") and inter-arrival time such as ("Flow IAT Min" and Flow IAT Max") were significant. While the exact reason for these features is unclear, although they provide potentially important features of network behavior from benign applications to malware.

4.4.2 Model Performance Analysis

We implemented a wide range of machine learning models to test the effectiveness of different classifiers in detecting Android malware. The classifiers we used in this work are:

- Gradient Boosting
- Random Forest

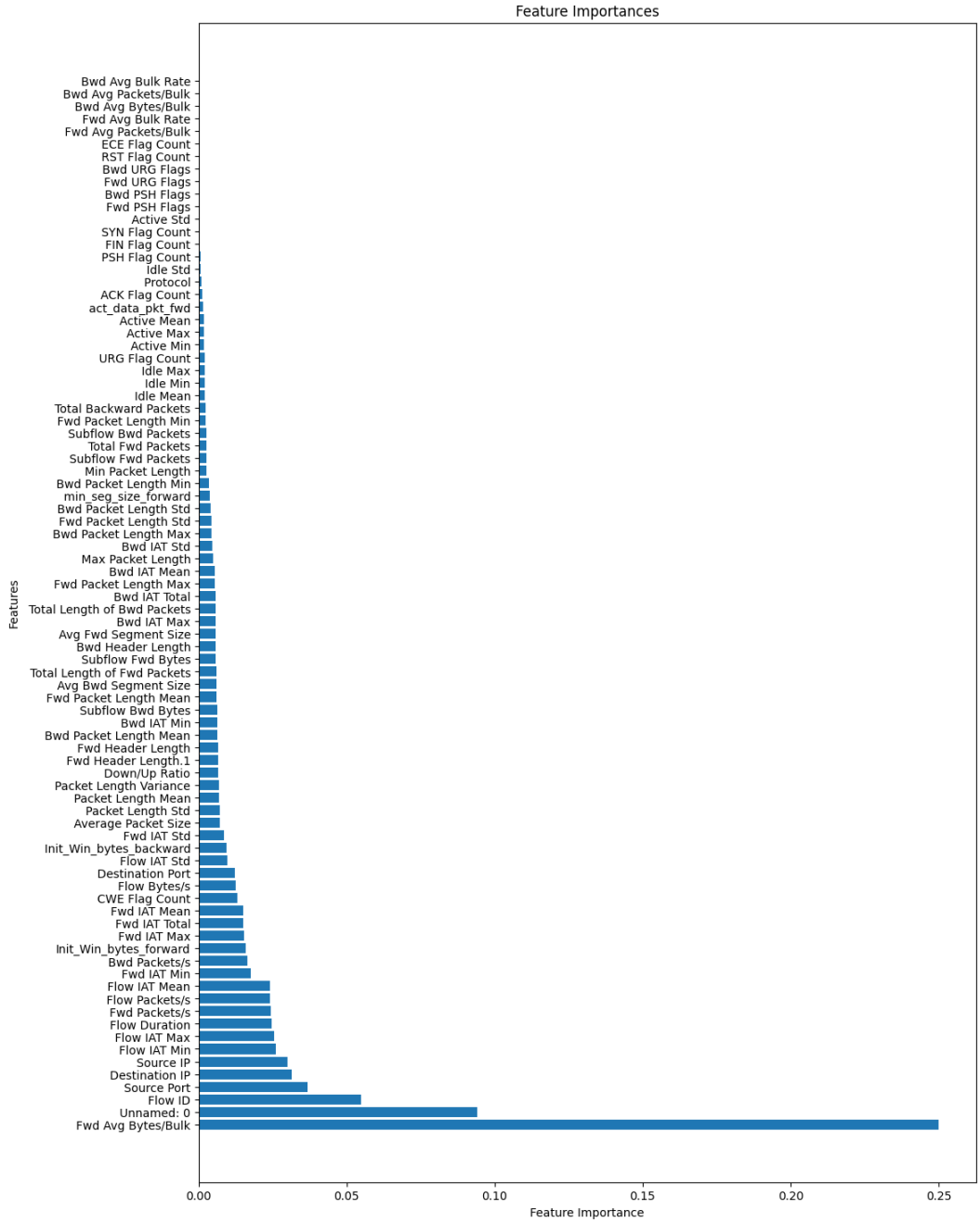

Figure 4.7: Visualization of the top features contributing to Android malware detection.

- AdaBoost
- Decision Tree
- Multi-layer Perceptron (MLP)
- Bagging Classifier
- Logistic Regression
- K-Nearest Neighbors (KNN)
- Naive Bayes (Gaussian)

Every model is trained based on the training set which of course is 80% of that in data, then subsequently evaluated against the test set. The accuracy, precision, recall, and F1-score were the most important metrics for evaluation. Finally, confusion matrices were created to see how the models performed in terms of true positives, true negatives, false positives and false negatives. Figure 4.8 shows the top ten features extracted, also exhibit source code used for training models.

4.4.2.1 Performance Analysis of Classifiers

In this section, we provide a thorough evaluation of the classifiers analyzed with their efficiency in detecting malware. The performance metrics of individual classifiers (precision, recall, F1-score and accuracy) are reviewed in Table 4.1.

Our studying ultimately provides us with various clues as to how the different classifiers have performed:

- **Gradient Boosting** obtained a performance equilibrium among precision, recall, F1-score and accuracy at 0.70 This indicates that the result is stable with an acceptable performance for applications in malware recognition and anti-malware services, as well as to reduce false positives.
- **Random Forest** was the best performing model, obtain the highest precision (0.81), recall (0.81), F1-score (0.80), and accuracy(0.80). The approach used

Table 4.1: Performance Metrics of Classifiers

Classifier	**Precision**	**Recall**	**F1-Score**	**Accuracy**
Gradient Boosting	0.70	0.70	0.70	0.70
Random Forest	0.81	0.81	0.81	0.81
AdaBoost	0.62	0.52	0.53	0.52
Decision Tree	0.81	0.81	0.81	0.81
Multi-layer Perceptron	0.52	0.42	0.24	0.42
Bagging Classifier	**0.84**	**0.84**	**0.84**	**0.84**
Logistic Regression	0.34	0.42	0.31	0.42
K-Nearest Neighbors	0.49	0.50	0.49	0.50
Naive Bayes (Gaussian)	0.38	0.33	0.24	0.33

for ensemble-based mechanism is extremely successful because handling a variety of intricate patterns and complexities relates to the characteristic inherent in the malware detection task.

- **AdaBoost**, on the other hand, was a lot weaker at 0.52 indicating the problems possibly related to a lot of class imbalance or limitations on model adaptation to the look of data.

- **Decision Tree** performance on a similar level to Random Forest, which could hint at the overall predictive strength of the decision tree by its hierarchical nature.

- **Multi-layer Perceptron** performed the worst, especially in recall and F1-score, which hints at possible problems with model tuning or lack of specificity to this use-case.

- **Bagging Classifier** showed a promising and consistent high value performance like Random Forest getting standout metrics and reached 0.84 for precision, recall, F1-score, as well as accuracy.

- **Logistic Regression** had a lower precision and F1-score, with an accuracy

of 0.42 signifying difficulties capturing the complex relationships within the dataset.

- **K-Nearest Neighbors** did so-so with an accuracy of 0.50, performing weaker in capturing patterns when compared to tree-based and ensemble methods.

- **Naive Bayes (Gaussian)** showed the worst performance among others specially in recall and F1-score due to its assumption of independence among features that is not accurate for more complex relationship in malware datasets.

The insights obtained from Table 4.1 indicates productivity of different classifiers, and we see from above plot that Bagging Classifier was best performing classifier in our study, if you observe the table, it is having high precision, recall, F1-score and accuracy giving correct prediction of malware sample. These findings highlight the necessity of using strong ensemble methods or decision tree based techniques to ensure effective performance in malware detection dumps. The comparative study also reinforced the importance of selected models based on an accurate understanding of complexity and required properties from different tasks. as shown in Figure 4.8.

4.4.3 Comparative Analysis

This section delivers a comparative study of different techniques that are being used in the domain of Android malware detection based on their respective approach, techniques and results. In this study, we use Ensemble Methods—specifically Random Forest and Bagging Classifier—to overcome the problem of classifying malicious applications with a dataset taken from kaggle. We summarize the outcomes of our approach in Table 4.2, which shows excellent precision, recall, F1-score and accuracy metrics (0.81 for Random Forest; 0.84 for Bagging Classifier). Ensemble methods provide a new idea for the analysis of malware patterns again and have good performance in various scenes.

Bakır et al. Fazly and Mendz [8], examined the potential of utilizing Droid Encoder—an auto encoder that extracts abstracted features from APKs over various

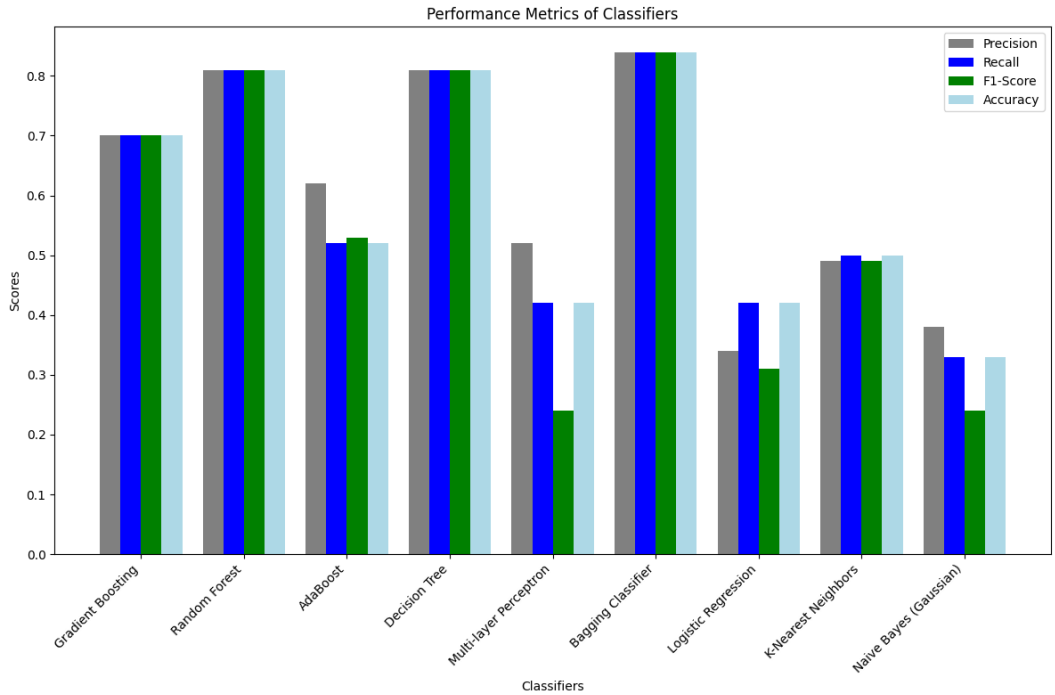

Figure 4.8: Comparative analysis of the different classifiers.

diverse and intricate behaviors of malware. It follows a rigorous feature extraction methodology as opposed to the data-level fusion adopted by ensemble methods, and hence can be used for understanding the subtleties of Android malware behaviors that may not be captured with ensemble models. Conversely, Hefter et al. In addition, the challenges of metadata-based detection were pointed out by Zhang [4], also arguing that this class of method is susceptible to evasion techniques using obfuscated metadata. Our ensemble based approach addresses some of these concerns by focusing more on sound feature-based detection than metadata reliance.

The studies by Yerima et al. [7] and Chaieb et al. Ref. [19] also add to the comparative framework. Yerima et al. demonstrated that ensemble-based learning provided high accuracy in detecting malware, thus it is a reliable model for handling various datasets and types of malwares. In contrast, Chaieb et al. [20] explained computational complexities and training that one can expect while creating the BERTroid (BERT for malware detection) architecture. Our results do differ—the

ensemble methods we examined might not provide much if any improvement over other techniques, but they are still competitive and come with less computational cost making them well suited for real-time applications in mobile security.

Finally, this work results can be used to improve Android malware detection. Through its strong performance and contribution towards addressing the inevitable challenges in malware behavior detection, our work clues in valuable revelations into cyber security practice — especially as most threats today, especially within mobile environments drive away from low level detection mechanisms.

4.5 Discussion

It was a study about Android malware detection with a dataset which only came from Dataset: From Kaggle with 355,630 instances and 85 features. The main purpose was to study real life Android malware dataset, develop machine learning based models and ensemble learning methods in particular, Random forest and Bagging classifier for the task of accurately classifying Android applications as malicious or benign based on their behavior.

This echoes the relative success of these most robust ensemble learning techniques, in particular Random Forest and Bagging Classifier. Those models showed good precision, recall, F1-score and accuracy across all classifiers — it outperformed other ones like AdaBoost, Multi-layer Perceptron and Logistic Regression. With its ensemble-based approach, Random Forest could capture the complex patterns and relationships within the data set while still being able to mitigate overfitting and handle feature correlations effectively.

The results were, therefore, interpreted in context after analyzing them comparatively (with different methodologies) which assisted to bring context for the results. Studies by Bakır et al. [8] and Yerima et al. Same for MAVs where previous work on this topic to achieve high accuracy was also by using ensemble techniques [19]. A notion supported by these comparisons is the robustness and efficacy of ensemble methods across datasets and malicious software families, which lends itself to

real-world cybersecurity settings.

This research poses further consequences for use of this model in cybersecurity applications. This study also recommends to implement ensemble methods for an improved malware detection system deployed in mobile security domains. We plan to pursue future research directions such as sophisticated feature engineering, deep learning architectures designed for mobile security tasks and adversarial learning methods specially developed to improve the robustness against new evasion tactics proposed by malware authors.

In addition, investigation into real-time deployment situations and the need to scale will further ensure the operational relevance of these techniques in cybersecurity systems. With mobile malware steadily increasing in innovation and complexity, refinement and adaptation of detection techniques will be essential to remain ahead of the curve.

The reliable performance of ensemble methods in this study did have certain limitations that should not go unsaid. This involves the computational complexity associated with training and deploying ensemble models on resource-constrained mobile devices and biases that might be introduced due to dataset depictions and feature engineering choices. Solving these problems will be key towards making malware detection solutions practical and thus more reliable and scalable.

4.6 Summary

This chapter examines the efficacy of different ML models in the detection of Android malware, specifically focusing on ensemble methods such as Random Forest and Bagging Classifier. These models showed high values for precision, recall, F1-score and accuracies(0.81 and0.84) respectively, in comparison with other classifiers (Gradient Boosting, AdaBoost or Logistic Regression). The results from the analysis show that ensemble methods have the ability of dealing correctly with high organized malware into complex patterns, and that is represented by in "Fwd Avg Bytes/Bulk" feature. Some comparative insights from earlier studies validate the robustness and

fidelity of ensemble learning; however, competing methods reach complexity and evasion risks compared to both metadata-based detection models as well as BERT architectures.

Table 4.2: Comparative Analysis of Malware Detection Approaches

Study	**Approach and Methods**	**Performance Insights**
This Study	Ensemble Methods: Random Forest, Bagging Classifier	Achieved high precision, recall, F1-score, and accuracy (0.81 for Random Forest, 0.84 for Bagging Classifier). Demonstrated robust performance in detecting malware patterns and complexities.
Bakır et al. [8]	DroidEncoder: Auto-encoder based feature extraction	Focuses on feature extraction using auto-encoder, potentially addressing diverse and complex malware behaviors differently from ensemble methods.
Hefter et al. [7]	Metadata-based detection	Vulnerable to evasion techniques using obfuscated metadata. Your approach with ensemble methods avoids some of these vulnerabilities by focusing on feature-based detection.
Yerima et al. [19]	Ensemble Learning Techniques	Achieved high accuracy in malware detection. Your study reinforces the effectiveness of ensemble methods with similar or superior performance metrics.
Chaieb et al. [20]	BERTroid: Malware detection with BERT architecture	Focuses on computational complexity and training requirements with BERT architecture. Your ensemble methods provide competitive performance with potentially lower computational overhead.

Chapter 5

Conclusions and Future Work

In this chapter, we summarize the research works presented in this dissertation and make final concluding remarks with a few directions for future works.

5.1 Summary of the Dissertation

The dissertation investigated the intricacies of conduction malware detection in Android using machine learning (ML) techniques. Leveraging such data, we then covered major drawbacks of standard detection routines, which mostly are overwhelmed by the speed with which mobile threats progress. Our results also showed that ensemble methods such as Random Forest and Bagging Classifier improves the detection accuracy, precision, recall and F1-score. The systematic approach used in this study consists of data collection, preprocessing, feature engineering, model building and tuning as well as exhaustive performance evaluation. More important, is that the inclusion of Explainable AI techniques ensured transparency and interpretability to make sense about why it made predictions.

The results suggest that machine learning methods can provide large gains in the ability to detect sophisticated malware patterns, and proper dataset regularization is necessary for combating classification biases. In short, our study provides important observations which may be used as guidelines to better deploy and develop advanced detection systems for the protection of mobile devices users.

5.2 Future Research Directions

The next studies will probably try to improve detection of Android malware with:

- **Incorporation of Deep Learning**: Fusion of DL runners in the hybrid models could help to catch more compound malware behavioral patterns and thus enhance detection rates.
- **Real-Time Adaptation**: Building upon security frameworks, which feature continuous learning methods to update the model so as to include new emerging malware, is critical toward maintaining its success for real-world applications.
- **Feature Expansion**: Creating systems, for learning to enable the model to adjust to emerging malware threats in time will be essential for ensuring effectiveness, in practical scenarios.
- **Cross-Platform Detection**: Exploring features, like studying behavior and analyzing network traffic patterns may improve the models ability to detect threats that were previously unknown.
- **User Privacy and Ethical Considerations**: Expanding the study to incorporate methods that can detect across platforms may enhance the significance and practicality of the results.

In research it is important to explore the ethical aspects and privacy issues associated with gathering and analyzing data, for mobile malware detection. By following these instructions and implementing them effectively the realm of detecting Android malware can make strides offering strong defense against the increasing variety of cybersecurity risks.

Bibliography

[1] M. Kim, D. Kim, C. Hwang, S. Cho, and M. Park, "Machine-learning-based android malware family classification using built-in and custom permissions," *Applied Sciences*, vol. 11, p. 10244, 2021.

[2] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini, and E. De Cristofaro, "A family of droids-android malware detection via behavioral modeling: Static vs dynamic analysis," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2018, pp. 1–10.

[3] D. Li, L. Zhao, Q. Cheng, N. Lü, and W. Shi, "Opcode sequence analysis of android malware by a convolutional neural network," *Concurrency and Computation: Practice and Experience*, vol. 32, 2019.

[4] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[5] W. Niu, R. Cao, X. Zhang, K. Ding, K. Zhang, and T. Li, "Opcode-level function call graph based android malware classification using deep learning," *Sensors*, vol. 20, p. 3645, 2020.

[6] T. Lu, Y. Du, L. Ouyang, Q. Chen, and X. Wang, "Android malware detection based on a hybrid deep learning model," *Security and Communication Networks*, vol. 2020, no. 1, p. 8863617, 2020.

[7] A. Hefter, C. Sendner, and A. Dmitrienko, "Metadata-based malware detection on android using machine learning," *arXiv preprint arXiv:2307.08547*, 2023.

[8] H. Bakır and R. Bakır, "Droidencoder: Malware detection using auto-encoder based feature extractor and machine learning algorithms," *Computers and Electrical Engineering*, vol. 110, p. 108804, 2023.

[9] Z. Iqbal, M. Altaf, and A. Shahid, "Unraveling ransomware in the digital battlefield: Threat analysis and countermeasures," *Lahore Garrison University Research Journal of Computer Science and Information Technology*, vol. 7, no. 4, 2023.

[10] T. Chow, Z. Kan, L. Linhardt, L. Cavallaro, D. Arp, and F. Pierazzi, "Drift forensics of malware classifiers," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023, pp. 197–207.

[11] P. G. Balikcioglu, M. Sirlanci, O. A. Kucuk, B. Ulukapi, R. K. Turkmen, and C. Acarturk, "Malicious code detection in android: the role of sequence characteristics and disassembling methods," *International Journal of Information Security*, vol. 22, no. 1, pp. 107–118, 2023.

[12] M. T. Alam, R. Fieblinger, A. Mahara, and N. Rastogi, "Morph: Towards automated concept drift adaptation for malware detection," *arXiv preprint arXiv:2401.12790*, 2024.

[13] A. Muzaffar, H. R. Hassen, H. Zantout, and M. A. Lones, "Actdroid: An active learning framework for android malware detection," *arXiv preprint arXiv:2401.16982*, 2024.

[14] M. Maray, M. Maashi, H. M. Alshahrani, S. S. Aljameel, S. Abdelbagi, and A. S. Salama, "Intelligent pattern recognition using equilibrium optimizer with deep learning model for android malware detection," *IEEE Access*, vol. 12, pp. 24 516–24 524, 2024.

[15] S. Malik and N. Sharma, "A vast review of recognizing the presence of android malware based on ensemble machine learning technique," *Indian Journal of Science and Technology*, vol. 17, pp. 149–165, 2024.

[16] J. Zhang, K. Xia, Z. Huang, S. Wang, and R. G. Akindele, "Etam: Ensemble transformer with attention modules for detection of small objects," *Expert Systems with Applications*, vol. 224, p. 119997, 2023.

[17] Y.-S. Cho, "Decoupled variational graph autoencoder for link prediction," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 839–849.

[18] S. J. Hamdi, N. Omar, A. AL-zebari, K. J. Merceedi, A. J. Ahmed, N. O. Salim, S. S. Hasan, S. F. Kak, I. M. Ibrahim, H. M. Yasin *et al.*, "A state of art survey for understanding malware detection approaches in android operating system," *Asian Journal of Research in Computer Science*, vol. 11, no. 3, pp. 44–60, 2021.

[19] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Information Security*, vol. 9, no. 6, pp. 313–320, 2015.

[20] M. Chaieb, M. A. Ghorab, and M. A. Saied, "Detecting android malware: From neural embeddings to hands-on validation with bertroid," *arXiv preprint arXiv:2405.03620*, 2024.

[21] J. M. Vidal, M. A. S. Monge, and L. J. G. Villalba, "A novel pattern recognition system for detecting android malware by analyzing suspicious boot sequences," *Knowledge-Based Systems*, vol. 150, pp. 198–217, 2018.

[22] X. Meng, "Hertdroid: Android malware detection method with influential node filter and heterogeneous graph transformer," *Applied Sciences*, vol. 14, no. 8, p. 3150, 2024.

[23] L. Liu, W. Ren, F. Xie, S. Yi, J. Yi, and P. Jia, "Learning-based detection for malicious android application using code vectorization," *Security and Communication Networks*, pp. 1–11, 2021.

[24] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, Y. Wang, and Y. Xiang, "A3cm: Automatic capability annotation for android malware," *IEEE Access*, vol. 7, pp. 147 156–147 168, 2019.

[25] R. Srinivasan, S. Karpagam, M. Kavitha, and R. Kavitha, "An analysis of machine learning-based android malware detection approaches," *Journal of Physics: Conference Series*, vol. 2325, no. 1, p. 012058, 2022.

[26] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124 579–124 607, 2020.

[27] B. Tang, "Mudroid: Android malware detection and classification based on permission and behavior for autonomous vehicles," *Transactions on Emerging Telecommunications Technologies*, vol. 34, no. 11, 2023.

[28] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21 235–21 245, 2019.

[29] D. Li, L. Zhao, Q. Cheng, N. Lü, and W. Shi, "Opcode sequence analysis of android malware by a convolutional neural network," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 18, 2019.

[30] "Untitled," *ACM Transactions on Software Engineering and Methodology*, vol. 26, no. 3, 2018.

[31] S. Aurangzeb and M. Aleem, "Evaluation and classification of obfuscated android malware through deep learning using ensemble voting mechanism," *Scientific Reports*, vol. 13, no. 1, 2023.

[32] T. S. John and T. Thomas, "Evading static and dynamic android malware detection mechanisms," in *Security in Computing and Communications: 8th International Symposium, SSCC 2020, Chennai, India, October 14–17, 2020, Revised Selected Papers 8*. Springer, 2021, pp. 33–48.

[33] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection: An eigenspace analysis approach," in *2015 Science and Information Conference (SAI)*. IEEE, 2015, pp. 1236–1242.

[34] H. Rathore, S. K. Sahay, S. Thukral, and M. Sewak, "Detection of malicious android applications: Classical machine learning vs. deep neural network integrated with clustering," in *International conference on broadband communications, networks and systems*. Springer, 2020, pp. 109–128.

[35] M. J. Iqbal, S. Aurangzeb, M. Aleem, G. Srivastava, and J. C.-W. Lin, "Rthreatdroid: A ransomware detection approach to secure iot based healthcare systems," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 5, pp. 2574–2583, 2022.

[36] H. Kumar, S. Sharma, B. Chakraborty, and S. Mukhopadhyay, "Towards robust real-time hardware-based mobile malware detection using multiple instance learning formulation," *arXiv preprint arXiv:2404.13125*, 2024.

Appendix A

Appendix

A.1 Snapshot of Dataset

A1:

Flow ID	Source IP	Source Port	Destination	Destination	Protocol	Timestamp	Flow Durat	Total Fwd F	Total Backs	Total Length	Total Length	Fwd Packet	Fwd Packet	Fwd Packet	Fwd Packet	Bwd Packet	Bwd Packet	Bwd Packet	Bwd Packet	Flow Bytes/	Flow Packe	Flow IAT M	Flow IAT St	Flow IAT M	Flow IAT M
172.217.6.2	10.42.0.211	50004	172.217.6.2	443	6	13/06/2017	37027	1	1	0	0	0	0	0	0	0	0	0	0	0	54.01638	37027	0	37027	37027
172.217.6.2	10.42.0.211	35455	172.217.6.2	443	6	13/06/2017	36653	1	1	0	0	0	0	0	0	0	0	0	0	0	54.56793	36653	0	36653	36653
131.253.61.	10.42.0.211	51775	131.253.61.	443	6	13/06/2017	534099	8	12	1011	11924	581	0	126.375	207.79931	1460	0	993.66667	656.47438	24218.357	37.446241	28110.474	43148.095	158676	8
131.253.61.	10.42.0.211	51775	131.253.61.	443	6	13/06/2017	9309	3	0	0	0	0	0	0	0	0	0	0	0	0	322.26877	4654.5	5137.1308	8287	1022
131.253.61.	10.42.0.211	51776	131.253.61.	443	6	13/06/2017	19890496	8	6	430	5679	218	0	53.75	99.538578	1460	0	946.5	710.4122	307.13161	0.7038537	1530038.2	5377887.2	19426714	7
10.42.0.211	10.42.0.211	50186	23.208.43.1	443	6	13/06/2017	19196263	5	2	0	0	0	0	0	0	0	0	0	0	0	0.3646543	3199377.2	5886828.2	14995041	3915
10.42.0.211	10.42.0.211	50186	23.208.43.1	443	6	13/06/2017	4304325	1	2	0	0	0	0	0	0	0	0	0	0	0	0.6969734	2152162.5	3007423.4	4278732	25593
172.217.3.1	10.42.0.211	40087	172.217.3.1	443	6	13/06/2017	37926	1	2	23	0	23	23	23	0	0	0	0	0	0	606.44413	79.101408	18963	25819.297	37220	706
172.217.3.1	10.42.0.211	52564	172.217.3.1	443	6	13/06/2017	36631	1	2	23	0	23	23	23	0	0	0	0	0	627.88349	81.897846	18315.5	25708.281	36494	137
10.42.0.211	10.42.0.211	46130	31.13.71.3	443	6	13/06/2017	36535	1	2	0	31	0	0	0	0	31	0	15.5	21.92031	848.50144	82.113042	18267.5	25752.122	36477	58
10.42.0.211	10.42.0.211	47658	31.13.71.1	443	6	13/06/2017	221194	2	0	0	0	0	0	0	0	0	0	0	0	0	9.0418366	221194	0	221194	221194
10.42.0.211	10.42.0.211	47658	31.13.71.1	443	6	13/06/2017	914675	2	0	0	0	0	0	0	0	0	0	0	0	0	2.186569	914675	0	914675	914675
10.42.0.211	10.42.0.211	47658	31.13.71.1	443	6	13/06/2017	3692864	2	0	0	0	0	0	0	0	0	0	0	0	0	0.5415851	3692864	0	3692864	3692864
10.42.0.211	10.42.0.211	47658	31.13.71.1	443	6	13/06/2017	14770330	2	0	0	0	0	0	0	0	0	0	0	0	0	0.1354066	14770330	0	14770330	14770330
10.42.0.211	10.42.0.211	49564	52.221.112.	7077	6	13/06/2017	65695129	4	4	568	442	568	0	142	284	442	0	110.5	221	15.374047	0.1217746	9385018.4	24522844	64997021	2496
10.42.0.211	10.42.0.211	33026	121.41.26.1	8090	6	13/06/2017	65823449	4	4	279	1200	279	0	69.75	139.5	1200	0	300	600	22.469196	0.1215372	9403349.9	24464704	64883252	1724
10.42.0.211	10.42.0.211	47706	31.13.71.1	443	6	13/06/2017	71205206	5	6	217	257	217	0	43.4	97.04535	138	0	42.833333	56.087135	6.6568156	0.1544831	7120520.6	20427315	64998661	32
10.42.0.211	10.42.0.211	47707	31.13.71.1	443	6	13/06/2017	71166704	4	6	219	257	219	0	54.75	109.5	138	0	42.833333	56.087135	6.6885211	0.1405151	7907411.6	21504867	65001606	16
10.42.0.211	10.42.0.211	50923	31.13.71.37	443	6	13/06/2017	65110831	9	9	1404	751	1078	0	156	353.98976	422	0	83.444444	134.51497	33.097412	0.2764517	3830048.9	15723280	64845229	29
10.42.0.211	10.42.0.211	50923	31.13.71.37	443	6	13/06/2017	436	3	0	42	0	42	0	14	24.248711	0	0	0	0	96330.275	6880.7339	218	155.56349	328	108
172.217.6.2	172.217.6.2	443	10.42.0.211	37041	6	13/06/2017	44	2	0	55	0	55	0	27.5	38.890873	0	0	0	0	1250000	45454.545	44	0	44	44
172.217.6.2	10.42.0.211	37041	172.217.6.2	443	6	13/06/2017	25340	4	0	61	0	38	0	15.25	18.643587	0	0	0	0	2407.2612	157.8532	8446.6667	13784.012	24358	143
10.42.0.211	10.42.0.211	58830	31.13.69.20	443	6	13/06/2017	66893737	42	98	1366	106481	330	0	32.52381	78.263856	1398	0	1086.5408	551.31205	1612.2137	2.0928716	481249.91	5512557.1	64999919	4
10.42.0.211	31.13.69.20	443	10.42.0.211	58830	6	13/06/2017	32138	1	1	0	0	0	0	0	0	0	0	0	0	0	62.231626	32138	0	32138	32138
10.42.0.211	10.42.0.211	34245	31.13.69.22	443	6	13/06/2017	65472885	21	23	2596	9938	528	0	123.61905	179.2402	1398	0	432.08696	513.09665	191.43803	0.6720339	1522625.2	9902875.3	64949507	13
10.42.0.211	10.42.0.211	34245	31.13.69.22	443	6	13/06/2017	183	3	0	0	0	0	0	0	0	0	0	0	0	0	16393.443	91.5	6.363961	96	87
216.58.219.	216.58.219.	80	10.42.0.211	47132	6	13/06/2017	239120	2	0	0	0	0	0	0	0	0	0	0	0	0	8.3640013	239120	0	239120	239120
172.217.6.2	172.217.6.2	443	10.42.0.211	41528	6	13/06/2017	246066	2	1	0	0	0	0	0	0	0	0	0	0	0	12.191851	123033	149070.84	228442	17624
8.0.6.4-8.6.	8.6.0.1	0	8.0.6.4	0	0	13/06/2017	96052881	3	0	0	0	0	0	0	0	0	0	0	0	0	0.0312328	48026441	1626727.3	50590924	45461957
10.42.0.211	10.42.0.211	36385	121.41.26.1	8090	6	13/06/2017	65666616	4	4	279	1200	279	0	69.75	139.5	1200	0	300	600	22.52286	0.1218275	9380945.1	24497833	64935794	957
169.47.5.24	10.42.0.211	42598	169.47.5.24	443	6	13/06/2017	116413035	53	45	1830	1295	307	0	34.52302	78.958754	283	0	28.777778	48.91546	26.844073	0.8418301	1200134.4	4573768.3	25563137	104
172.217.3.1	172.217.3.1	443	10.42.0.211	50495	6	13/06/2017	248207	2	1	0	0	0	0	0	0	0	0	0	0	0	12.086686	124103.5	172051.1	245762	2445
156.154.20	10.42.0.211	55703	156.154.20	443	6	13/06/2017	355853	10	10	891	4989	368	0	89.1	148.44112	1448	0	498.9	672.50038	16523.677	56.202983	18729.105	26412.614	98334	22
156.154.20	10.42.0.211	55703	156.154.20	443	6	13/06/2017	17239	2	0	0	0	0	0	0	0	0	0	0	0	0	116.01601	17239	0	17239	17239
10.42.0.211	10.42.0.211	40705	23.206.168.	443	6	13/06/2017	271354	8	10	829	5771	486	0	103.625	174.54262	1448	0	577.1	619.1547	24322.472	66.334014	15962	28733.549	113319	9
10.42.0.211	10.42.0.211	40705	23.206.168.	443	6	13/06/2017	28741	3	0	0	0	0	0	0	0	0	0	0	0	0	104.3805	14370.5	20190.02	28647	94
172.217.8.3	172.217.8.3	443	10.42.0.211	51400	6	13/06/2017	75986659	1	3	0	31	0	0	0	0	31	0	10.333333	17.897858	0.4079664	0.0526408	25328886	43740909	75836465	811
162.208.22.	10.42.0.211	39706	162.208.22.	443	6	13/06/2017	5325354	10	10	1252	5120	474	0	125.2	179.29851	1460	0	512	570.50757	1196.5402	3.7556189	280281.79	1134333.8	4963575	21
172.217.7.1	172.217.7.1	443	10.42.0.211	44593	6	13/06/2017	544353	1	2	0	0	0	0	0	0	0	0	0	0	0	5.5111297	272176.5	272129.34	464601	79752

A2:

Fwd IAT Mi	Fwd IAT Std	Fwd IAT Ma	Fwd IAT Mi	Bwd IAT Tol	Bwd IAT Mi	Bwd IAT Std	Bwd IAT Ma	Bwd IAT Mi	Fwd PSH Fla	Bwd PSH Fl	Fwd URG Fl	Bwd URG Fl	Fwd Header	Bwd Heade	Fwd Packet/	Bwd Packet/	Min Packet	Max Packet	Packet Leng	Packet Leng	Packet Leng	FIN Flag Cou	SYN Flag Cc	RST Flag Co	PSH Flag Cc
0	0	0	0	0	0	0	0	0	0	0	0	0	32	32	27.007319	27.007319	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	32	32	27.282896	27.282896	0	0	0	0	0	0	0	0	0
68762.857	62376.182	158676	449	487990	44362.727	86342.043	217790	8	0	0	0	0	180	244	14.978496	22.467745	0	1460	615.95238	672.74479	452585.55	0	0	0	1
4654.5	5137.1308	8287	1022	0	0	0	0	0	0	0	0	0	60	0	322.26877	0	0	0	0	0	0	0	0	0	0
2841499.4	7314093.2	19426714	639	410964	82192.8	154845.68	355945	7	0	0	0	0	180	124	0.4022021	0.3016516	0	1460	407.26667	626.95847	393076.92	0	0	0	1
4799065.8	7048340.7	15210827	28574	14998956	14998956	0	14998956	14998956	0	0	0	0	188	80	0.2604674	0.1041869	0	0	0	0	0	0	0	0	1
0	0	0	0	4278732	4278732	0	4278732	4278732	0	0	0	0	32	64	0.2323245	0.4646489	0	0	0	0	0	0	0	0	0
0	0	0	0	706	706	0	706	706	1	0	0	0	32	64	26.36716	52.734272	0	23	11.5	13.279056	176.33333	0	1	0	0
0	0	0	0	137	137	0	137	137	1	0	0	0	32	64	27.299282	54.598564	0	23	11.5	13.279056	176.33333	0	1	0	0
0	0	0	0	58	58	0	58	58	0	0	0	0	32	64	27.371014	54.742028	0	31	7.75	15.5	240.25	0	0	0	0
221194	0	221194	221194	0	0	0	0	0	0	0	0	0	64	0	0.0418366	0	0	0	0	0	0	0	0	0	0
914675	0	914675	914675	0	0	0	0	0	0	0	0	0	64	0	2.186569	0	0	0	0	0	0	0	0	0	0
3692864	0	3692864	3692864	0	0	0	0	0	0	0	0	0	64	0	0.5415851	0	0	0	0	0	0	0	0	0	0
14770330	0	14770330	14770330	0	0	0	0	0	0	0	0	0	64	0	0.1354066	0	0	0	0	0	0	0	0	0	0
232702.67	204487.24	406923	7569	65436079	21812026	37401979	64999517	22091	0	0	0	0	136	136	0.0608873	0.0608873	0	568	112.22222	224.90097	50580.444	0	0	0	1
313399	262591.05	481980	10844	65488823	21829608	37388693	65001447	1724	0	0	0	0	136	136	0.0607686	0.0607686	0	1200	164.33333	399.18542	159349	0	0	0	1
1551624.3	3057191.7	6137349	539	71175165	14235033	28504744	65001145	32	0	0	0	0	168	200	0.0702196	0.0842635	0	217	39.5	70.776858	5009.3636	0	0	0	1
2055017.7	3502944.9	6099870	32260	71136834	14227367	28508790	65003799	16	0	0	0	0	136	200	0.0562061	0.0843091	0	219	43.272727	73.552826	5410.0182	0	0	0	1
33194.5	43731.021	122500	937	65080769	8135096.1	22915701	64848404	29	0	0	0	0	296	296	0.1382259	0.1382259	0	1078	113.42105	256.52514	65805.146	0	0	0	1
218	155.56349	328	108	0	0	0	0	0	0	0	0	0	96	0	6880.7339	0	0	42	10.5	21	441	0	0	0	0
44	0	44	44	0	0	0	0	0	1	0	0	0	64	0	45454.545	0	0	55	36.666667	31.754265	1008.3333	0	1	0	0
8446.6667	13784.012	24358	143	0	0	0	0	0	0	0	0	0	128	0	157.8532	0	0	38	12.2	17.527122	307.2	0	0	0	0
46189.22	143128.66	749579	15	66858866	689266.66	6599122.6	65006472	4	0	0	0	0	1352	3144	0.6278615	1.4650101	0	1398	764.87234	670.74047	449892.78	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	32	32	31.115813	31.115813	0	0	0	0	0	0	0	0	0
26166.05	52347.777	235017	115	65437908	2974450.4	13853076	64997080	13	0	0	0	0	680	744	0.3207435	0.3512905	0	1398	278.53333	414.46294	171779.53	0	0	0	1
91.5	6.363961	96	87	0	0	0	0	0	0	0	0	0	96	0	16393.443	0	0	0	0	0	0	0	0	0	0
239120	0	239120	239120	0	0	0	0	0	0	0	0	0	64	0	8.3640013	0	0	0	0	0	0	0	0	0	0
246066	0	246066	246066	0	0	0	0	0	0	0	0	0	64	32	8.1279006	4.0639503	0	0	0	0	0	0	0	0	0
48026441	3626727.3	50590924	45461957	0	0	0	0	0	0	0	0	0	0	0	0.0312328	0	0	0	0	0	0	0	0	0	0
243607.33	212209.74	394489	957	65337860	21779287	37430636	65000029	1348	0	0	0	0	136	136	0.0609138	0.0609138	0	1200	164.33333	399.18542	159349	0	0	0	1
2237322.9	6110290.2	25563137	104	116341149	2644117	6569566.3	25675544	328	1	0	0	0	1696	1440	0.4552755	0.3865547	0	307	31.59596	66.325099	4390.0188	0	1	0	0
248207	0	248207	248207	0	0	0	0	0	0	0	0	0	64	32	8.0577905	4.0288952	0	0	0	0	0	0	0	0	0
35163	31115.948	98334	67	318067	35340.778	50095.157	140031	22	0	0	0	0	328	328	28.101491	28.101491	0	1448	280	509.44205	259531.2	0	0	0	1
17239	0	17239	17239	0	0	0	0	0	0	0	0	0	64	0	116.01601	0	0	0	0	0	0	0	0	0	0
32652.143	38070.897	113319	238	247812	27534.667	53253.518	158379	9	0	0	0	0	264	328	29.481784	36.85223	0	1448	347.36842	515.7059	265952.58	0	0	0	1
14370.5	20190.02	28647	94	0	0	0	0	0	0	0	0	0	96	0	104.3805	0	0	0	0	0	0	0	0	0	0
0	0	0	0	75837276	37918638	53623905	75836465	811	0	0	0	0	32	96	0.0131602	0.0394806	0	31	6.2	13.863621	192.2	0	0	0	0
40191.778	31320.512	85366	50	5283891	587099	1655340	4999982	21	0	0	0	0	220	212	1.8778094	1.8778094	0	1460	303.42857	450.74301	203169.26	0	0	0	1
0	0	0	0	464601	464601	0	464601	464601	0	0	0	0	32	64	1.8370432	3.6740865	0	0	0	0	0	0	0	0	0

Appendix B

Appendix

B.1 Snapshot of Dataset

A3:

Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	1016299	1016299	14995041	0	14995041	14995041	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
2283.9549	36189	32959	35568005	41621233	64998661	6137349	Android_Adware
468.8118	32923	32260	35550738	41649817	65001606	6099870	Android_Adware
0	265556	265556	64845229	0	64845229	64845229	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	1893758	1893758	64999919	0	64999919	64999919	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	523321	523321	64949507	0	64949507	64949507	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	48026441	3626727.3	50590924	45461957	Android_Adware
0	0	0	0	0	0	0	Android_Adware
1825736.6	4241128	127694	20282884	4062798.5	25563137	16215824	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	149383	149383	75836465	0	75836465	75836465	Android_Adware
0	0	0	0	0	0	0	Android_Adware
0	0	0	0	0	0	0	Android_Adware

201-16-497

ORIGINALITY REPORT

18%	14%	13%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.daffodilvarsity.edu.bd:8080 Internet Source	1%
2	pureadmin.qub.ac.uk Internet Source	1%
3	www.researchgate.net Internet Source	1%
4	www.hindawi.com Internet Source	<1%
5	"Innovative Data Communication Technologies and Application", Springer Science and Business Media LLC, 2022 Publication	<1%
6	ebin.pub Internet Source	<1%
7	Shamsher Ullah, Jianqiang Li, Farhan Ullah, Jie Chen, Ikram Ali, Salabat Khan, Abdul Ahad, Victor C.M. Leung. "The revolution and vision of explainable AI for android malware detection and protection", Internet of Things, 2024 Publication	<1%