Spring 2023

Explainable AI for Android Malware Detection

Maithili Kulkarni
San Jose State University

Explainable AI for Android Malware Detection

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Maithili Kulkarni

May 2023

The Designated Project Committee Approves the Project Titled

Explainable AI for Android Malware Detection

by

Maithili Kulkarni

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Mark Stamp	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science
Dr. Genya Ishigaki	Department of Computer Science

ABSTRACT

Explainable AI for Android Malware Detection

by Maithili Kulkarni

Android malware detection based on machine learning (ML) is widely used by the mobile device security community. Machine learning models offer benefits in terms of detection accuracy and efficiency, but it is often difficult to understand how such models make decisions. As a result, popular malware detection strategies remain black box models, which may result in a lack of accountability and trust in the decisions made. The field of explainable artificial intelligence (XAI) attempts to shed light on such black box models. In this research, we apply XAI techniques to ML-based Android malware detection systems. We train classic ML models (Support Vector Machines, Random Forest, and k-Nearest Neighbors) and deep learning (DL) models (Multi-Layer Perceptron and Convolutional Neural Networks) on a challenging Android malware dataset. We then apply state-of-the-art XAI techniques, including Local Interpretable Model-agnostic Explanations (LIME), Shapley Additive exPlanations (SHAP), Eli5, PDP plots, and Class Activation Mapping (CAM). We obtain global and local explanation results and we discuss the utility of XAI techniques in this problem domain. We also provide an extensive literature review of recent XAI work related to deep learning methods for Android malware, and we discuss XAI research trends, challenges, and consider future research directions.

ACKNOWLEDGMENTS

I want to express my gratitude to my project advisor, Dr. Mark Stamp, for his guidance, support, and encouragement throughout my graduate studies. He has always been patient in listening to the tiniest issues and roadblocks that came up while working on the project.

I would also like to thank my committee member Dr. Thomas Austin for his time and guidance.

I am further thankful to my committee member, Dr. Genya Ishigaki, for his valuable inputs.

Finally, I would like to thank my parents and friends for their unwavering support and guidance throughout my Master's degree program.

TABLE OF CONTENTS

CHAPTER

1 Introduction 1

2 Background 4

2.1 Malware and Categories 4

2.1.1 Ransomware 5

2.1.2 Phishing Attacks 5

2.1.3 Botnets 5

2.1.4 Spam 6

2.2 Malware Detection Methods 6

2.2.1 Classic Machine Learning Based Malware Detection 6

2.2.2 Deep Learning Based Malware Detection 7

2.3 XAI: Motivation and Workflow 10

2.4 Taxonomy of Machine Learning Interpretability Techniques . . . 12

2.4.1 Ante-hoc explanations VS Post-hoc Explanations 12

2.4.2 Model-Agnostic VS Model-Specific Explanations 13

2.4.3 Global VS Local Interpretability Explanations 13

2.5 Scale of Machine Learning Interpretability Techniques 14

2.5.1 High interpretability Techniques 14

2.5.2 Medium interpretability Techniques 14

2.5.3 Low interpretability Techniques 15

2.6 XAI Techniques 15

2.6.1 Support Vector Machines (SVMs) 15
2.6.2 Local Interpretable Model-agnostic Explanations (LIME) . 16
2.6.3 SHapley Additive exPlanations (SHAP) 17
2.6.4 ELI5 19
2.6.5 Partial Dependence Plot (PDP) 19
2.6.6 Gradient-weighted Class Activation Map (Grad-CAM) . . 20

3 Related Work 22

4 Experiments and Results 29

4.1 Dataset 29
4.2 Setup 30
4.3 Preprocessing 30
4.4 Evaluation metrics 32
4.5 Implementation 32
4.6 Results and Discussion: Performance of ML/DL models 34
4.7 Results and Discussion: XAI Results 35
4.7.1 Feature Importance and Eli5 36
4.7.2 LIME Interpretations 37
4.7.3 SHAP Interpretations and PDPs 41
4.7.4 Grad-CAM 48
4.8 Comparative study of XAI techniques 49

5 Conclusion and Future Work 52

LIST OF REFERENCES 55

APPENDIX

A LIME notations 59

B LIME Explanations on Other Test Instances 60

C SHAP Explanations on Other Test Instances 61

CHAPTER 1

Introduction

Malware, short for malicious software, is a mechanism used by software disruptors worldwide [1]. They appear in various kinds as worms, viruses, adware, ransomware, etc. Most of these are actual executable software that slides into a victim's computer system and causes some form of damage. In recent years, there is an increase in the use of smartphones over laptop computers and smartphones remain the preferred devices for consumers. This increasing popularity of smartphones means these devices are susceptible to malware attacks.

With the increasing use of machine learning in every software product, it is no surprise that machine learning is the most popular approach used by Intrusion Detection Systems for detecting malware, including mobile devices [2]. Computer scientists have long been studying malware and have been researching techniques that can be used for early detection of the presence of malware. They have managed to extract various features, including byte features, opcodes, and other properties from live malware by sand-boxing it and have built complex machine learning models around it [3]. We elaborate on some of these techniques in Chapter 2.

Although ML provides significant capabilities in the areas of malware detection and related research, such techniques are generally treated as black boxes [4]. This black box aspect of ML can limit the trust that users are willing to place in such models. From a security perspective, black box models are more susceptible to adversarial attacks, where an attacker attempts to modify a model to yield incorrect results, e.g., a specific malware sample might be classified as benign. Moreover, these high-performing ML models fail to protect against the newest threats. It is hard to identify where the model is failing. Thus, it has been popularly theorized that these systems need to be further analyzed to be able to offer explanations behind such

malware detection. The proposed project banks on this idea.

The emerging field of explainable artificial intelligence (XAI) deals with understanding the inner workings of ML models in general, and neural networking models in particular [5]. Explainable AI (XAI) is the hottest research topic which helps to derive information on the model's outcome and also visualize those results providing optimum transparency. XAI explains the model's outcome by quantifying the influence of each input variable or by using approximation or surrogate models. It gives a transparent and interpretable view of the model's decisions which helps tackle its exploitable weak points and understand where the model is most uncertain. In this research, we will focus on XAI in the context of Android malware detection.

Our proposed research will consider XAI for a variety of classic ML techniques and deep learning (DL) models that have been trained on the KronoDroid - Android malware dataset. Specifically, for classic ML models, we train Support Vector Machine (SVM), k-Nearest Neighbor (k-NN), Random Forest (RF). We also train deep learning models Multi-Layer Perceptron (MLP) and Convolutional Neural Networks (CNNs). In general, classic ML techniques are reasonably interpretable, as such models typically have natural probabilistic, algebraic, or geometric interpretations. In contrast, most neural networking models are opaque, in the sense that it is non-trivial to obtain an understanding of how they are making decisions. This research aims to provide a comparative study on XAI for classic ML vs deep learning methods for Android malware detection.

For each trained model, we apply relevant XAI techniques from among the following: Local Interpretable Model-Agnostic Explanations (LIME), SHapley Additive exPlanations (SHAP), PDP, and Eli5 [6, 7, 8]. We will consider the trade-offs between explainability and the accuracy of models, with an emphasis on the distinction between classic ML and DL techniques. Additionally, the research gives a review

of recent literature work that focused on using XAI for deep learning methods in Android malware. This latter objective can be viewed as an extension of the previous research [9]. Our literature review covers recent XAI work on deep learning methods in the Android malware domain along with a discussion on XAI trends, research focuses, and challenges in DL-based Android malware defenses.

The remainder of this paper is organized as follows. Chapter 2 includes a range of relevant background topics on malware detection strategies and also discusses explainable AI techniques that we employed in our experiments, in detail. Chapter 3 gives an overview of related previous work on malware classification and provides a literature review on recent XAI work on deep learning methods for Android malware. Chapter 4 covers the implementation of classic and deep learning models used for malware detection in this research, as well as our experiments and results. Finally, Chapter 5 concludes with a discussion of a few potential avenues for future work.

CHAPTER 2

Background

We begin this chapter by giving a brief overview of malware, followed by a discussion on existing popular machine-learning algorithms to detect and classify malware. We broadly categorize them as classic machine-learning algorithms and deep-learning algorithms. This chapter ends by providing detailed background on the popular state-of-the-art XAI techniques.

2.1 Malware and Categories

The devices we rely on, such as our smartphones and computers, face ongoing challenges from a range of threats such as phishing attacks, spam, ransomware, botnets, and other malicious viruses [10]. Among these threats, malware poses the greatest security risk, as it can be created for reasons ranging from harmless pranks to serious crimes like organized crime, warfare, or espionage. Recent data depicts a significant surge in the number of android malware attacks over the past few years, as illustrated in the Figure 1.

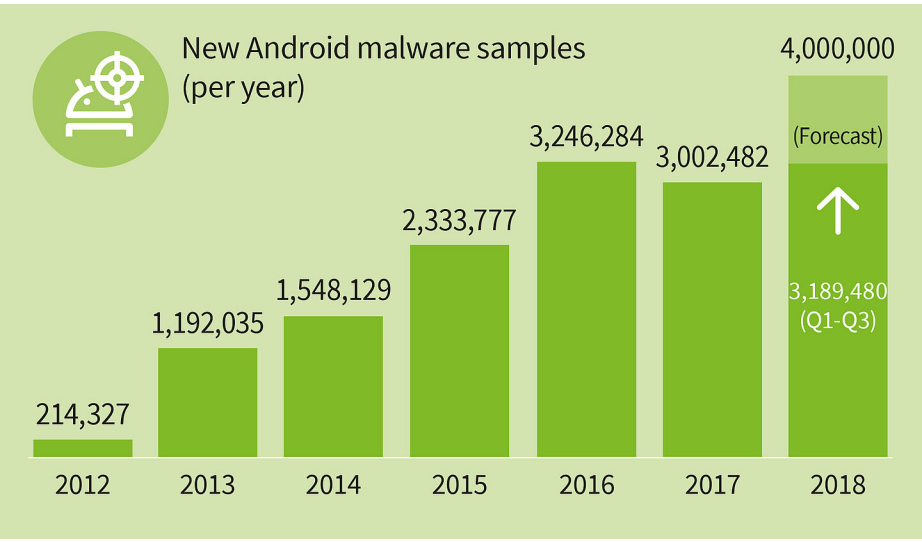

Figure 1: Total count of mobile malware rises [11]

Malware is not just a single threat, but covers a broad category of threats such as

phishing attacks, spam, ransomware, and botnets. We give a brief overview of these threats in the following sections before we discuss XAI in detail.

2.1.1 Ransomware

Ransomware is a case where an attacker takes the data of a victim and holds it hostage [10]. The ransomware program encrypts the user's data and demands a ransom in order for the victim to gain the decryption key to regain access to their data. There is typically no way to access the data once the ransomware encrypts it which can be devastating for major companies and other users.

2.1.2 Phishing Attacks

Phishing attacks are attacks that target a victim using a variety of techniques in order to steal sensitive information such as bank information or credit card numbers [10]. Phishing attacks can take the form of emails or messages. Additionally, attackers typically utilize social engineering tactics such as a sense of urgency or fear in order to trick victims into giving away personal and sensitive information. This attack primarily relies on human error as it utilizes deception as a means to steal any information [12].

2.1.3 Botnets

A botnet is a network of computers that have already been infected with malware by an attacker, which in this case would be called a botmaster. The botmaster would most likely have infected the computers with malware so that they have remote control over the victims' computers without their knowledge. Once a computer becomes a part of the botnet, they essentially become a zombie and follow the commands of the botmaster. Botnets can become a significant threat as they can be utilized to initiate additional malicious attacks [13].

2.1.4 Spam

Spam is unwanted messages that are typically sent through email, but can be sent through other forms of messaging as well [14]. A majority of spam is an advertisement, but can also potentially contain links to fraudulent or malicious websites. There are various ways a spammer can collect email addresses to target: website scraping, purchasing lists, etc. Once they have a number of email addresses, spammers are able to send a large quantity of emails at once using automation.

2.2 Malware Detection Methods

To avoid detection by antivirus tools, malware writers are consistently enhancing their malware. Meanwhile, security software companies are continuously researching ways to enhance their malware detection methods. In the upcoming sections, we will explain the strategies employed in our research to identify Android malware.

2.2.1 Classic Machine Learning Based Malware Detection

Malware detection and classification often utilize machine learning techniques, including Support Vector Machine (SVM), Random Forest (RF), k-NN, and Multilayer Perceptron (MLP). The two primary categories of malware detection are static malware detection and dynamic malware detection.

2.2.1.1 Support Vector Machines (SVM)

Support Vector Machines (SVMs) are a type of supervised machine learning model utilized for classification tasks that can detect and identify common patterns. SVMs employ the kernel function to transform the training data into a high-dimensional space, making the problem linearly separable. Once the model has been trained, it can classify new data into one of the designated categories with a high degree of accuracy [2].

2.2.1.2 Random Forest (RF)

Random Forest is a machine-learning approach utilized for classification and regression [15]. It is an ensemble of decision trees, where random features are considered at each node to split each tree. This random selection of data and features helps to reduce overfitting and increase the model's accuracy. Ultimately, the model selects the most commonly chosen class among all trees. While a greater number of trees can lead to better accuracy, random Forest can also overfit on data [2].

2.2.1.3 k- Nearest Neighbor (k-NN)

The k-Nearest Neighbors (k-NN) algorithm is a supervised machine learning technique where samples are classified based on the k nearest samples in the training set. When classifying an input point X, k-NN identifies the k closest training data points to X and assigns the class label based on the majority class of these neighbors. One of the benefits of k-NN is that it does not require explicit training, making it one of the simplest algorithms to use [2]. k-NN is an intuitive algorithm that can be useful for small datasets with well-defined structures but can be computationally expensive for large datasets. Hence, it is important to carefully choose the value of k to avoid overfitting and to consider the computational costs of calculating the distances from each data point.

2.2.2 Deep Learning Based Malware Detection

Deep learning techniques use neural networks to solve complex problems in machine learning. These networks can have a large number of layers and parameters, making them computationally demanding to train. To decrease the training time, deep learning techniques often utilize graphics processing units (GPUs). Various deep learning models such as AutoEncoders (AEs) [16], Long Short-Term Memory (LSTM), Feedforward Neural Networks, and Convoluted Neural Networks (CNN)

are being used in malware security applications. These models provide the basis of deep learning and set it apart from traditional machine learning models due to their complex architecture and their usage in various high-level research.

2.2.2.1 Multi-layer Perceptron (MLP)

Artificial Neural Networks (ANNs) are mathematical models that are designed to simulate the way the brain operates. Multilayer Perceptrons (MLPs) are a type of Artificial Neural Network (ANN) and the simplest useful neural network architecture. MLPs are feed-forward networks that generalize basic perceptrons to allow for nonlinear decision boundaries. They are sometimes also referred to as ANNs.

MLPs consist of multiple layers of perceptrons, including one or more hidden layers. Figure 2 shows the architecture of MLP. It has two hidden layers. Each edge in an MLP architecture represents a weight or a parameter, that is calculated during the training phase of the network. MLPs are trained by making modifications to the weights such that it causes large error reduction resulting in speed convergence. This technique is called backpropagation [2].

2.2.2.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are specifically designed for image analysis and are composed of several layers, including convolutional and pooling layers, that allow them to identify and extract local features from images.

Figure 3 shows the basic architecture of CNN. In a CNN architecture [2], the first layer is typically a convolutional layer, which is responsible for extracting features from the input images. Each filter (or kernel) in the convolutional layer is applied by sliding it over the whole input image. The output that we obtained from this layer is a feature map. This output is fed to the second layer of CNN. In the next layers, filters are applied over the entire output of the first layer. The pooling layer

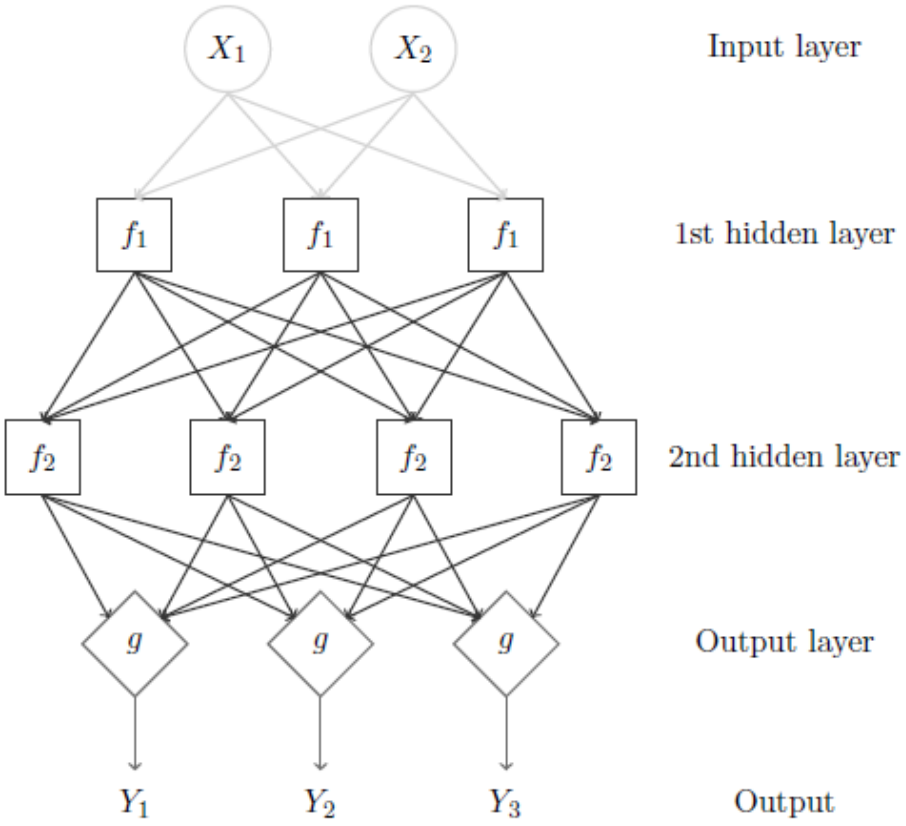

Figure 2: MLP with two hidden layers [2]

helps to downsample the output of the convolutional layer and reduce the number of parameters in the model. The three types of pooling operations are commonly used in CNNs, with max pooling being the most popular. In max pooling, the maximum value within each pooling window is selected, while in min pooling, the minimum value is selected. Average pooling calculates the average of all the values in each pooling window. The choice of pooling operation depends on the specific problem and the characteristics of the data. This convolution of convolution can be performed over and over again. At each layer, we obtain a feature map that represents more and more information about the input image. Lastly comes the fully connected layer that connects the neurons between two different layers. The classification process begins

at this stage. Dropout is a regularization technique to deal with overfitting in neural networks, by dropping out neurons, the network learns more robust and generalizable features, resulting in better performance on the test data. A Rectified Linear Unit (`ReLU`), `sigmoid`, `softmax`, and the `tanh` function are the common activation functions used in CNN.

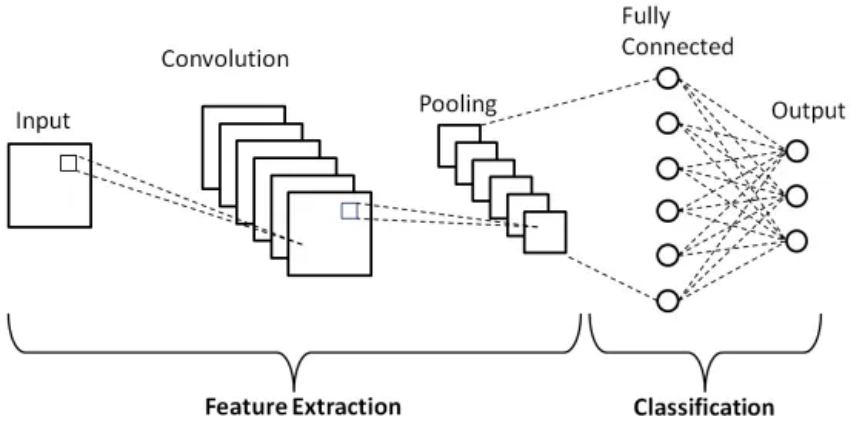

Figure 3: CNN architecture [17]

2.3 XAI: Motivation and Workflow

The applications of artificial intelligence in the security domain induce many challenges. The cybercriminals who train malware to become ML immune may try to manipulate the Android malware apps such that the ML-based Android malware detection framework fails to identify any anomalous behavior of the system, also known as evasion attacks. Also, these ML-based security systems can be compromised unethically exposing them to adversarial attacks. The statistical model has a tendency to overfit on the training data and prevents its capacity to analyze accurately in cases of new or unknown malware samples. These can be dealt with only by developing an understanding of the how model works and its reasoning. Additionally, DL systems require heavy computational power, memory, and data. By using XAI techniques, we can perform feature reduction based on feature importance. Furthermore, deep

neural network models (DNN) are vastly popular but lack interpretability. The use of XAI on DNN paves the way for researchers to understand and rely on the results generated by complex DNN systems and makes users adapt them into real-world applications. Figure 4 details the challenges in confidently adopting existing malware detection systems that drive the need of applying XAI in ML/DL-based Android malware systems.

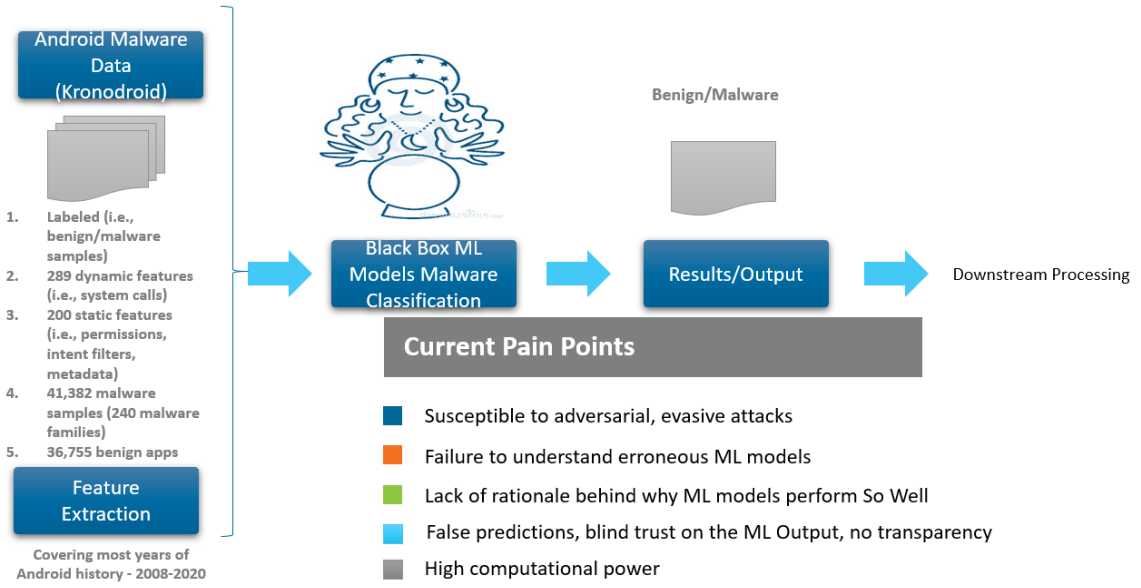

Figure 4: Current process: ML-based malware detection

Explainable Artificial Intelligence (XAI) is becoming increasingly important in the field of machine learning as it helps in making complex models more transparent and interpretable. This, in turn, can help build trust in the model's decisions and outputs, particularly in sensitive domains such as information security. Figure 5 shows how Explainable Artificial Intelligence (XAI) bridges the gap between the black box model and the output produced by them. After training the ML model on the input dataset, the output of the ML system, whether it is a prediction or classification, is fed into the XAI system. The XAI system interprets the model, identifying the features that

have the most significant impact on the model's possible decision outcome, providing users with greater confidence in the outcomes produced by these models.

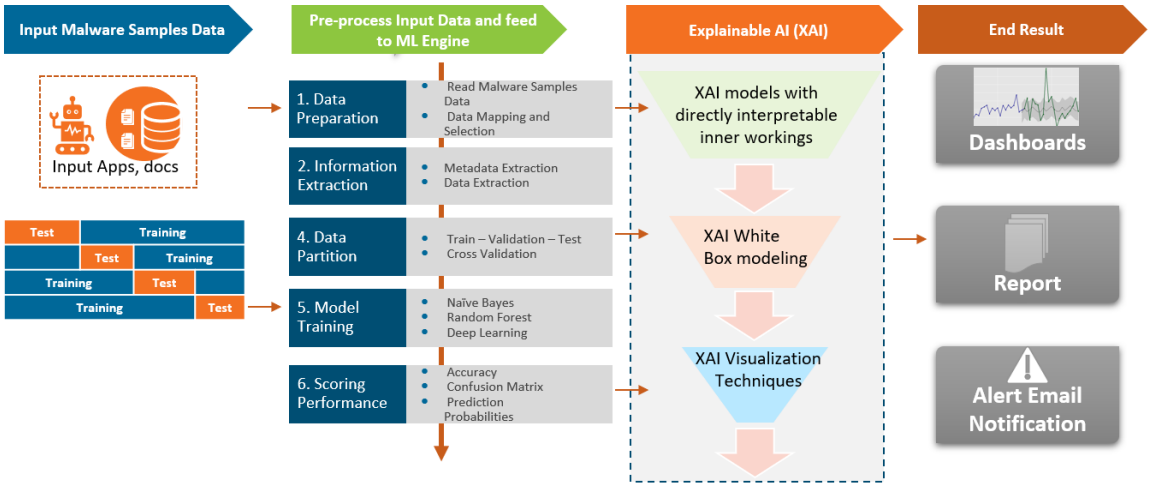

Figure 5: Deep dive: Explanation workflow in cybersecurity

2.4 Taxonomy of Machine Learning Interpretability Techniques

XAI techniques in cybersecurity are based on different criteria such as structural characteristics of models, granularity level at which explanation is provided, and their application scope. We analyze each category of explanations in our experiments. Below we discuss these categories briefly.

2.4.1 Ante-hoc explanations VS Post-hoc Explanations

Depending on whether explanations are derived by virtue of the intrinsic explainable nature of the model or need to be applied after an ML model is trained makes an explanation ante hoc or post hoc.

ML Models that are simple in structure (intrinsically interpretable models) can be termed as ante hoc interpretable models. For example, linear models fall into the ante hoc interpretable models category. Post hoc model interpretability means applying explicit interpretation methods after the model is trained. Post hoc techniques can also be used on models that inherently provide interpretable results after they are

trained [7]. For example, the feature importance method `feature_importances_` of `GridSearchCV` tuning technique can be considered as a post hoc interpretable technique.

2.4.2 Model-Agnostic VS Model-Specific Explanations

Some of these XAI techniques are model agnostic, in the sense that they can be applied to any machine learning model, regardless of its underlying architecture or training algorithm, while others are model-specific. For example, LIME, SHAP, PDP plots, and Eli5 are model-agnostic interpretable techniques and can be applied to almost any machine learning predictions while CAM is specific to CNN models.

Intrinsic interpretable models have model-specific interpretations, while model-agnostic methods generate explanations post hoc, i.e. after the ML model has been trained. Model-specific techniques are simple and directly use the model on which predictions are to be made, which is why they tend to be more accurate while model-agnostic methods are convenient but usually use approximations to generate the explanations, as a result, accuracy might be lower compared to model-specific explanation techniques [5].

2.4.3 Global VS Local Interpretability Explanations

This criterion deals with whether some explanation methods explain an individual prediction and some provide explanations for the group of predictions i.e. they explain the entire model behavior. Local interpretability techniques provide insight into how and why a model made a specific prediction for a single instance. Also, local interpretable methods can be used to understand the certain prediction of a model for a group of instances [18]. Locally, the prediction may exhibit a linear or monotonic dependence on certain features, while displaying a simpler or less complex relationship with others. Global techniques, in contrast to local techniques, focus on interpreting

the model as a whole by considering all features and instances in a holistic manner. For example, LIME proposes to focus on interpreting locally instead of providing a global model interpretation while SHAP can be used for both local and global explanations.

2.5 Scale of Machine Learning Interpretability Techniques

More the complex model, the more it has become increasingly opaque. It has been noted that mostly the high-performing models like deep learning models are the least explainable, and the least accurate models like decision trees are the most explainable [19].

2.5.1 High interpretability Techniques

Models that use linear and monotonic functions are considered highly interpretable because their response to changes in input can be easily predicted. In a linear function, the output changes at a fixed rate when the input variable changes. Monotonic functions, on the other hand, always either increase or decrease as their inputs change. Classic regression algorithms are examples of models that truly belong to this highly interpretable class. For example, SVM linear model is a highly interpretable model. Other explanatory techniques such as LIME use linear and monotonic functions as approximations.

2.5.2 Medium interpretability Techniques

Models with nonlinear and monotonic functions are considered moderately interpretable. Nonlinear functions do not have a fixed rate of change in output for a given change in input and use a combination of the model parameters and have dependencies on one or more independent variables. Thus, they can be bit more complex to interpret. For example, the SVM RBF model is a fairly interpretable model.

2.5.3 Low interpretability Techniques

Machine learning models with nonlinear and non-monotonic functions fall into the low interpretability category of explainable AI techniques. Most of the complex ML models like ensemble models and deep learning models such as MLP and CNN are often difficult to interpret and provide low interpretation compared to other machine learning models. For example, CAM is a low interpretability technique.

2.6 XAI Techniques

Machine learning models have a tendency to pick up biases from the training data which makes them erroneous due to bias. Interpretability helps end users in detecting bias in machine learning models. The following sections describe existing state-of-the-art explainable techniques.

To enable fairness, accountability, and transparency as well with the intention of understanding model decision-making better and to increase the trust and confidence end users can place in using malware detection models in the practical scenario, we use both popular traditional interpretable models as well as advanced state-of-the-art XAI techniques. Before moving on to detailing our experiments, it is important to shed some light on how standard explainable AI techniques like SVM interpretations, LIME, SHAP, Eli5, PDP, and CAM work.

2.6.1 Support Vector Machines (SVMs)

Linear SVMs are interpretable models as they help determine the most contributing features for the SVM classifier. In `sklearn` Python library, it is easy to obtain feature weights for linear kernel using `coef_` method [20]. It assigns and provides a ranking of weights assigned to the features when the kernel is linear. Non-linear SVM is not highly interpretable. Thus, it is not easy to assess how the independent variables affect the target variable.

2.6.2 Local Interpretable Model-agnostic Explanations (LIME)

Local Interpretable Model-Agnostic Explanations (LIME) is an approach that utilizes local surrogate models to provide explanations for individual predictions made by complex machine learning models [21]. LIME achieves this by training surrogate models to approximate the predictions of the complex model. To use LIME, the trained machine learning or deep learning model, along with the predictions made by it, are required as inputs. The purpose of LIME is to provide an understanding of why a certain prediction was made by the machine learning model. LIME achieves this by monitoring the variations in predictions when the input data is perturbed and creating a new set of perturbed samples. Using these samples, LIME trains a surrogate model that approximates the predictions of the black box model, measured by how close the perturbed instances are to the local sample pertinent to the study. While the learned surrogate model provides a good approximation of the machine learning model's predictions locally, it may not be a good global approximation, as the technique is based on local fidelity.

The mathematical expression for LIME is [5]

$$\text{explanation}(x) = \text{argmin}_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

Appendix A can be referred to for more information on LIME mathematical model notations.

The above equation represents the explanation model, g that minimizes loss, L for the input sample, x. The loss function measures how close the explanation generated by LIME is to the prediction of the original model f, while the model complexity $\Omega(g)$ is kept low. G is the set of all possible explanations. π_x are the proximity measures that define how large the neighborhood around instance x is.

LIME Implementation Steps:

1. Choose an instance of a dataset, of which local explanation you are interested in.

2. Build your ML/DL model on the entire training dataset and generate the model outcome i.e. prediction or classification results on the test data.

3. Weigh in the perturbed instances according to their nearness to the instance under consideration.

4. Train a surrogate, interpretable model on the dataset by providing input variations.

5. Generate local explanations by using the interpretable model trained in the earlier step.

2.6.3 SHapley Additive exPlanations (SHAP)

SHapley Additive exPlanations (SHAP) is a widely used technique for Explainable AI, alongside other methods such as LIME and Generalised Additive Models (GAM). SHAP adopts the linear additive feature attribution explanation approach to compute the contribution of each feature to a prediction and generate an explanation for a given instance x [22]. Shapley values are determined based on whether the values of the features are present or not. The larger the Shapley value, the greater the contribution the feature makes to predicting the model outcome. For each input instance and feature pair, a Shapley value is calculated, and the model is explained by analyzing these values.

2.6.3.1 Shapley Value

Shapley, L. came up with a concept of Shapley value [5] to calculate the marginal contributions of each player in a coalition game by computing the average contribution of each player over all possible coalitions that include that player. This concept has

been extended to explain the contribution of each feature in a machine learning model prediction using the SHapley Additive exPlanations (SHAP) technique.

Consider Aaron (A), Brandon (B), and Clara (C) together working on finishing a piece of work of 28 units and the task is to find out: "What is everyone's contribution in work units to finish the given piece of work?"

To answer this question, refer to Table 1 which lists all the permutations of the work distribution data. To calculate the contribution of each player to the overall work, we consider every possible ordering of the players and calculate the marginal contribution of each player for that ordering. The marginal contribution of a player is the difference in the total work completed with and without that player's participation in the current ordering. For example, when Aaron is first, then Brandon, and then Clara, the total work completed is $2 + 22 + 4 = 28$ units, so Aaron's marginal contribution is 2 units. Similarly, Brandon's marginal contribution is 22 units, and Clara's is 4 units. We calculate the Shapley value for each player as the average of their marginal contributions across all possible orderings.

Table 1: Permutations of marginal contribution of work data

Combination	Aaron	Brandon	Clara	Total in work units
A, B, C	2	22	4	**28**
A, C, B	4	24	0	**28**
B, A, C	2	22	4	**28**
B, C, A	0	18	10	**28**
C, A, B	2	26	0	**28**
C, B, A	0	18	10	**28**
Average	0	22	4	**28**

2.6.3.2 SHAP Strengths

Lundberg and Lee [23] came up with SHAP method that can be used to describe the output of any machine learning model. SHAP offers several advantages when it comes to explaining the output of any machine learning model. Firstly, it provides

global explanations by identifying correlations, whether positive or negative, between one or more features and the target variable. Secondly, it also offers local explanations, similar to those provided by LIME, by providing SHAP values for each individual input instance, which helps to explain why a particular output was predicted and the role played by each feature in deciding the model outcome. Lastly, the SHAP method is model-agnostic, making it suitable for nearly any type of model, including tree-based models, deep learning models, linear regression, and logistic regression models.

2.6.4 ELI5

ELI5 (Explain Like I'm 5) is a Python package that helps understand how a machine learning model works by explaining its internal decision-making process [24]. It provides explanations for machine learning models by visualizing and breaking down their decision-making process. It allows users to understand the predictions of a model by providing feature importance scores and highlighting the most influential factors. ELI5 can help to interpret the predictions made by a model and to identify which features of the input data are the most important for the model's decision.

ELI5 provides support for black-box models, such as neural networks, by using a technique called perturbation. This involves randomly perturbing the input features and observing how the model's predictions change. By repeating this process for multiple input instances and averaging the results, ELI5 can estimate the importance of each feature even for models where feature importances are not directly available.

2.6.5 Partial Dependence Plot (PDP)

PDP stands for Partial Dependence Plots. It is a way to visualize the relationship between a set of input features and the output of a machine-learning model [5]. The idea behind PDP is to hold all features of the model constant except for one, and to vary that one feature over its entire range while measuring the output of the model.

The output of the model for each value of the single varying feature is then averaged over all possible combinations of the other input features. The result is a partial dependence plot, the directions for the values plotted on a chart tell us the direction in which the input features affect the outcome.

PDPs can help us to understand how the model is using each feature to make predictions and to identify any non-linear relationships between features and the output of the model. This information can be useful for feature selection, model tuning, and explaining the behavior of the model to stakeholders.

2.6.6 Gradient-weighted Class Activation Map (Grad-CAM)

The technique Grad-CAM is helpful when the input dataset consists of images belonging to several classes. It helps to comprehend the specific location within an image where a convolutional layer searches for a particular classification [5]. It is a class-discriminative localization technique for CNN interpretability. Grad-CAM uses the gradient information to obtain a heatmap of the important regions of an image that contributed to the final prediction. Grad-CAM heatmap is a weighted combination of feature maps which is often followed by ReLU. This heatmap can be superimposed onto the original image to visually identify the regions that were most important for the prediction.

One advantage of Grad-CAM over other interpretation methods is that it does not require any modifications to the original CNN model, and can be applied to any CNN model without retraining or fine-tuning. Additionally, Grad-CAM can be used to visualize the important regions of an image for any target class, not just the predicted class, making it a versatile tool for interpreting CNN models. Figure 6 shows the typical output generated by Grad-CAM on malware classification families.

The `visualize_cam` function generates a Grad-CAM that maximizes the layer

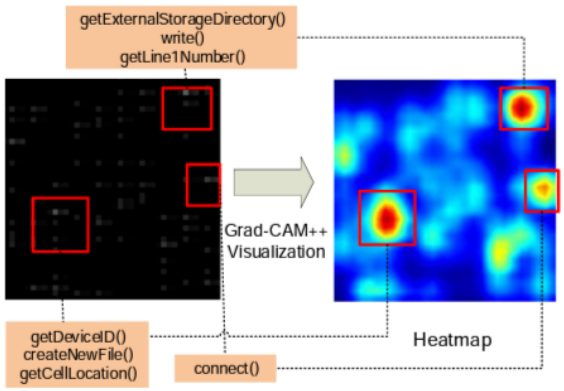

Figure 6: Grad-CAM sample output [25]

activations for a given input, for a specified output class.

CHAPTER 3

Related Work

The field of XAI is constantly evolving, and researchers have explored its application in the domain of malware detection and classification. This chapter will review previous attempts at applying XAI to malware classification and detection, listed in chronological order of publication, starting from the most recent ones.

Undoubtedly, mobile security applications are critical and explanations play a crucial role in enabling users to comprehend, trust, and proficiently operate machine learning-based mobile security systems. Machine learning (ML) models like support vector machine (SVM), neural networks, random forests, and graph models give high performance in the real world but are opaque in terms of explainability. There is a trade-off between the high prediction accuracy that ML models achieve and their non-black-box-ness i.e. their explainability. It has been noted that mostly the high-performing models like deep learning models are the least explainable, and the least accurate models like decision trees are the most explainable [26]. As a result, many attempts have been made to make ML models transparent by building explainable AI systems around them to create effective and human-understandable AI systems.

Manthena et al. [27] used a performance metrics dataset to conduct research on online malware detection. They studied the impact of malware on the behavior of virtual machines (VMs) in a cloud environment using a cloud environment. The research used only one interpretation technique SHAP extensively on various ML and DL models. The study applied three variants of SHAP explainability techniques, namely KernalSHAP, TreeSHAP, and DeepSHAP, to analyze the outcomes of various machine learning models including SVM Linear, SVMRBF (Radial Basis Function), Random Forest (RF), Feed-Forward Neural Net (FFNN), and Convolutional Neural Network (CNN) models. The models were trained on an online malware dataset to

understand their decision-making process and provide explanations for their predictions. The researchers utilized the SHAP interpretations to perform feature reduction by analyzing the global contribution of the CNN model. The SHAP technique helped the researchers obtain local as well as global explanations for the system under consideration. The study had a limited number of malware samples from various families, and the results are not representative of other platforms such as different operating systems (OS), including Android OS, iOS, or Windows OS.

Yan et al. [7] provided a comprehensive overview of explainable machine learning (ML) in the field of cybersecurity, with a particular focus on ante hoc and post hoc explanations. The authors discussed various techniques and tools that can be used to provide ante hoc and post hoc explanations, highlighting their advantages and limitations in the context of cybersecurity. Specifically, in the ante hoc explanation, the research outlines the linear model, tree-based model, and parametric model and their relevance, and usability according to security applications. While for the post hoc explanation, they considered techniques such as LIME, CAM, SHAP, and Local Explanation Method using Nonlinear Approximation (LEMNA), and Layerwise Relevance Propagation (LRP) and reported six metrics to evaluate them namely accuracy, sparsity, completeness, stability, efficiency, and fidelity. The research concludes by mentioning LRP as the most efficient XAI technique along with listing down open issues to be considered for the future, for example, the tradeoff between accuracy and explanation is one of the challenges that need to be dealt with while using XAI in cybersecurity ML methods.

Charmet et al. [28] reviewed XAI techniques from the point of their applications to cybersecurity, and by evaluating them using the security properties of XAI. The research provides a comparative study of XAI for different cybersecurity tasks and offers guidance on the most effective explanation techniques for achieving transparency

and trust. It also highlights XAI techniques that are best suited for explaining errors in the models and those that can improve the performance of the classifier. Lastly, the research compares works that aim to improve the security of XAI by focusing on properties such as fairness, integrity, privacy, and confidentiality. The noteworthy finding of the research is that XAI methods such as heatmaps and saliency maps can be easily compromised.

Ullah et al. conducted explainable AI experiments to explain and validate their proposed ML technique for malware detection using both text and greyscale image data [29]. The experiments were performed in a traditional host-based environment as opposed to a cloud-based VM used in research [28]. They employed a pre-trained model called Bidirectional Encoder Representations from Transformers (BERT) for transfer learning. The study utilized graphical features of malware to detect Android malware. The researchers validated their models using the Local Interpretable Model-Agnostic Explanation (LIME) and Shapley Additive Explanations (SHAP) libraries. The aim was to determine the impact of each feature on the accuracy of the model. The SHAP values were used to identify how much each feature contributed to the model output.

Liu et al. have used Explainable AI (XAI) methods to investigate the reasons behind the high performance of ML-based malware models in the presence of temporal inconsistencies in malware and benign samples [30]. They used Drebin, XMal and Fan et al. [31] to evaluate the performance of these models on a real-world Android malware dataset. Finally, they used LIME to evaluate the contribution of each feature to the model's output. The researchers found that the Drebin technique, which uses static analysis to identify malware, was less effective in identifying newer malware samples. In contrast, the XMal technique, which utilizes feature sets associated with API calls and permissions, was found to be more effective in identifying newer malware

samples by focusing on the behavior of the app rather than its static features. They also found that the LIME technique was effective in explaining the model's predictions by highlighting the most important features in the decision-making process. However, their research mainly focused on understanding the impact of temporal inconsistencies on the performance of ML-based malware detection approaches.

The importance of XAI cannot be overlooked when it comes to deep learning algorithms such as convolutional neural networks in the field of malware analysis and detection. Kinkead et al. recognize the importance of explaining predictions of Android malware classifiers model and figured out the locations important to the opcode sequence of Android apps using convolutional neural network (CNN) [32]. After identifying important regions using CNN for Android malware detection, the researchers compared these regions with those identified by the LIME method and found a close match. This comparison increased their trust and confidence in the CNN model. Their research is useful for security applications that consume sequence-based input.

As discussed in Section 2.1.1, backdoor attacks could be fatal. Severi et al. proposed a model-agnostic methodology to create backdoors in ML classifiers and used SHAP to analyze the vulnerability of classifiers to such attacks [33]. The study involved conducting static and dynamic analyses on different datasets, Portable Executable (PE) files, PDFs, and Android application files. SHAP was used in selecting high-contributing features in testing the feasibility of backdoor attacks. The evaluation of backdoor attacks was carried out against a variety of machine learning models such as RF, SVM, and deep neural networks (DNN). Researchers also claim that the proposed explanation-guided attack method is more robust and could also be used in non-security application domains.

Fan et al. discovered that the variability in the explanations produced by different

explainability techniques poses a challenge to relying on explanations generated by existing techniques for trustworthy malware analysis [31]. The researchers investigated the reliability of various explanation approaches by evaluating their consistency and sensitivity to changes in the dataset. They also designed metrics to assess the quality, stability, and robustness of explanation methods by calculating the similarity between generated explanations. They carried out sanity checks of LIME, Anchor, Local Rule-based Explanations (LORE), SHAP, and LEMNA in Android malware detection on malware classifiers (i.e., Multilayer Perceptron (MLP), Random Forest (RF), and Support Vector Machines (SVM)) using the metrics they have devised. As stated, the research is mainly focused on tackling the reliability problem of explanations generated by existing approaches.

Warnecke et al. [34] provide general recommendations about applications of explanation methods for deep learning in the security domain. The XAI methods such as LIME, LEMNA, SHAP, Gradients, Integrated Gradients (IG), and LRP were evaluated on different security systems against criteria such as accuracy, completeness, efficiency, and robustness. The research finds that the Integrated Gradients and LRP methods comply best with all requirements and recommend these techniques to use in the security domain.

Finally, Liu et al. [9] conducted a literature survey on deep learning methods for Android malware defenses. Our study can be considered as a continuation of the previous research conducted in [9], which had a general focus on discussing deep learning methods in Android malware. However, our research specifically aims to explore the explainability of deep learning algorithms for Android malware detection in recent years (years 2016-2023). We considered an extensive list of a total of 16 such papers for our review. The literature review on XAI for deep learning in Android malware along with their evaluation on different criteria is presented in Table 2.

Table 2: Summary of previous work

Work	Dataset	XAI Techniques	Program Analysis Approach
[27]	Performance metrics	SHAP	Online
[35]	Drebin+	LIME, CAM, SHAP	Hybrid
[28]	ImageNet, MNIST etc.	SHAP, LIME, LEMNA,CAM	Hybrid
[29]	CIC-InvesAndMal2019	LIME	Visual
[30]	Androzoo	LIME,SHAP	Hybrid
[36]	VirusTotal, AMD	LIME	Static
[33]	Grad-CAM	SHAP	Static
[37]	-	DistanceBased	Static
[38]	Google, HUAWEI App Store	MLP,LIME	Static
[32]	Drebin	LIME	Hybrid
[34]	Drebin, Genome	LRP,LIME,SHAP	Static
[39]	Androzoo	LEMNA	Static
[31]	LORE	static	Static
[40]	VirusTotal	MDI	Hybrid
[41]	PlayDrone	LIME	Static
[42]	Google Play	DroidDetector	Hybrid

We end this section by discussing open challenges in current XAI work on Android malware. There are only a limited number of studies that specifically focus on evaluating the reliability of XAI methods. Our literature survey shows only 2 out of 17 (11%) papers [31] [34] evaluated XAI methods for security, however, both of them employed different criteria for evaluation i.e. [34] employed completeness, efficiency, and robustness criteria for evaluation while [31] employed stability, robustness, and effectiveness as their XAI assessment criteria. There is currently no standard accepted method or set of criteria for evaluating XAI techniques in the context of Android malware. Furthermore, there is no clear consensus or recommendation on which XAI method would be most effective in the Android malware domain. Our literature survey shows only 1 out of 17 (6%) papers [34] provide recommendations on the use of the XAI technique, according to the research, LRP works best for security systems. Again, no paper has proposed a definitive recommendation on the best XAI approach in the Android malware context specifically. We need more research to gain a comprehensive

understanding of the practical applicability of XAI methods for addressing real-world Android malware issues.

CHAPTER 4

Experiments and Results

In this section, we present various experiments. First, we train and explain classic ML models, followed by DL models. We end this section by discussing the results and providing a comparative study on XAI techniques. Figure 7 outlines the deliverables or outcomes of this research.

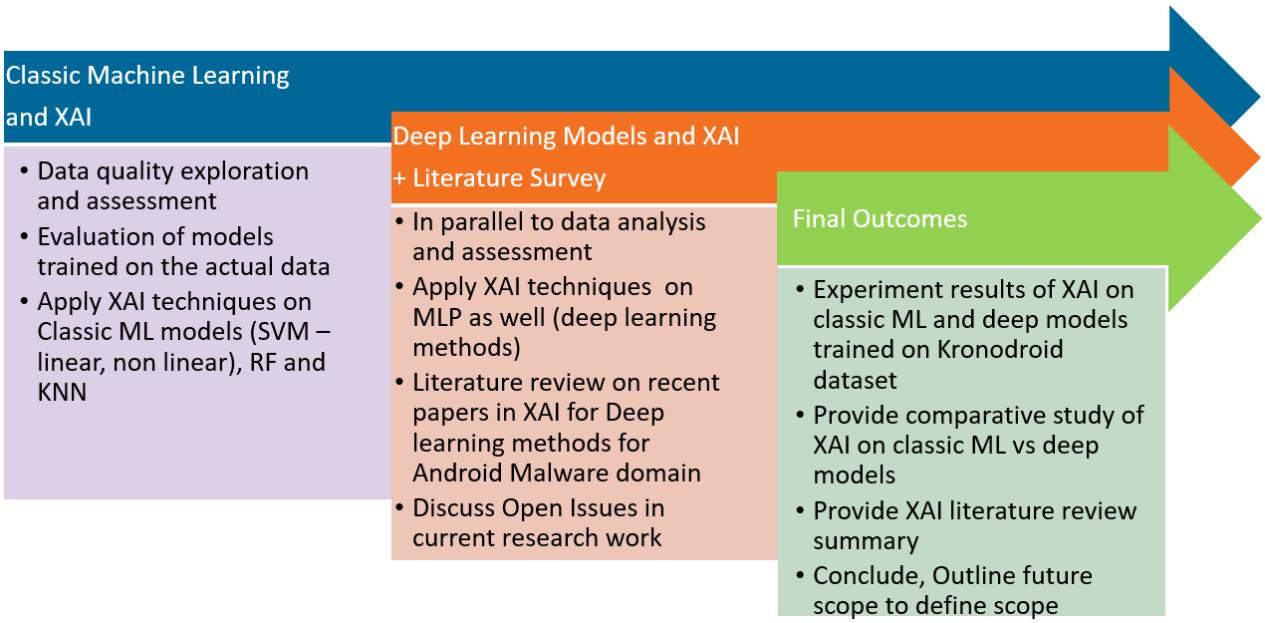

Figure 7: Research next steps

4.1 Dataset

For this research, we use the latest (the year 2021) KronoDroid dataset [43]. The KronoDroid dataset is a good choice for our project because the dataset is large, contains the most recent samples, and has numerous malware families. Moreover, this dataset takes the time effect into consideration, Therefore, it is possible to detect changes and evolution in Android malware over time through these experiments. Also, the presence of structured data means classic ML models like SVM can be built on it, and post hoc explanations can be carried out on the results. This dataset includes

labeled samples from 240 malware families, with 78,137 total samples, of which 41,382 correspond to malware and 36,755 correspond to benign apps. For each sample, 289 dynamic features (e.g., `system calls`) and 200 static features (e.g., `permissions`) are provided, and timestamps are also included. Various malware families consist of multiple samples that are collected over a long period of time. These samples share a code base and possess similar characteristics.

The top 10 malware families by sample size, as shown in Table 3 and illustrated by Figure 8. For the sake of this classification project, only the top 10 malware families are considered. For the sake of this project, the top 10 malware families by sample size were chosen to work with.

Table 3: Top 10 malware families by sample size

Case	Family	Sample Count
1	Airpush/StopSMS	7775
2	SMSreg	5019
3	Malap	4055
4	Boxer	3597
5	Agent	2934
6	FakeInst	2384
7	Locker/SLockerRansomware	1846
8	BankBot	1297
9	Dogwin	1145
10	FakeApp	994
-	Total	31046

4.2 Setup

All classic machine algorithms experiments are performed on a single host machine and deep learning experiments are performed on GPU. All experiments in this research are conducted on the computer, as detailed in Table 4.

4.3 Preprocessing

In the KronoDroid dataset, there are samples that have missing data. To address this, we removed samples with null values and infinity. This is because such missing

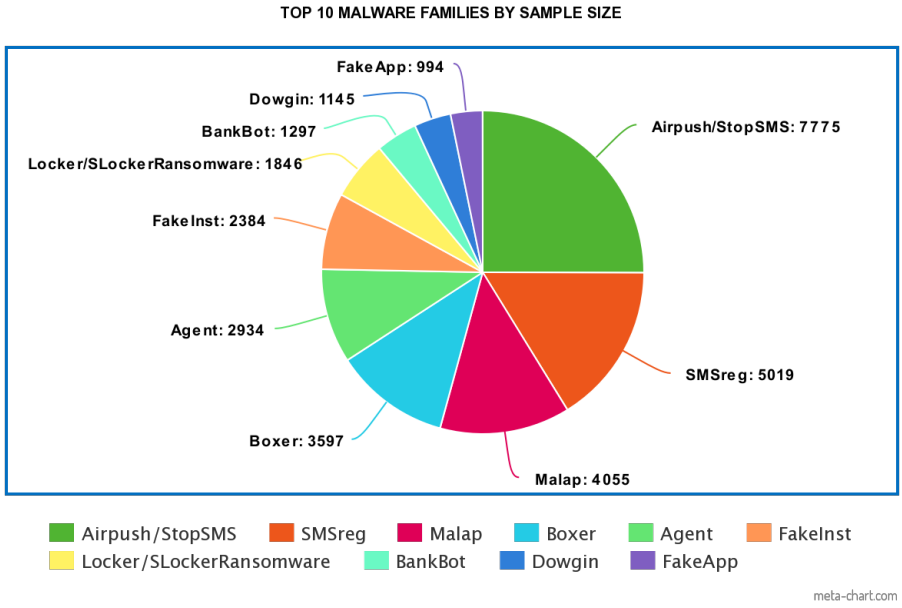

Figure 8: Top 10 malware families by sample size

Table 4: Computing resources used in experiments

Computing hardware	Details
Model	Dell XPS 13
Processor	Intel Core i5-7200U CPU @ 2.50 Ghz, 2.70 Ghz
RAM	8.0 GB
Operating System	Windows 10 Enterprise 64-bit

values can negatively impact the performance of machine learning algorithms. It is worth noting that the KronoDroid dataset has considerably fewer malware samples in one family compared to the other malware families. We discarded benign samples and considered only the top 10 malware families samples as shown in Figure 8. Prior to our work using this dataset eliminated labels, namely, `Detection Ratio`, `Package`, and other columns with object data from our research as these features were impacting the model accuracy negatively. As a result of this preprocessing step, our dataset contains a total of 468 features. Finally, we standardized data using a standard scaler.

4.4 Evaluation metrics

To evaluate the performance of each classifier in our experiments, we utilize two metrics: accuracy and F1-score. Accuracy is determined by dividing the total number of correct predictions by the total number of samples tested. The F1-score is calculated as the weighted average of precision and recall. The F1 score ranges between 0 and 1, with 1 representing the highest achievable score, The F1 score formula is as follows:

$$\texttt{F1} = 2 \times \frac{(\texttt{Precision} \times \texttt{Recall})}{(\texttt{Precision} + \texttt{Recall})}$$

where

$$\texttt{Precision} = \frac{\texttt{True Positives}}{(\texttt{True Positives} + \texttt{False Positives})}$$

and

$$\texttt{Recall} = \frac{\texttt{True Positives}}{(\texttt{True Positives} + \texttt{False Negatives})}$$

4.5 Implementation

Choosing the right dataset, data analysis, model building, and exploration of different XAI techniques and choosing the one suitable for the use case at hand is crucial as a part of this research, and needed most of the effort and time. Figure 9 details the different phases involved in carrying out this research, we worked step by step towards achieving the end goal of this research.

This study aims to explore and discuss the utility of XAI techniques in the Android malware domain. Towards this, we generate explanations and obtain interpretations on Support Vector Machines (SVM) - linear and non-linear, Random Forest (RF), k-Nearest Neighbors (k-NN), Multilayer Perceptron (MLP), and Convolutional Neural Networks (CNN). We perform various experiments from generating ante hoc explanations using the model's inherent interpretable capabilities to post hoc explanations, global to local explanations using various model agnostic tools such as LIME, SHAP, Eli5, PDP Plots, and model-specific tools such as CAM performed on

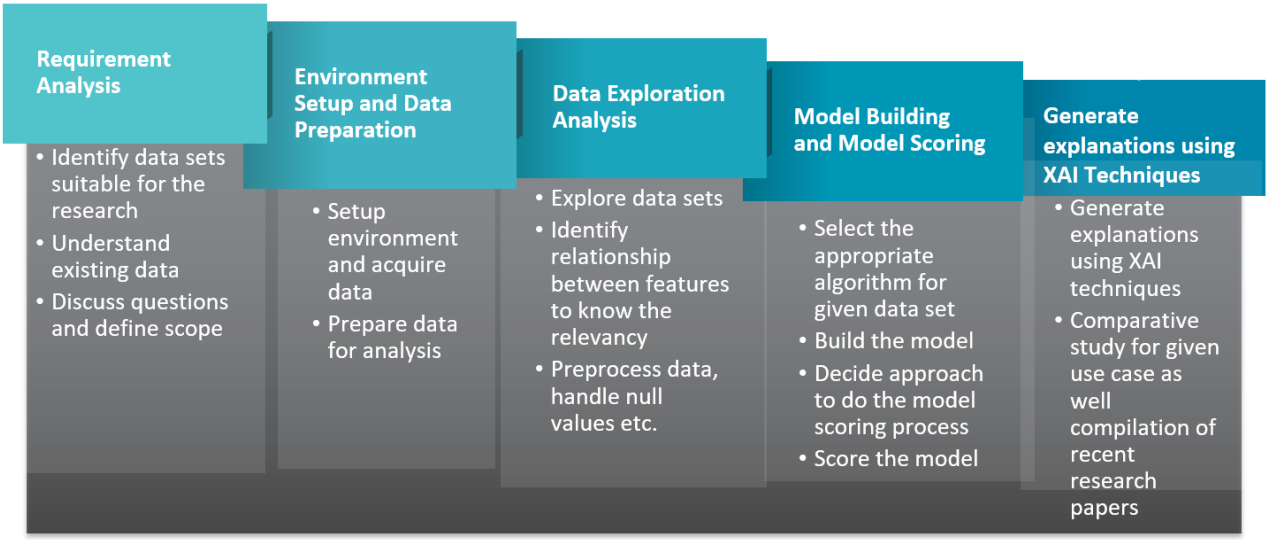

Figure 9: How to get there?

the implementation of the models that we mentioned in Section 2.6. All experiments were implemented in Python using various libraries. The majority of experiments utilized the `Scikit-learn` package, except for the CNN experiments, which were implemented using the `Tensorflow` and `Keras` libraries. Then we compare the results generated by different XAI techniques on our best-performing classifiers for the top 10 malware families. We used `GridSearchCV` function for 5-fold cross-validation. For the experiments, we used all of the dataset samples we mentioned in Section 4.1 for training our ML/DL models.

In our experiments, we utilized various machine learning and deep learning hyperparameters configurations. Specifically:

1. To determine the optimal kernel for our dataset, we conducted initial tests for SVM, which led us to select the Gaussian radial basis function (RBF).

2. In the case of RF, we performed several experiments with different hyperparameter values and found that setting the number of n_estimators to 100 yielded the

best performance on our dataset. We chose to keep the other hyperparameters at their default values in Scikit-learn, as we found that this provided satisfactory results and avoided overfitting. By doing this, we were able to achieve high accuracy and F1 scores for our RF experiments.

3. For k-NN, the value of k was set to 5 which gave best results for our dataset. The choice of k is an important hyperparameter for k-NN and determines the number of nearest neighbors used to classify a sample.

4. Our MLP architecture involves 300 hidden layers, which is a relatively large number of layers. We chose the rectified linear unit (ReLU) activation function is a popular choice for neural networks, known for its ability to reduce the likelihood of the vanishing gradient problem. The learning rate *alpha* for our experiments is set to 0.0001.

5. Our CNN model utilizes max pooling. We experimented with various hyperparameters and found that the below combination works best for our model. An initial number of convolution filters (32), filter size 2×2, and percentage dropout (0.25) worked best for this research.

4.6 Results and Discussion: Performance of ML/DL models

We collected a total of 78,000 malware samples from different families and used them in our experiment. We split the data into 60% for training, 20% for testing, and 20% for validation. All the models were trained by taking only the top 10 malware families samples. This division of the dataset is common in machine learning experiments to ensure that the model is not overfitting to the training data and is able to generalize well to new, unseen data. All models were configured to the optimal values of their hyperparameters. To provide a more comprehensive evaluation of the

model's performance than accuracy alone, we used the F1 score as a benchmark since it takes into account precision and recall metrics. Performance metrics of the ML and DL models were computed on the test data, and the corresponding results are presented in Table 5. The F1 score shows that RF outperformed other models. The top 3 models by F1 score are RF, MLP, and CNN - with F1 scores of 93.14%, 92.07%, and 90.91% respectively. The F1 scores of the k-NN and SVM models were found to be 90.54% and 89.98%, respectively. While these scores are respectable, they were not as high as the scores obtained by some of the other models. The recall metric, which measures the proportion of actual positives that are correctly identified, is an important metric for malware detection as we want to minimize false negatives. The RF model achieved the highest recall score of 93.22% among all the models evaluated. Based on the performance comparison results, we can infer that RF outperformed the other models.

Table 5: Performance of ML and DL models

Model	Accuracy	Precision	Recall	F1
Linear SVM	91.80	91.94	87.19	89.17
Nonlinear SVM	89.17%	89.37%	89.17%	88.98%
RF	93.22%	93.18%	93.22%	93.14%
k-NN	90.61%	90.52%	90.61%	90.54%
MLP	92.09%	92.06%	92.09%	92.07%
CNN	90.76%	90.89%	89.76%	90.91%

4.7 Results and Discussion: XAI Results

After training the ML and DL models, the next step was to interpret and explain their results using various XAI techniques such as LIME, SHAP, Eli5, PDP plots, and CAM. For SVM, k-NN, and MLP models, KernalSHAP was used to explain the results, while TreeSHAP was used for RF and DeepSHAP for DL models. Among the three SHAP variants, TreeExplainer was the fastest. The results of the XAI

experiments were presented as graphs, and their interpretations were discussed in detail.

4.7.1 Feature Importance and Eli5

Feature importance using model coefficients is a method for explaining the results of ML models. The method identifies which features contribute most significantly to the output by analyzing the coefficients assigned to each feature in the model. These coefficients indicate the strength and direction of the relationship between each feature and the prediction. In order to facilitate ante hoc explanations, `sklearn GridSearchCV` package provides a `feature_importances_` method and `coef_` method of `SVC` output the top contributing features influencing the model outcome. We calculated feature importance by standardizing features and taking a look at the coefficients of the SVM and RF models. Figure 10 and Figure 11 show the top 10 important features of linear SVM and RF respectively. `BLIND_DEVICE_ADMIN`, `SET_WALLPAPER` and `READ_SMS` are the biggest drivers of model predictions of linear SVM while for RF, `ACCESS_COARSE_LOCATION`, `total_perm`, and `read` contribute the most. This gives us an idea of the relative feature importance of one feature vs. another. We noticed that the RF feature importance results on the train and test datasets are consistent, therefore, we can tell that RF is not overfitting on the KronoDroid dataset. Extracting coefficients is not possible in a nonlinear rbf kernel.

Permutation importance provides insights into the impact of randomly shuffling data on model accuracy. Permutation importance measures the change in model error after a single model feature's values have been shuffled. The Python library, `ELI5`, offers a simple method for computing permutation importance. Figure 12 shows the permutation importance of RF. The output generated by `ELI5` shows the most important features at the top and the least important at the bottom. Each row

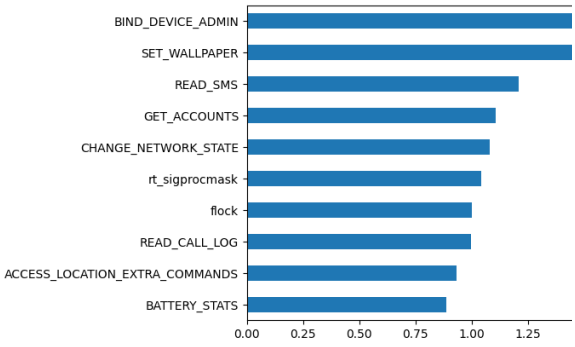

Figure 10: Linear SVM feature importance

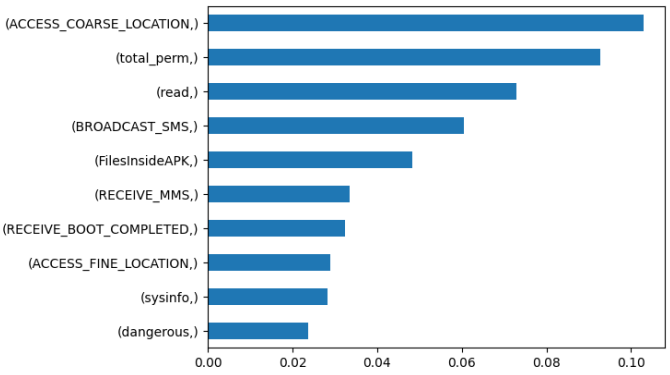

Figure 11: RF feature importance

in the output represents a feature, and the first number indicates how much the model's performance decreased when that feature was randomly shuffled. The number after the ± symbol represents the degree of variation in performance across multiple shuffles, providing a measure of the randomness. By examining the permutation importance output, we can identify which features are most important for the model's predictions and potentially improve the model's performance by focusing on those features. Figure 12 the top 3 features are `SEND_SMS`, `RECEIVE_BOOT_COMPLETED`, and `TimesSubmitted`, while the 3 least significant are `prctl`, `READ_LOGS`, and `fchmod`. Shuffling the data in the most important `SEND_SMS` column caused the RF mean squared error to increase by 0.0010.

4.7.2 LIME Interpretations

LIME provides feature importance scores for individual data samples, allowing us to understand how much each feature contributes to the prediction for that particular data point. It provides local interpretability by focusing on individual instances rather than the overall model behavior, explains why the model is predicting the output, and shows that a small change in features contributing to the result can most impact the prediction probability.

We define a tabular explainer object using the lime library because the given

Weight	Feature
0.0033 ± 0.0010	SEND_SMS
0.0032 ± 0.0003	RECEIVE_BOOT_COMPLETED
0.0021 ± 0.0020	TimesSubmitted
0.0015 ± 0.0011	GET_ACCOUNTS
0.0014 ± 0.0016	FilesInsideAPK
0.0012 ± 0.0006	GET_TASKS
0.0011 ± 0.0017	UFileSize
0.0011 ± 0.0005	READ_EXTERNAL_STORAGE
0.0010 ± 0.0002	READ_PHONE_STATE
0.0008 ± 0.0006	dangerous
0.0008 ± 0.0013	signature
0.0008 ± 0.0002	SYSTEM_ALERT_WINDOW
0.0007 ± 0.0011	mprotect
0.0006 ± 0.0005	WRITE_SECURE_SETTINGS
0.0005 ± 0.0009	sysinfo
0.0005 ± 0.0004	CHANGE_CONFIGURATION
0.0005 ± 0.0009	fsync
0.0005 ± 0.0012	prctl
0.0004 ± 0.0004	READ_LOGS
0.0004 ± 0.0008	fchmod
… 448 more …	

Figure 12: RF interpretability: Permutation importance using Eli5

KronoDroid dataset has tabular data. It takes the input parameters such as the trained model, features used in model training, and labels of target classes. We interpreted the model explainer based on the values available in the test set.

Figures 13 through 16 show the LIME explanations for predictions of the models such as SVM, k-NN, RF, and MLP for the first instance of the test dataset. All models correctly classify the first instance of test data with higher confidence as class 'ransomware' which is also the true label for this instance. The left side of the lime explanation shows the probability with which the sample is classified as 'ransomware' - the pink color indicates that the contribution is towards the 'ransomware' family, and the purple color indicates that the contribution is towards 'malap' family. Figure 13 through 16 tells that models SVM, k-NN, RF, MLP classify

this instance as ‘ransomware’ with probabilities of 0.82, 1.0, 1.0, 1.0 respectively. LIME output displayed in Figure 13, shows the classification result for the top two classes of high probability for the first instance prediction, in the middle of each figure, there is a graph telling the reason why this instance belongs to the class ‘ransomware’, it identifies and lists how strongly that feature contributes to the model’s prediction for the specific instance being explained, in the order of importance. At the (right side) end of each figure, there is a table that is telling about what are the values in test data for the identified features, pink color values are the reason for the final prediction, and green color values are the ones that do not support the prediction outcome. From Figure 15, we can tell that this first instance of the test set has a value of `SEND_SMS` less than or equal to -1.06 and `CALL_PHONE` less than or equal to -0.67 makes it more likely to be ‘ransomware’ while Figure 13 tells that instance values such as `REQUEST_INSTALL_PACKAGES` less than or equal to -0.05 and `REQUEST_IGNORE_BATTERY_OPTIMIZATION` less than or equal to -0.08 makes it more likely to be ‘not ransomware’. Overall, considering all the features of the sample (on the right panel), the sample is predicted to be ransomware. These observations fit our intuition and our knowledge about the ‘ransomware’ family. Knowing this, we are more confident that the model is making predictions correctly as per our intuition.

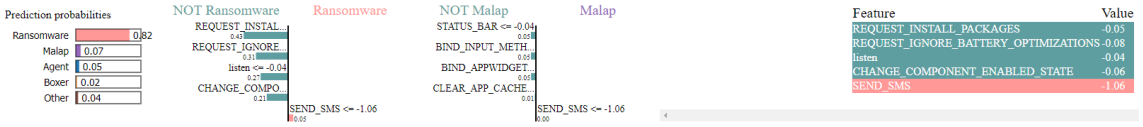

Figure 13: SVM - LIME explanations on correctly classified sample

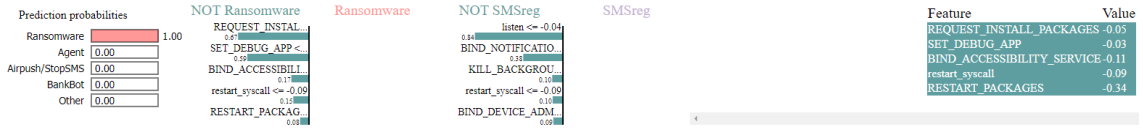

Figure 14: k-NN - LIME explanations on correctly classified sample

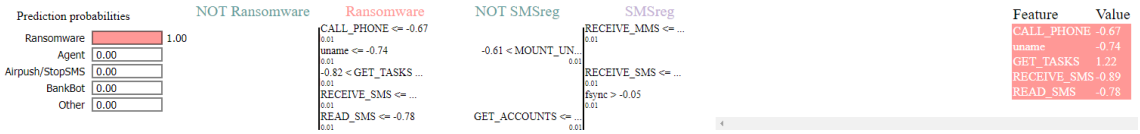

Figure 15: RF - LIME explanations on correctly classified sample

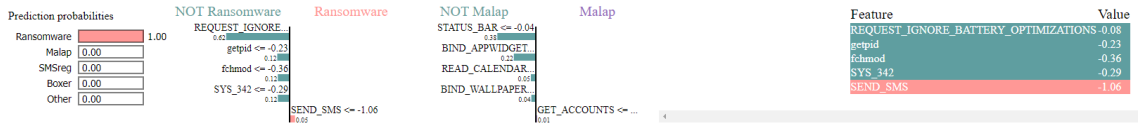

Figure 16: MLP - LIME explanations on correctly classified sample

Figures 17 through 20 show the LIME explanations for non-linear SVM, k-NN, RF, MLP for the 97th instance of test data that is misclassified by all of these models.

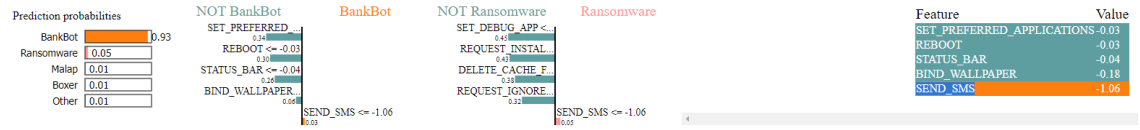

Figure 17: SVM - LIME explanations on misclassified sample

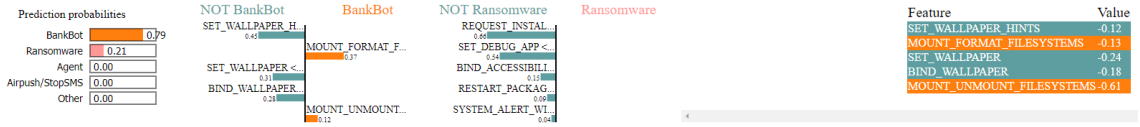

Figure 18: k-NN - LIME explanations on misclassified sample

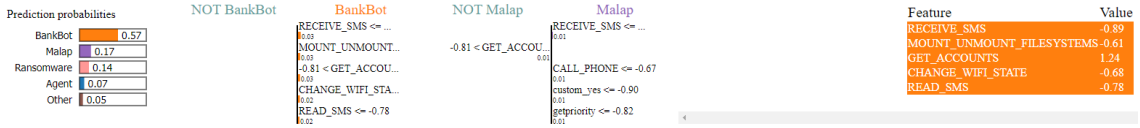

Figure 19: RF - LIME explanations on misclassified sample

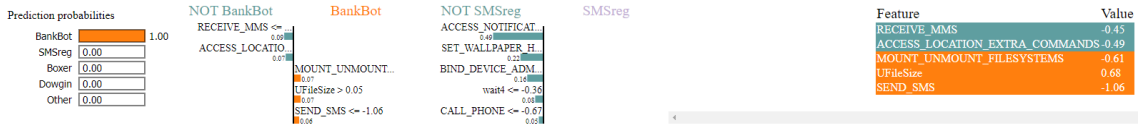

Figure 20: MLP - LIME explanations on misclassified sample

We considered another scenario where we analyzed the incorrectly predicted instance using LIME. We considered the 97th sample of the test dataset, all models misclassify this instance as the 'BankBot' family whereas it actually belongs to the 'Malap' family. 0.93, 0.79, 0.57, and 1 are the probabilities with which SVM, kNN, RF, and MLP classify this sample as the 'BankBot' family. Orange color values are the reason for the final prediction, and green color values are the ones that do not support the prediction outcome. Figure 17, Figure 18, and Figure 20 show that the above instance is incorrectly classified as 'Bankbot' owing to the features that denote `MOUNT_UNMOUNT_FILESYSTEMS` less than or equal to -0.61 and `SEND_SMS` less than or equal to -1.06. These features are found to be contributing to incorrect classes. This might have swayed the prediction to be incorrect. RF is the only model that makes a guess at the possibility of this sample being the correct 'Malap' family with a probability score of 0.17. Figure 19 shows that the features such as `GET_ACCOUNTS` greater than -0.81 make the RF model classify this as 'not Malap', the value of `GET_ACCOUNTS` for this sample is quite high which is 1.24. Another feature value `RECEIVE_SMS` less than or equal to -0.89 is a key factor in determining the model outcome as it plays a role in deciding both the families 'Bankbot' and 'Malap'. It gives strong support for the class 'BankBot' with a feature importance value of 0.3 as opposed to a value of 0.01 for the 'Malap' family. This makes the RF model less confident about making a prediction as 'Malap'. This explanation is insightful and it helps in understanding what features actually drove the prediction to be incorrect.

4.7.3 SHAP Interpretations and PDPs

We use SHAP's `KernalExplainer`, `TreeExplainer`, and `DeepExplainer` to explain the predictions. `KernelShap` is used to generate explanations on SVM, KNN model, and `TreeShap` is used to explain predictions on the RF model. To handle the

resource-intensive task of computing SHAP values for a large number of samples, we employed sampling and Recursive Feature Elimination (RFE) techniques. Figure 21 shows RFE accuracy VS number of features graph on RF classification. The graph shows the accuracy of the RF model goes high initially, but then later gets plateaued out when 10 or more features are selected. We also observed that the computation of Shapley values using `KernelSHAP` is exceptionally slow compared to `DeepSHAP` and `TreeSHAP`.

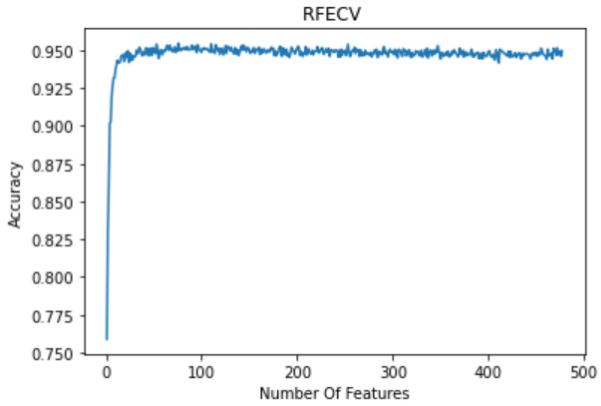

Figure 21: RFE accuracy vs features

Using global model interpretation techniques, we get an idea of how our model behaves in general. Toward this purpose, we use the SHAP's two most used global model interpretation plots - SHAP variable importance plot and the SHAP dependence plot. We discuss them in the below subsections.

4.7.3.1 Variable Importance Plot — Global Interpretability

The `shap.summary_plot()` function is a part of the SHAP library in Python, which can be used to generate a visual representation of the feature importance for a machine learning model using Shapley values. Figures 22 through 25 provide the SHAP global explanations - top 10 most significant variables in descending order - on our trained learning models (SVM-RBF, RF, and KNN), and deep learning models

(MLP and CNN). The plot displays each feature as a horizontal bar with its length indicating its importance. The average magnitude impact of the model output is on the x-axis while the y-axis shows the average impact (mean SHAP value across all samples) of that variable on model output. First of all, we noticed that the global explanation plot on both the train and test datasets is the same which confirms the model is not overfit on the input data. It is interesting to mention that all models share the contributing features, and the top two ranking features for all of them are `dangerous` and `total_perm`. Further, the analysis indicates that the nonlinear SVM model is particularly sensitive to the top features, as evident from the high SHAP values of the most significant features such as `dangerous` and `total_perm` with values of 8.1 and 7.8 respectively. In contrast, other models such as KNN, RF, and MLP have less variation in their feature contributions, indicating a more uniform distribution of feature importance.

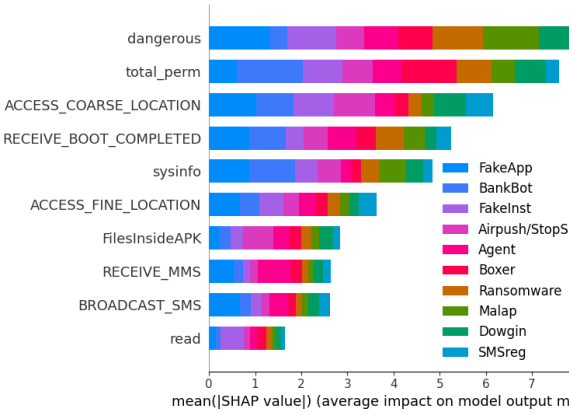

Figure 22: SVM - Variable importance plot

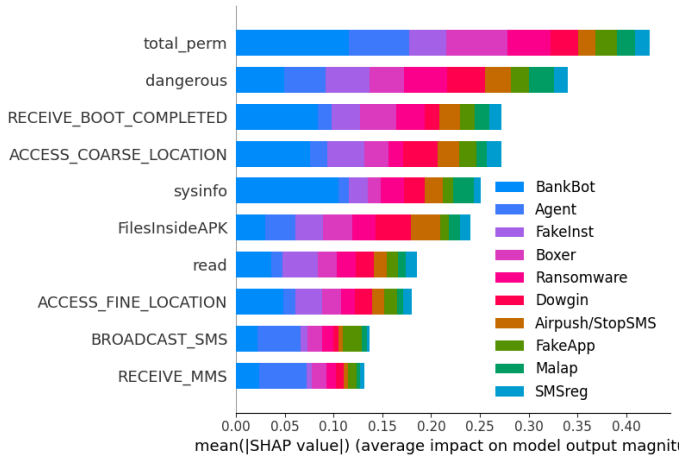

Figure 23: RF - Variable importance plot

Figure 26 presents a SHAP value plot that displays the direction and magnitude of the relationships between the predictors and the target variable. The plot is generated using the `shap.summary_plot` function. The plot encompasses all the data points

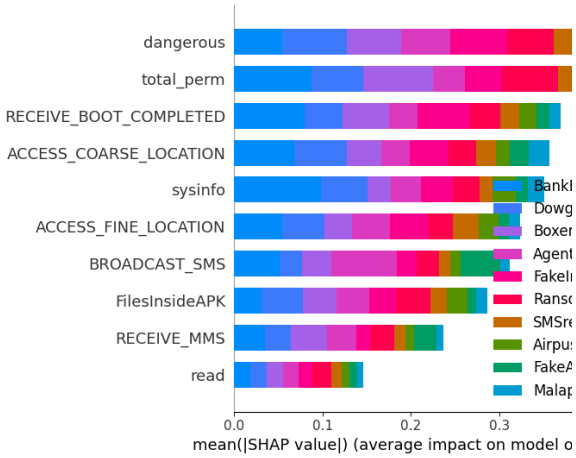

Figure 24: KNN - Variable importance plot

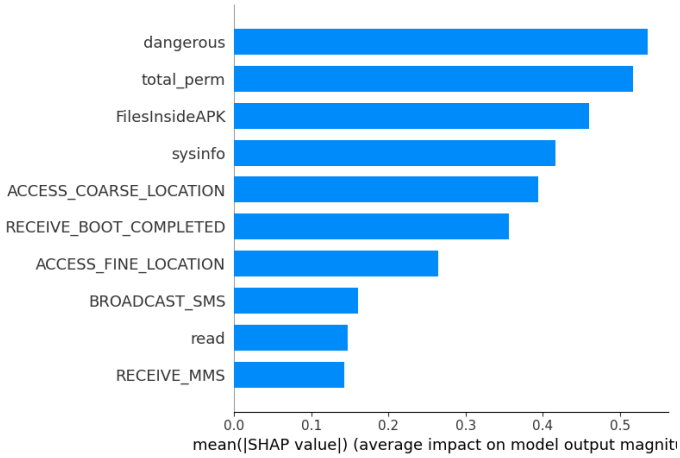

Figure 25: MLP - Variable importance plot

in the test dataset. The code `shap.summary_plot` produces the plot. This plot is made of all the dots in the test data. It provides three major important pieces of information: First, it signifies the feature's importance in descending order. Second, the plot demonstrates how the SHAP value affects the prediction, a lower SHAP value indicates a higher prediction, while a higher SHAP value indicates a lower prediction. The red color of the dot shows that an original value of a variable value is high for that observation and the blue color shows it is low. Third, it tells us the correlation of an input feature with a target variable. In the figure, high values of `ACCESS_COARSE_LOCATION` and `RECEIVE_BOOT_COMPLETED` have a high and negative influence on the model outcome which means it is strong negative association or correlation with the target variable. Similarly, the low value of `sysinfo` has a positive impact on the model outcome.

4.7.3.2 SHAP Dependence Plot — Global Interpretability

Partial dependence plots show two-way interactions between input variables and dependent variables in complex models.

In our analysis, we used the `dependence_plot` function of the SHAP library to

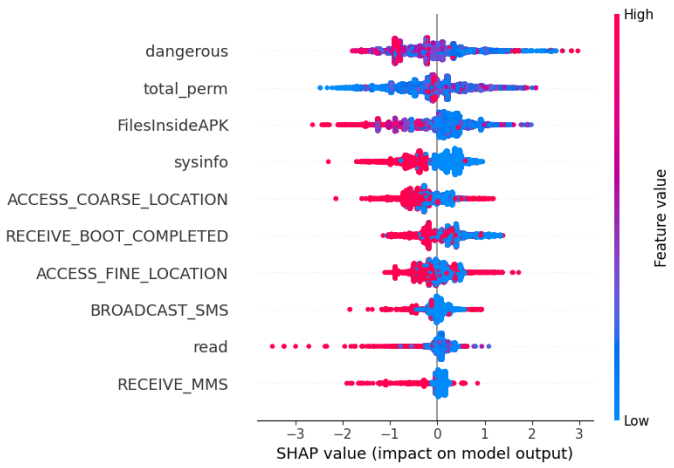

Figure 26: MLP global interpretation - Value plot

generate PDPs. This function automatically includes the variable that the chosen feature interacts most with, allowing us to visualize two-way interactions between variables. The resulting plots can help identify the functional form of the relationship between the target variable and the feature of interest, including any non-linearities or threshold effects. The PDP on our models shown in Figures 27 through 30 show PDP for variable `dangerous` for RBF SVM, k-NN, RF, and MLP respectively. The analysis reveals a linear pattern between the predictor variable `dangerous` and the target variable. Additionally, `dangerous` frequently interacts with other variables such as`ACCESS_COARSE_LOCATION`, `FilesInsideAPK`, and `sysinfo` for SVM, RF, K-NN, and MLP models, respectively, which is consistent with the negative relationship shown in the variable importance plot. In summary, the PDPs gave additional insights into the relationship between the important input features and the model output. Using this, we understood the model's behavior.

Similarly, Figures 31 through 34 show PDP for feature `FilesInsideAPK` and Figures 35 through 38 show PDP for `total_perm` respectively.

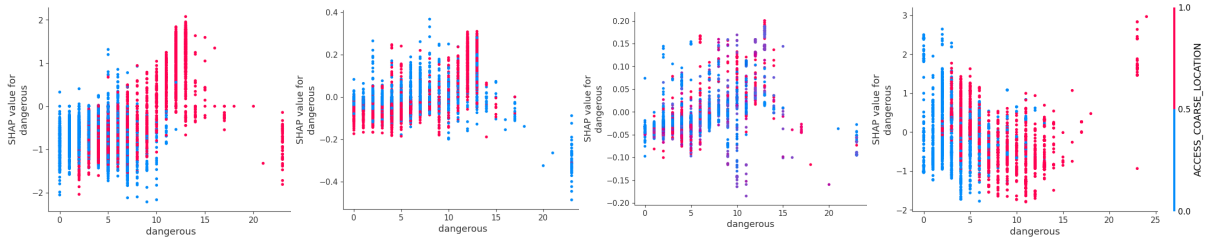

Figure 27: SVM PDPFigure 28: KNN PDP Figure 29: RF PDP Figure 30: MLP PDP

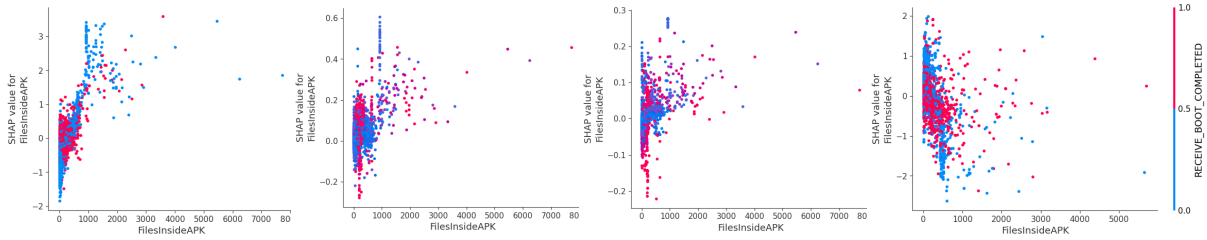

Figure 31: SVM PDPFigure 32: KNN PDP Figure 33: RF PDP Figure 34: MLP PDP

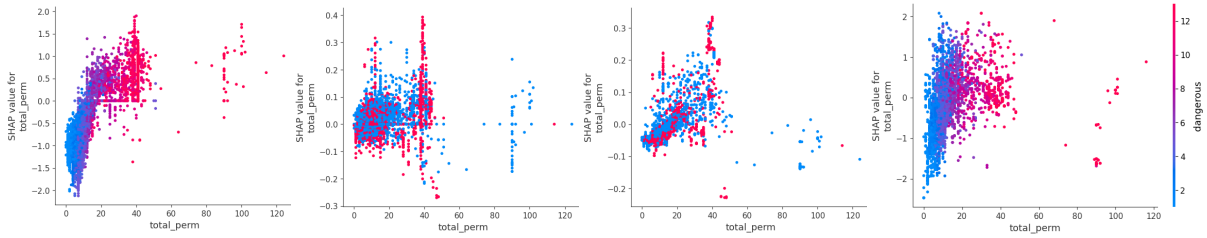

Figure 35: SVM PDPFigure 36: KNN PDP Figure 37: RF PDP Figure 38: MLP PDP

4.7.3.3 SHAP Value Plot — Local Interpretability

Finally, we generate local explanations for individual predictions using SHAP `force_plot` method. We input the following to the `force_plot` method: the base value used in the following plot is the average of the model output calculated over the training data - the 'base value' is a reference point used in SHAP, then the next parameter passed is the SHAP values computed on training data using the technique explained in Section 2.6.3, the last parameter is the observation values for which we wish to get a local explanation. Figure 39 shows the SHAP local explanation generated on the first instance of the test dataset. Individual SHAP value plot for

instance 0 of test data of RBF SVM shows that the output value for this instance is 0. It is the model prediction for that observation (both the actual class and predicted class of this instance is the 'BankBot' family). The graph also shows the base value or mean prediction, or mean(y_{hat}) of 3.1. We ran the experiments on the top 10 malware families which means our y_{hat} values range from 0 to 9. The base value is the predicted value when no features are known for the current output. This is justified by the fact that the mean prediction of y_{test} is 3.1. Red indicates features that positively influence the prediction (i.e., push it higher or to the right), while blue represents features that negatively influence the prediction (i.e., push it lower or to the left). `FilesInsideAPK`, and `total_perm` has a more positive impact on the target variable. `FilesInsideAPK` has a positive impact on the model outcome. In the context of the model output plot, a high value of the feature `FilesInsideAPK` is associated with a higher prediction, and this feature has a value of 482 in the current example, which is higher than the average value of 183. Therefore, this feature has a positive impact on the prediction and pushes it toward the right of the plot. This also indicates that for this specific instance, the feature `sysinfo` has a negative effect on the model prediction. The value of `sysinfo` for this instance is 0, which is below the average value of 0.39. This below-average value contributes to pushing the prediction to the right, towards a higher value. No feature values for this observation push the prediction to the left for this observation.

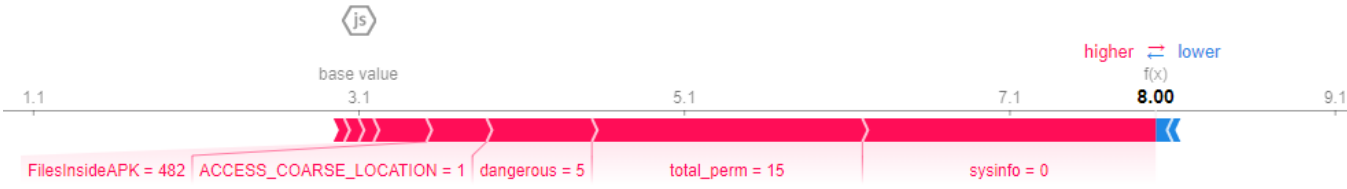

Figure 39: SHAP explanations on MLP - Interpret last observation

Appendix B can be referred to for more SHAP explanations on other instances of test dataset.

4.7.4 Grad-CAM

Convolutional neural networks (CNNs) use model-specific mechanisms such as Class Activation Mapping (CAM) to increase their interpretability or to create surrogate models for other nonlinear models. For this experiment, we represent the input array as an image and and a 1D CNN is trained using the original 1-dimensional array of each dataset sample. We generate explanations using CAM. The output of CAM is reshaped to a 2-dimensional image we reshape each sample to be 2-dimensional, specifically a 22×22. The 468 features are mapped to individual pixels and the rest are filled with zeros. The ordering of the pixels follows the RF ranking of feature importance. The pixels corresponding to the more important features are placed in the top-left of the image, while those for less important features are located towards the bottom-right. This allows for the visualization of the contribution of each feature to the model's output by looking at the corresponding regions of the image. We use `iNNvestigate` library to generate Grad-CAM output on our CNN model output. 3 steps involved in generating explanations are analyzing a prediction, training an analyzer, and analyzing a prediction w.r.t to a specific output neuron. The method `create_analyzer` of `iNNvestigate` works by analyzing the operations the network function uses to extract or compute the output, i.e. how would changing an input feature change the output? It also analyzes the components of the input that cause the output, i.e., which parts of an input image or which directions of input are used to determine the output Lastly, it attributes the importance of input features for the output, i.e., how much would changing an input feature change the output? Using the `gradient` function, we analyzed an image from the test dataset to obtain the

gradient of the output neuron with respect to the input. Figure 40 shows the sample test image reshaped as 22×22 grayscale image and pixels are ordered as their RF importance. Figure 41 shows the Grad-CAM output for the prediction made by CNN for our sample, we can visually verify where in the image the CNN is looking. It highlights the important features contributing to the CNN model outcome, that is features numbered from 88 to 374.

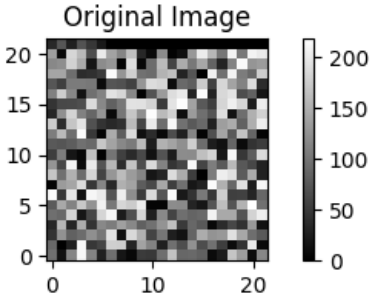

Figure 40: Original test image

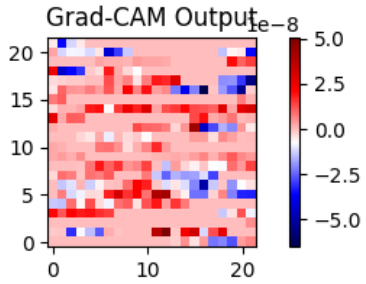

Figure 41: Grad-CAM explanation

4.8 Comparative study of XAI techniques

Among all the XAI techniques, SHAP is widely used in the research community, followed by CAM and LIME [44]. Figure 42 shows the usage of XAI tools over time. SHAP is a powerful technique as it offers both global as well as local explanations as opposed to other techniques. Also, it offers visualization plots for a variety of problems such as PDP - for getting insights on the relationship between features and target variables, and also with other features, variable importance plot - to get a global explanation of the model and value plot - to get a comprehensive overview of the model's behavior and for understanding the underlying relationships between the predictors and the target variable. Moreover, we found that the `TreeShap` and `DeepShap` kernels provided by SHAP run faster and computation time is significantly lower for generating explanations on the global/local level even for large datasets. For

our experiments, `TreeShap` took around 53 seconds or less to complete the execution given the dataset size is 41,382. `KernelShap`, on the other hand, is usually slow and took about 1 hour to run and analyze the results of our experiments. Our results show that SHAP provided useful insights into the experiments we performed.

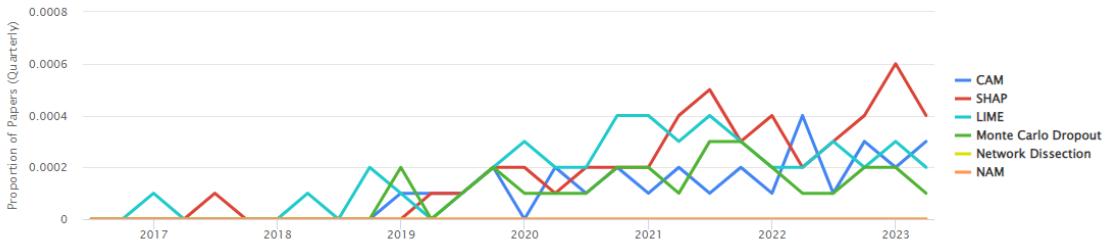

Figure 42: XAI tools usage over time [44]

On the other hand, LIME provides only local explanations. The technique is easy to use and works on tabular data, text, as well as images. The execution time of LIME, for our experiments, took less than 30 seconds, which is less as compared to SHAP. We were able to verify the results generated by LIME as it matches our intuitions and domain knowledge. Both SHAP and LIME are model-agnostic and can be used on a variety of classic as well as deep learning algorithms.

We observed that the feature importance method described in Section 4.7.1 that uses Eli5, Permutation Importance, and SHAP as described in Section 4.7.3.1 produce consistent results for variable importance plots. Hence, both techniques can be trusted and reliable.

Finally, CAM is a powerful technique when it comes to CNN. CAM is tailored toward working with images. It is model-specific but helps gain more information on CNN model outcomes. Deep learning techniques are popular in the Android security domain and CAM has received traction [44], it is the most widely used XAI technique after SHAP. CAM provides an advantage when it comes to working at different

attribution levels.

CHAPTER 5

Conclusion and Future Work

In this paper, we performed a comparative study of XAI techniques for a variety of classic ML models (SVM Linear, SVM-RBF, RF, and k-NN) and deep learning models (MLP, CNN) with a focus on the Android malware domain. We found that RF performed best among all models with an F1 score of 93.14, followed by MLP with an F1 score of 92.07. We applied XAI to these models to evaluate our models, we employed a diverse set (local, global, model agnostic, model-specific, ante hoc, and post hoc,) of techniques such as LIME and SHAP, Eli5, PDP, and CAM for explainability in our analysis, to explain the model outcomes and features contributing to the model outcomes.

From the obtained results, we conclude that SHAP is an all-in-one package and gives more flexibility in choosing from different interpretability methods. It allows global as well as local explanations. The explanations generated by SHAP are intuitive and match our domain knowledge. LIME came in handy for generating explanations at a more granular level (individual instances). CAM results uncovered outcomes of complex models such as CNN trained on our KronoDroid dataset. Using our SHAP, we could verify that our models are not overfitting as global explanations generated on the train and test datasets remained the same. We observed that the classic ML models like SVM and k-NN have the least performance accuracy than deep learning models however due to their natural probabilistic, algebraic, or geometric interpretation; they are inherently interpretable and it is easy to obtain global explanations for classic ML models using model's inbuilt interpretability methods. RF on the other hand worked well but also provides easy-to-understand explanations. On the other hand, the complexity of the neural structure in deep learning models means that there can be numerous interactions between neurons and their corresponding weights, making

it difficult to isolate the contribution of individual features to the model's output. This complexity highlights the need for effective explainability techniques to provide insights into the model's decision-making process. In our research, MLP and CNN performed well but cannot be explained globally using the ante hoc technique. We used LIME, SHAP, and CAM to uncover the complex models and we conclude, finding the right balance between accuracy and explainability is crucial.

We also conclude that there is an overlap between global explanations generated by interpretability methods provided by SVM, RF feature, Eli5, and SHAP variable importance plot methods. This means that the global explanation results are consistent across different XAI techniques. However, in our case, local explanations produced by LIME for the particular instance do not fully overlap which calls for the need for evaluating available XAI techniques thoroughly in the Android security domain. In contrast, SHAP produces coherent local interpretations for a specific instance of the test dataset that are consistent across all models. It is interesting to mention that all models share the same contributing features at the global level, and the top two ranking features for all of them are `dangerous` and `total_perm`. Further, the analysis indicates that the nonlinear SVM model is particularly sensitive to the top features, as evident from the high SHAP values of the most significant features such as `dangerous` and `total_perm`. In contrast, other models such as KNN, RF, and MLP have less variation in their feature contributions, indicating a more uniform distribution of feature importance.

No XAI research in the security domain to date has explored PDP plots and Eli5 for generating explanations. Our research shows Eli5 is effective in providing global explanations and PDP plots are efficient in determining the relationship between two variables and how it influences the model outcome.

We also examined and conducted a literature survey on XAI research for deep

learning methods in the Android malware domain for recent years (2016-2023). The previous research [9] conducted a literature review on deep learning methods for Android malware. Our research is an extension of the previous research in [9] with a specific focus on XAI work on deep learning methods in the Android malware domain for recent years. The compilation of all recent XAI research for deep learning in Android malware along with their evaluation on different criteria is given in Section 3. We believe this survey will help future researchers in applying XAI effectively in detecting Android malware.

At the end of Chapter 3, we discussed open issues in XAI for Android malware which opens potential avenues for future research work. Firstly, there is no general consensus on evaluation methods for XAI techniques for deep learning in Android malware or cybersecurity in general. We need evaluation criteria that holistically evaluate XAI techniques in the Android malware domain. Moreover, our research shows the lack of consistency in local explanations generated by LIME for the particular instance across different models, which means we need an agreeable evaluation method for XAI in Android malware. There are only a limited number of studies that specifically focus on this topic. Our literature survey shows only 2 out of 17 papers [31] [34] evaluated XAI methods for security, however, both of them employed different criteria for evaluation. Hence, future research should focus on developing an agreeable usable method for evaluating XAI providing guidelines for users. We also suggest that future researchers make more efforts at quantifying XAI results using existing criteria. Furthermore, more research is necessary to provide guidance on the most effective XAI approach for different circumstances.

LIST OF REFERENCES

[1] M. Stamp, *Information Security: Principles and Practice.* Wiley, 2011.

[2] M. Stamp, *Introduction to Machine Learning with Applications in Information Security*, 2nd ed. CRC Press, 2023.

[3] S. Rezaei, A. Afraz, F. Rezaei, and M. R. Shamani, ''Malware detection using opcodes statistical features,'' in *2016 8th International Symposium on Telecommunications (IST)*, 2016, pp. 151--155.

[4] Google, ''AI explanations whitepaper,'' https://storage.googleapis.com/cloud-ai-whitepapers/AI%20Explainability%20Whitepaper.pdf.

[5] C. Molnar, *Interpretable Machine Learning*, 2nd ed. Independently published, 2022, https://christophm.github.io/interpretable-ml-book.

[6] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, ''Explainable AI: A review of machine learning interpretability methods,'' *Entropy*, vol. 23, no. 1, 2021, https://www.mdpi.com/1099-4300/23/1/18.

[7] F. Yan, S. Wen, S. Nepal, C. Paris, and Y. Xiang, ''Explainable machine learning in cybersecurity: A survey,'' *International Journal of Intelligent Systems*, vol. 37, no. 12, pp. 12 305--12 334, 2022, https://onlinelibrary.wiley.com/doi/full/10.1002/int.23088.

[8] P. Mishra, *Model Explainability and Interpretability.* Apress, 2022, pp. 1--22, https://doi.org/10.1007/978-1-4842-7158-2_1.

[9] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, ''Deep learning for android malware defenses: A systematic literature review,'' *ACM Computing Surveys*, vol. 55, no. 8, pp. 1--36, 2022.

[10] ''Malware categories,'' https://developers.google.com/android/play-protect/phacategories, 2023.

[11] ''Cyber attacks on Android devices on the rise,'' https://www.gdatasoftware.com/blog/2018/11/31255-cyber-attacks-on-android-devices-on-the-rise, 2018.

[12] ''What is phishing?'' 2023. [Online]. Available: https://www.microsoft.com/en-us/security/business/security-101/what-is-phishing

[13] ''What is a botnet?'' 2023. [Online]. Available: https://www.kaspersky.com/resource-center/threats/botnet-attacks

[14] M. Stamp, *Information Security: Principles and Practice.* Wiley, 2011.

[15] IBM, ‘‘What is random forest?’’ https://www.ibm.com/topics/random-forest.

[16] X. Xing, X. Jin, H. Elahi, H. Jiang, and G. Wang, ‘‘A malware detection approach using autoencoder in deep learning,’’ *IEEE Access*, vol. 10, pp. 25 696--25 706, 2022.

[17] S. Balaji, ‘‘Binary image classifier CNN using TensorFlow?’’ https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697, 2020.

[18] ‘‘What is interpretability?’’ https://it.mathworks.com/discovery/interpretability.html.

[19] P. Hall and N. Gill, *An Introduction to Machine Learning Interpretability*, 2nd ed. O’Reilly Media and H2O, 2019.

[20] D. Cournapeau, ‘‘scikit-learn,’’ https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[21] ‘‘LIME,’’ https://github.com/marcotcr/lime.

[22] ‘‘SHAP,’’ https://github.com/slundberg/shap.

[23] S. Lundberg and S.-I. Lee, ‘‘A unified approach to interpreting model predictions,’’ http://arxiv.org/abs/1705.07874, 2017.

[24] ‘‘ELI5,’’ https://eli5.readthedocs.io/en/latest/index.html, 2016.

[25] Y. Wu, S. Dou, D. Zou, W. Yang, W. Qiang, and H. Jin, ‘‘Contrastive learning for robust android malware familial classification,’’ https://arxiv.org/abs/2107.03799, 2022.

[26] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, ‘‘XAI: Explainable artificial intelligence,’’ *Science Robotics*, vol. 4, no. 37, 2019.

[27] H. Manthena, J. C. Kimmel, M. Abdelsalam, and M. Gupta, ‘‘Analyzing and explaining black-box models for online malware detection,’’ *IEEE Access*, vol. 11, pp. 25 237--25 252, 2023.

[28] F. Charmet, H. Tanuwidjaja, S. Ayoubi, P.-F. Gimenez, Y. Han, H. Jmila, G. Blanc, T. Takahashi, and Z. Zhang, ‘‘Explainable artificial intelligence for cybersecurity: a literature survey,’’ *Annals of Telecommunications*, vol. 77, 2022.

[29] F. Ullah, A. Alsirhani, M. M. Alshahrani, A. Alomari, H. Naeem, and S. A. Shah, ‘‘Explainable malware detection system using transformers-based transfer learning and multi-model visual representation,’’ *Sensors*, vol. 22, no. 18, 2022.

[30] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, ‘‘Explainable AI for android malware detection: Towards understanding why the models perform so well?’’ https://arxiv.org/abs/2209.00812, 2022.

[31] M. Fan, W. Wei, X. Xie, Y. Liu, X. Guan, and T. Liu, ‘‘Can we trust your explanations? sanity checks for interpreters in Android malware analysis,’’ https://arxiv.org/abs/2008.05895.

[32] M. Kinkead, S. Millar, N. McLaughlin, and P. O’Kane, ‘‘Towards explainable cnns for android malware detection,’’ *Procedia Computer Science*, vol. 184, pp. 959--965, 2021, the 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.

[33] G. Severi, J. Meyer, S. Coull, and A. Oprea, ‘‘Explanation-guided backdoor poisoning attacks against malware classifiers,’’ in *30th USENIX Security Symposium*, ser. USENIX Security 21, 2021, pp. 1487--1504.

[34] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, ‘‘Evaluating explanation methods for deep learning in security,’’ https://arxiv.org/abs/1906.02108, 2020.

[35] F. Yan, S. Wen, S. Nepal, C. Paris, and Y. Xiang, ‘‘Explainable machine learning in cybersecurity: A survey,’’ *International Journal of Intelligent Systems*, vol. 37, no. 12, pp. 12 305--12 334, 2022, https://onlinelibrary.wiley.com/doi/abs/10.1002/int.23088.

[36] G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, ‘‘Towards an interpretable deep learning model for mobile malware detection and family identification,’’ *Computers & Security*, vol. 105, pp. 102--198, 2021.

[37] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, ‘‘CADE: Detecting and explaining concept drift samples for security applications,’’ in *30th USENIX Security Symposium*, ser. USENIX Security 21, 2021, pp. 2327--2344.

[38] B. Wu, S. Chen, C. Gao, L. Fan, Y. Liu, W. Wen, and M. R. Lyu, ‘‘Why an Android app is classified as malware? towards malware classification interpretation,’’ https://arxiv.org/abs/2004.11516, 2020.

[39] S. Chen, S. Bateni, S. Grandhi, X. Li, C. Liu, and W. Yang, ‘‘DENAS: Automated rule generation by knowledge extraction from neural networks,’’ in *Proceedings of ESEC/FSE 2020*, 2020, pp. 813--825.

[40] F. Pierazzi, G. Mezzour, Q. Han, M. Colajanni, and V. S. Subrahmanian, ‘‘A data-driven characterization of modern Android spyware,’’ *ACM Transactions on Management Information Systems*, vol. 11, no. 1, pp. 1—--35, 2020.

[41] J. Feichtner and S. Gruber, ‘‘Understanding privacy awareness in android app descriptions using deep learning,’’ in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, 2020, pp. 203--214.

[42] 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.

[43] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm, ‘‘KronoDroid: Time-based hybrid-featured dataset for effective android malware detection and characterization,’’ *Computers & Security*, vol. 110, no. C, pp. 8--14, 2021.

[44] PapersWithCode, ‘‘Class-activation map,’’ https://paperswithcode.com/method/cam.

APPENDIX A

LIME notations

The notation used in LIME mathematical expression is given in Table A.6.

Table A.6: LIME Notation

Notation	Explanation
x	Instance for which explanations on predictions are being made
g	Explanation model
L	Loss that needs to be minimized
f	Trained original machine learning model
$\Omega(g)$	Model Complexity that needs to be reduced
G	Family of possible explanations
π_x	Proximity measure of neighbourhood around instance x

APPENDIX B

LIME Explanations on Other Test Instances

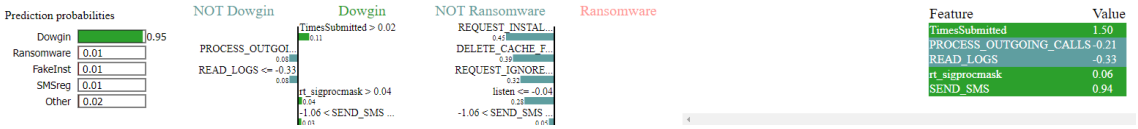

Figure B.43: SVM - LIME explanations on instance 10

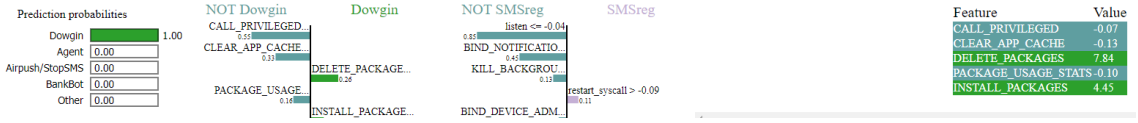

Figure B.44: k-NN - LIME explanations on instance 10

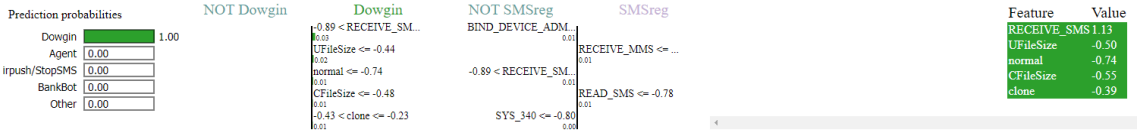

Figure B.45: RF - LIME explanations on instance 10

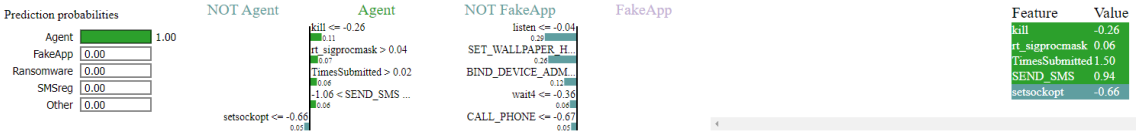

Figure B.46: MLP - LIME explanations on instance 10

APPENDIX C

SHAP Explanations on Other Test Instances

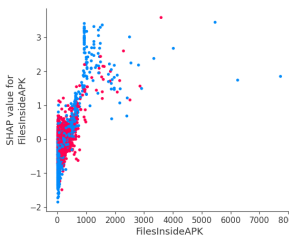

Figure C.47: SVM PDP

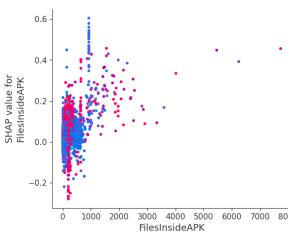

Figure C.48: KNN PDP

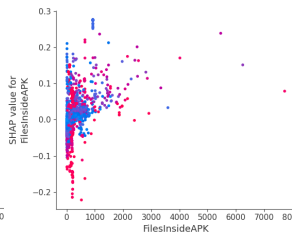

Figure C.49: RF PDP

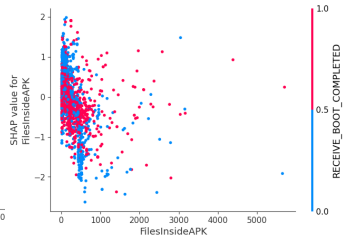

Figure C.50: MLP PDP

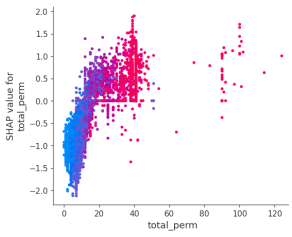

Figure C.51: SVM PDP

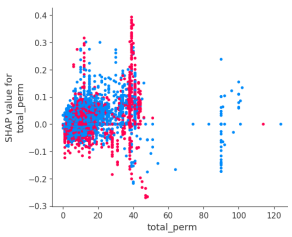

Figure C.52: KNN PDP

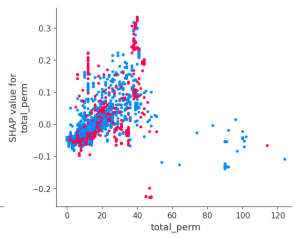

Figure C.53: RF PDP

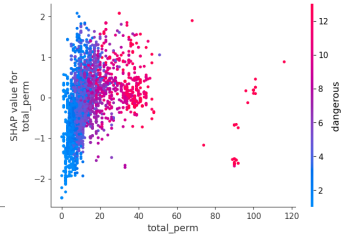

Figure C.54: MLP PDP