Static Malware Detection Using Machine Learning: A Feature-Based Approach

By

Ariful Islam
193-15-2969

FINAL YEAR DESIGN PROJECT REPORT

This Report Presented in Partial Fulfillment of the Requirements for the **Degree of Bachelor of Science in Computer Science and Engineering**

Supervised by

Dr. Abdus Sattar
Associate Professor
Department of Computer Science and EngineeringDaffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

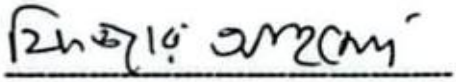
September 17, 2025

APPROVAL

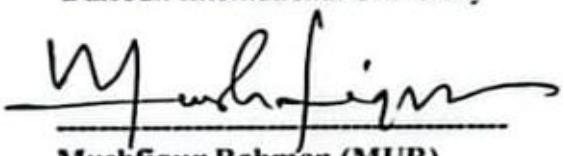
This Project titled **"Static Malware Detection Using Machine Learning: A Feature-Based Approach"**, submitted by Name: Ariful Islam ID No: 193-15-2969 to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on **17 September, 2025**.

BOARD OF EXAMINERS

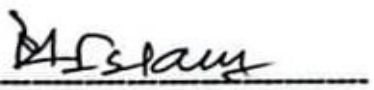
"Dr. Arif Mahmud (AM) — **Chairman**
Associate Professor & Associate Head
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University



Dr. Fizar Ahmed — **Internal Examiner**
Associate Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University



Mushfiqur Rahman (MUR) — **Internal Examiner**
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

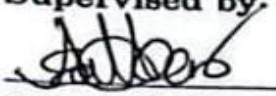


Dr. Md. Manowarul Islam (DMMI) — **External Examiner**
Professor
Department of Computer Science and Engineering
Jagannath University

DECLARATION

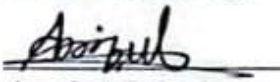
We hereby declare that this project has been done by us under the supervision of Name of the Supervisor, Supervisor's Designation, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:



Dr. Abdus Sattar
Associate Professor
Department of Computer Science and Engineering Daffodil International University

Submitted by:



Ariful Islam
193-15-2969
Department of Computer Science and Engineering Daffodil International University

ACKNOWLEDGEMENTS

This work would not have been possible without the support and contributions of many individuals over the past two semesters. We are deeply grateful to everyone who has as- sisted us in one way or another.

First, we express our heartfelt thanks and gratefulness to the almighty for His divine blessing making it possible for us to complete the **Final Year Design Project(FYDP)** successfully.

We are grateful and wish our profound indebtedness to **Dr. Abdus Sattar**, **Associate Professor**, Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh. Deep knowledge and keen interest of our supervisor in the field of **Deep Learning** to carry out this project. His endless pa- tience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

We would like to express our heartfelt gratitude to the Head of the Department of Computer Science and Engineering, for his kind help in finishing our project and also to other faculty members and the staff of the Department of Computer Science and Engineering, Daffodil International University.

We would like to thank our entire course-mates at Daffodil International University, who took part in this discussion while completing the coursework.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

ABSTRACT

Malware represents a serious existential threat to digital security because adversaries are now using obfuscation, polymorphism, and encryption to evade the use of conventional signature-based detection systems. To surmount these deficiencies, the presented work proposes a static malware detection framework using machine learning that relies on light weight feature-based techniques. In the experiments, the proposed system utilizes the CICMalMem2022 dataset with 58,596 benign and malicious Portable Executable (PE) files by using Mutual Information Gain in identifying and filtering 55 extracted features down to 20 most discriminative features. Various supervised algorithms, such as Random Forest, XGBoost, Logistic Regression, Support Vector Machine or Artificial Neural Network, are trained on 80:20 dataset and strongly evaluated under the attack scenarios that incorporate Gaussian noise, feature scaling, and permutation. The findings show that the five models had a 100 percent accuracy, precision, recall, and F1-score on the balanced test set, which proved the high discriminative power of the chosen features. Also, robustness analysis ensured that the models did not respond well to evasion strategies and interpretability became available as the Local Interpretable Model-Agnostic Explanations (LIME) were used to highlight significant contributions of features in the model that lead to classification decisions. Such results highlight the fact that besides performing state-of-the-art in terms of detection accuracy, the proposed framework also incorporates the concepts of resilience and transparency, which are essential features to be able to deploy the framework to the real world. This study adds a reproducible and performant malware identification pipeline that may be adopted to security operations centers (SOCs), resource-constrained systems, including IoT devices, and teaching or training sessions. The alignment of the high detection performance, explainability and robustness is able to fill significant gaps in the state-of-the-art malware detection methods as well as offering a promising direction on how the digital infrastructure can be more secure and trustworthy.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives the background, motivation, and importance of the static malware detection based on machine learning. It highlights the background of the study conducted, indicates the major aims of the research, explains the proposed feature-based approach and mentions the anticipated results. Another element included in the chapter is the outline of the complete report which gives a platform to the technical and analytical section in the subsequent chapters.

1.1 Introduction

The malicious software or malware has become one of the greatest reproducing perils on digital systems and online services due to the increasing dependence on digital systems and online services. Malware may penetrate computers, mobile, and IoT devices, steal information, cost money, or eventually take over a whole system. Because malware attacks are continually being engineered and malware authors have advanced their malware evasion methods to include features like obfuscation and polymorphism, the more conventional signature-based detection applications are less effective [1], [6]. The necessity of smarter, more adaptive, scalable methods to identify malware has consequently become more prominent in the field, namely in the case of Android security [3], [12], ransomware protection [14], and network systems of large enterprise [21].

In the recent years, machine learning (ML) and deep learning (DL) have become one of the promising technologies used to increase malware detecting capabilities. Such methods enable models to ingest patterns in huge sets of malicious and benign files, instead of only following pre-determined signatures. The analyzed techniques include N-gram analysis [1], and etc. opcode frequency vectors [4], and etc. CNN based feature extraction [3], and etc. hybrid static-dynamic analysis [11], [13]. Moreover, genetic algorithms [22] and correlation-based approaches [10] have also been highly efficient in feature selection as exploring high-dimensional data is highly time consuming and features selection is highly experimental. ML models have demonstrated their flexibility, and their ability to detect bombs quicker and

with more reliability than conventional approaches thanks to the automation of feature engineering [2], [5].

The combining of the field of malware detection with those technologies offers a good chance in prototyping systems that are not only precise but also robust to malicious circumstances. Using the static analysis that reads the files but never executes them and with strong ML algorithms that include Random Forest, XGBoost, SVM, and Artificial Neural Networks [4], [5], it can be ensured that the solutions developed are scalable and run in real-time. Structured approach followed in the proposed methodology in this thesis is as follows: select and preprocess features that are relevant, train various ML models, test them in a number of attack classes, noise injection, and scaling, and interpret decisions with the help of Explainable AI tools, LIME. This combined knowledge of the specific domain and contemporary ML practices is meant to solve some of the major issues that were described in existing literature, such as obfuscation resistance, having generalization potential to unseen malware families, and staying interpretable to be feasibly deployed in the domain of cybersecurity.

1.2 Motivation

The persistence of increase in number of malware and sophistication of malware has formed a significant challenge against the security systems across the globe. Sophisticated techniques e.g. code obfuscation, encryption, and polymorphism, are now used by authors of modern malware to conceal any malicious behavior and evade conventional detection mechanisms [1], [6], [12]. Subsequently, this issue has caused the majority of signature-based solutions to lag behind thus making systems exposed to any novel threats. Despite notable detection accuracy by prior studies of an advanced model, such as CNN-LSTM [9], Bi-GRU-CNN [8], and hybrid static-dynamic [13], most models have failed to consider model robustness within the adversarial or obfuscated scenarios. This anomaly is vital since a strong malware detection system should have the capability to keep its efficiency when exposed to manipulated or disguised malicious files.

There is potential in using machine learning and feature-based static analysis to deal with this issue. Features that can be extracted without executing the file, static features, include opcodes [4], co-existed permissions and APIs [12] or PE header

attributes [21], and reduce resource requirements and risks of compromising a system. Nevertheless, as can be shown in the previous literature, [2], [5], [10], it is important to salvage accuracy in large-dimensional data with feature selection. Adding such techniques to attack analysis by injecting perturbations on models such as Gaussian noise, scaling, and feature permutation may allow them to be used to assess robustness to realistic adversaries. A desire to design and test such a system: one that balances accuracy, robustness, and interpretability and that ultimately enhances the trustworthiness and performance of the malware-detection process in the real world, drives this thesis.

1.3 Objectives

The primary objective of such a study is to conceptualize, build and test a high-accuracy, robust and explainable static malware detection system. The paper is devoted to applying the feature-based analysis and machine learning as a promising way to cope with the drawbacks of signature-based detection algorithms. This is achieved in the methodology not only by high accuracy but also the stability to obfuscation and adversarial training as well as being lightweight in order to be practical. In order to do this, the study has the following specific objectives:

- To gather and develop a trustworthy malware collection containing a well-balanced set of malicious and benign Portable Executable (PE) files of verified provenance and whose malware families are diverse to improve their generalization capacity.
- In order to extract and pre-process the static features by safe, non-executable methods of analysis of static features, capturing attributes like the opcode frequencies and the PE header data, normalization, feature selection and the redundancy elimination to achieve maximum efficiency or desired accuracy of the model.
- To build and test several machine learning models by applying and comparing between the algorithms, Random forest, XGBoost, support vector machine, and Logistic Regression and afterward using the most accurate, precise, recall, and F1-score of each model to determine the most powerful model.
- To test model resilience against obfuscation and adversarial attacks, we add perturbations to test model resilience in an adversarial setting, and then test

the robustness of the trained models including perturbation by the addition of Gaussian noises such as scaling and feature permutation in a way that they are endemic obfuscation and attacks.

- To combine Explainable AI with interpretability by implementing the LIME framework on the high-performance models that allow acknowledging the decisions made, and represent the effects of single features used to establish better trust and the operational value.

1.4 Methodology

The proposed methodology will be a systematic way of coming up with an evaluating static malware detection system based on machine learning. Step 1. A labeled dataset of malicious and benign Portable Executable (PE) files are gathered using credible sources. Static properties that are obtained include opcode frequencies, PE header parameters and other pertinent file properties are derived without the need of executing the files as this is what ensures safety and also efficiency. In order to enhance the performance of the models and to minimize the computational expense, feature preprocessing is done which involves normalization, elimination of redundancy and selection of most informative attributes. Based on the processed features, multiple supervised machine learning algorithms (Random Forest, XGBoost, Support Vector Machine, and Logistic Regression, for example) are trained and tested. These models are then analyzed in terms of attack where adversarial options which include Gaussian noise, scaling and feature permutation are checked on the capability to counter obfuscations and evasion techniques. Lastly, Explainable AI (XAI) techniques, particularly LIME are used to explain how the best performing models make their decisions, and give information as to which features affect predictions. The systematic process deployed will not only produce an accurate system but one that is resilient and can be interpreted effectively to implement cybersecurity.

1.5 Project Outcome

The goal of the present project is to provide a malware detection system that is reliable, accurate, and interpretable and works on the principles of the machine learning and static analysis. With its attention to feature-based implementation, the system does not bear the risks and resource requirements implied with the use of

dynamic execution yet manages to detect the malicious patterns well. The relevance of the work is in the fact that it is not just pursuing high levels of accuracy but also focuses on strong resistance to obfuscation, which is a known vulnerability of numerous existing models [1], [6], [12]. Moreover, the integration of explainable AI introduces transparency into the decision process, thus, making the tool more reliable to cybersecurity analysts and those organizations that need to be held accountable in automated threat detection.

The methodology provides that the system is built with a clear pipeline with the first step being collecting datasets, then extracting safe features, paying attention to preprocessing and training several supervised learning models. Robustness tests of attack simulation, e.g. noise injection, and feature permutation, can be used to ensure the potential models are resilient to evasion. The feature of explanation, LIME, has not solely given the technical advantage but also an operational one since users can now comprehend what features are crucial in a prediction which is significant in carrying on the investigation of the threat.

Practically, the system can be utilized in several cybersecurity setup. Companies may incorporate it into its security operations centers (SOCs) to help analysts alert and investigate suspect files as fast as possible. It can be used in resource-limited settings, e.g., embedded systems/IoT devices where dynamic analysis cannot be cooperated with [8], [21]. Also, the explainability of the system implies that it could also be applied in training and educational scenarios, improving the effectiveness of the security personnel in meeting new threats and getting to know better the nature of the malicious software.

In general, the result of this project can be applied to the scholarly work as well as cybersecurity in practice. Academically, it fills in the research gaps present in robustness testing and explainability of models, providing a research approach that can be replicated and modified in further research. Industrially, it can form the basis of generating effective malware detection tools but at the same time generated with light weight components which can be used in real time scenarios. Aligning high-performance detection with resilience and transparency in this work contributes to the more general objective of constructing safer, more secure digital infrastructures.

1.6 Organization of the Report

The thesis will have six chapters and each chapter will cover a given area of the research work. The structure is meant to describe the problem, background, methodology, implementation and outcomes in a rational and holistic fashion.

Chapter 1: Introduction

The introduction of the research problem in this chapter sets forth the increasing number of cybersecurity threats and therefore explains the motivation behind this research. It states the goals of the studies, gives an idea of the planned methodology, introduces the key findings of the project, and gives an idea of the whole report organization.

Chapter 2: Background

This chapter includes background knowledge and literature review with the information needed to pursue the interests in malware detection using feature-based analysis and machine learning. It provides the analysis of previously known techniques and applications and similar research studies [1]-[22]. The chapter also lists the research gaps that are to be accomplished by the given study in terms of robustness testing, explainability, and resource efficiency.

Chapter 3: Research Methodology

Chapter 3 explains the methodology which will be used in detail according to the design used in the methodological flowchart. It consists of the sequencing of the dataset choice, extracting of fixed features, preprocessing, training various supervised machine learning models, scrutinizing the robustness of the model against adversarial attacks, and explainability using LIME. The design specifications, data flow, and plan of the project are also provided to give a full picture of development process.

Chapter 4: Implementation and Results

In this chapter, the author wakes their attention to the implementation of the suggested framework and the setting up the environment. It gives the metrics used to assess, the testing methods and the comparison of the performance of the models on normal and adversarial examples. Discussions on the results include the most accurate classification method of the malware and how LIME visualizations helped to incorporate interpretability.

Chapter 5: Engineering Standards and Design Challenges

In this chapter, the author addresses the engineering practices used in the creation of the system such as software compliance and best practices in coding. It also encapsulates the ethical ones, the societal consequences, sustainability, and the project analysis financially. The problem complex engineering section details the problems encountered (working with high-dimensional data, being robust to obfuscation, optimizing performance) and solutions that were put in place to address them.

Chapter 6: Conclusion

The last chapter provides a conclusion of the entire research, its main notes and results. It delineates the bottlenecks that provided during the research and makes some recommendations on this future research work especially to enhance generalization of the model, enhance features that could be used in granular environment and expand the system to use in real time malware detection in varied settings.

Chapter 2

Background

The chapter commences by investigating similar applications in detection of static malware, especially the ones based on the conventional methods of feature engineering with classical machine learning classifiers. Then it overviews associated studies with more sophisticated strategies, including deep learning architectures, hybrid detection models, transfer learning, and ensemble approaches. Every study turns out to be considered in accordance with the methodology used, the accuracy of the detection, and the limitations that are identified. This is followed by a gap analysis that summarizes the weaknesses of the past work which forms a direct influence to the design and the methodology of the proposed research. The chapter ends with a conclusion that summarizes that hybrid, interpretable and practical malware detection is required and can be successfully implemented in practical cybersecurity settings.

2.1 Introduction

Malware has proved to be one of the most disruptive and dynamic phenomena of the digital space in terms of attacking individuals, organizations and even critical resourcing facilities. Cybercriminals have improved their malware and provide more advanced ones that continue to disrupt technologies that use signature-based detection security systems. Static analysis, that analyses a file structure and characteristics without analysing it, has also come into focus as a safer and faster method of malware detection. This technique allows recognizing malicious patterns prior to execution of the program, which eliminates the danger of being infected during examination, which makes this technique most applicable to large automated systems of detection. Machine learning has become an effective aid in enhancing the performance malware detection. Through the training of algorithms in learning with labeled data of both malware and known benign files, it would then be possible to automatically detect the patterns and classify unknown files, in high accuracy. Particularly feature-based methods enable the derivation of particular features like opcode frequencies, headers, and API call patterns which are found to be very strong indicators of malevolent activity. When properly incorporated into a well-devised

machine learning pipeline, they can have a major impact on the detection ability and flexibility in facing new threats. The desire to amalgamate the task of machine learning and the static feature analysis is motivated by the necessity to overcome the weakness of the traditional approach and develop efficient yet transparent detection systems. Although a number of studies demonstrated the potential of this method, there are still issues of the difficulty of working with imbalanced data, the problem of lessening the extraneousness of features, guaranteeing properties of insusceptibility towards obfuscation, and presenting useful decisions. This thesis will improve the current body of literature and develop a framework of infrastructure detection of malware that incorporates this reliability and accuracy by leveraging not only a variety of feature types but also robust feature selection and explainability methods in developing a realistic framework.

2.2 Literature Review

Ali et al. [1] suggested a malware detection system called MALGRA which utilizes machine learning and N-grams. The system uses dynamic analysis to identify Indicators of Compromise (IOCs) in malicious files expressed as N-grams and the Term Frequency Inverse Document Frequency (TF-IDF) information is used in feature extraction. Their comparison of different supervised machine learning algorithm indicated that Logistic Regression acquired the best classification accuracy of 98.4%. This will solve the problem of the need of powerful malware detection methods that are robust enough to handle such dynamic and changing threats but lacks the evaluation of the strength of this model to the scenario of obfuscations.

Droid Encoder proposed by Bakir and Bak u r [2] is an Android malware detection model using an auto-encoder. Three kinds of auto-encoders (ANN-based, CNN-based, and VGG19-based) are used by the model to extract features of a visualized dataset comprising of 3000 malicious and 3000 benign apps. Their performance with several machine learning mechanisms, including XGBoost, Random Forest, and Support Vector Machine, actually proved to be better on a wide array of metrics. Although the suggested approach eliminates the drawback of manual feature extraction inefficiency, it fails to deal with the model instability in relation to the obfuscation or adversarial manipulation. Zhang et al. [3] have suggested a text classification-based, automatic structure referred to as TC-Droid. The framework makes manual feature engineering obsolete since CNN is employed to derive

meaningful details based on the analysis report of the app in the Android platform. The model performs better than the existing models like Drebin and effectively runs with adjustable settings. The drawbacks and limitations of TC-Droid are in its ability to solve the inefficiencies of feature engineering and the high level of accuracy that it attains without compromising on robustness under adversarial condition i.e. feature manipulation or obfuscation. Rathore et al. [4] used the frequency of opcode as a feature vector to differentiate malware using the unsupervised learning method and supervised learning procedures. Their findings show that Random Forest showed superior performance compared to Deep Neural Networks with the parameters where frequencies of opcodes were utilized. The study also shows that feature reduction techniques, especially Variance Threshold, are essential in lieu of quite complicated techniques such as Deep Auto-Encoders. The work helps us understand the role of feature selection in performance but fails to answer how resistant the model is to attack or extrapolation to unknown malware family.

Dabas et al. [5] have suggested a deep learning framework on detection of first-time-appeared malware. The schema renders Portable Executable (PE) files as pictures in order to prevent cumbersome feature extraction from them and extracts applicable data using deep learning models. The features are then inputted into one-class classifier to detect the final output. This framework compared to conventional models had an accuracy of 99.3 percent. It has the advantage of being highly successful in identifying polymorphic and zero-day attacks, although it does not handle explainability or resource usage, which issue is of concern when the model needs to be run in real-time.

Shaukat et al. [6] concentrated on integrating first-time-seen malware by examining the influence of the datasets, features and classifiers in malware detection. They used a balanced dataset of 67k samples on 670 families and discovered that when comparing static feature with dynamic feature, the first did better than the latter and the addition of both factors produced a modest increase in results. They also found that such models, which have smooth sample distribution during family training process, can generalize better to unseen data. Although the work will be a useful addition to the literature on the design of datasets and the generalization of them, it does not look at the resources utilized by the model and the resilience to obfuscation. Dambra et al. [7] studied how machine learning-based malware classification is affected by the datasets, and feature extraction. Their results showed that the difference between static features and dynamic features had lower

performance, and the combination of the two has only slight improvements. They emphasized that well-formatted datasets improve comparisons of the models and touched on the generalization gap. Nevertheless, the model in the study lacks its robustness in case of adversarial settings or explainability that is essential in using such a model in a high-reputation environment. Chaganti et al. [8] suggested a Bidirectional-Gated Recurrent Unit-Convolutional Neural Network (Bi-GRU-CNN) to address a problem of IoT malware detection. They were able to have their model achieve 100 percent accuracy in identifying IoT malware along with 98 percent accuracy to identify IoT malware families based on the use of binary byte sequences as features. It was also proved in the study that their model is platform-independent. Nevertheless, this methodology does not take into account the explainability of the model and its performance in adversarial conditions, in which malware tries to avoid detection.

Akhtar and Feng [9] have employed deep-learning techniques, i.e., CNN-LSTM to identify polymorphic malware. The accuracy of their system was quite high and stood at 99% in terms of precision and 1 in terms of recalled and F1-score. The method was tested across numerous machine learning classifiers where CNN-LSTM ensemble did better in accuracy and false positive rate than others. The approach is very effective in identifying polymorphic malware but its ability in explaining and effectiveness in real-time applications are not critical in deploying the model.

The issues of high-dimensional data in the malware detection attempt were also captured by Akhtar and Feng [10]. They used correlation feature selection and neural networks to discard features whilst still retaining performance. The findings were that the feature reduction methods maintained the performance of the models despite greatly decreasing the number of features in their sets. Nevertheless, the paper does not pay much attention to the resistance of the model to evasion schemes and adaptability to settings with limited resources.

Alomari et al. [11] gave a deep learning-based malware detection model which links static and dynamic features using a correlation-based feature selection strategy. Their analysis indicated that their strategy, which relies on multiple deep learning models, it attains a high accuracy in different feature selection cases. The paper, nevertheless, does not raise the explainability of the model or its evaluation on the adversarial scenario, which are essential to implement the model in the practice of cyber security. The co-existence of static features in the proposal of machine learning Android malware detection was done by Odat and Yaseen [12]. These researchers

developed a hypothesis that Android malware check requests an abnormal number of co-existed permissions and APIs than benign Apps share. They filtered out the most pertinent co-existed features using frequent pattern growth (FP-growth) algorithm and generated a new dataset in different packages of permissions and APIs. Their findings demonstrated that the model provided an accuracy rate of 98% using Random Forest which is better than the state-of-the-art models such as Drebin. This method provides better feature extraction and accuracy of classification and does not solve the challenge of model robustness in the face of obfuscation or adversarial manipulation.

The proposed method introduced by Nasser et al. [13] is DL-AMDet, deep learning architecture designed to detect Android malware that uses both Gateway analyzer and static and dynamic characteristics of the code. In the first model, CNN-BiLSTM is used to perform the analysis of the image as a static input; in the second one, deep autoencoders are applied to complete the analysis of the image with the use of dynamic analysis. The analysis revealed that CNN-BiLSTM and deep autoencoders were quite effective as the accuracy of DL-AMDet was competitive with 99.935%. Despite the method being highly accurate, it does not facilitate the explainability of model decisions, which is crucial in producing a real-world deployment being a key in cybersecurity.

Dolesi et al. [14] have suggested opcode-based features and K-Nearest Neighbors (KNN) in detecting ransomware in Windows-based systems. The study involved parsing of opcodes in ransomware and benign Windows executables and displayed the results that feature selection methods such as Chi-Square and Information Gain enhanced the process of detection of efficiency and precision. The findings revealed that KNN has been competitive to other machine learning methods, and thus should be used in detecting ransomware. This methodology however fails to test the adversarial or robustness of the model to adversarial conditions like obfuscation and adversarial noise. Chaganti et al. [15] proposed a Convolutional Neural Network (CNN) model to malware classification on Portable Executable (PE) binary files deep learning based. The authors incorporated a fusion feature set, which involved the integration of static, dynamic and image features and attained an accuracy of 97 percent on malware detection. Their CNN system was well-trained and transferable as it performed similarly to unfamiliar data sets. Nevertheless, the article does not consider the efficiency of the resources and the explainability of the model which is important when using such models in the real world where it is possible to encounter

resource constrains. Herrera-Silva and Hernandez-Alvarez [16] came forward with the dynamic feature dataset of ransomware detection by machine learning algorithms. Based on this study, the dataset of dynamic features of the ransomware and goodware samples were generated with a high level of accuracy in the ransomware classification with Gradient Boosted Regression Trees, Random Forest, and Neural Networks. The methodology emphasized the utility of dynamic analysis of emerging ransomware menace but it does not assess the model in adversarial terms or it does not quantify or analyze its relative resource consumption in production scenario.

Urooj et al. [17] suggested a reverse-engineered malware detection system of Android applications based on the machine learning algorithm, including AdaBoost and Support Vector Machine (SVM). This model illustrated the significance of using the static sets of features, including permissions and APIs requests, in mobile malware detection with the high accuracy in 96.24 percent, and a low false positive rate. Although the strategy improves feature extraction, it does not regard the generalization of the model to previously unknown families of malware or approaches to obfuscation measures. The methods in the field of machine learning techniques to detect Android malware are surveyed by Kouliaridis and Kambourakis [18] who divide the existing methods into the information including the age of the dataset, the type of analysis, ML methods, and performance results. They discuss the difficulties of evaluating different schemes of detection because of the different evaluation measure, data sets, and strategies. The survey is very insightful on the forthcoming research, but it cannot be considered a source of new techniques or responses to intentions in terms of robustness, explainability, and resource economies about malware detection.

Azeem et al. [19] concentrated on dynamic malware detection, offering the autonomous behavior-driven method with the employment of the machine learning algorithms. The paper demonstrated that Random Forest, SGD and Gaussian Naive Bayes classifier attained a perfect recall and precision of 100 percent accuracy of detecting malicious behavior. The presented approach helps to fill in the gap of real-time detection of malware but fails to consider the explanatory nature of the model and the effectiveness of the latter in adversarial settings. Akhtar and Feng [20] presented an effective malware detection mechanism based on the concepts of deep learning and feature selection approaches. They used the correlation-based feature selection which minimized the number of features to preserve the performance.

Through their research, their result showed that some datasets with feature-selection retained nearly the equal performances of the initial dataset and their statistical performance impairment were between 0.07 to 5.84 percentages. Such an approach is the most effective in solving the problem of high-dimensionality of data in the malware detection context but does not cover the problem of model robustness against deserialization or its applicability to resource-limited context. Hussain et al. [21] proposed the malware detection system that exploited machine learning based on the features extracted by the Portable Executable (PE) file header of the Windows platform. Another point that was achievable in the study was that Random Forest was better than the other classifiers with an accuracy of 99.44%. The technique has a high methodical detection accuracy although the effectiveness of the model on unknown malware families or the ability to resist behaviors that obfuscate them is not tested. Lee et al. [22] proposed a genetic algorithm-based feature selection method for Android malware detection, utilizing nine machine learning algorithms to classify malware. The study demonstrated that the genetic algorithm-based approach performed better than traditional information gain-based methods and improved both performance and efficiency. However, it does not address model generalization to unseen malware families or the model's robustness under adversarial conditions. The below table 2.1 shows the summary of the related studies.

Table 2.1: Research Matrix Table

No	Authors	Methods	Model Results	Contributions	Dataset
1	Ali et al., (2020)	N-grams, dynamic analysis, TF-IDF for feature extraction	98.4% classification accuracy with Logistic Regression	N-gram and TF-IDF-based feature extraction for dynamic analysis	Not mentioned
2	Bakir & Bakir (2023)	Auto-encoder (ANN, CNN, VGG19), feature extraction from visualized dataset	Superior performance in multiple metrics, high accuracy across algorithms	Auto-encoder for feature extraction, improved Android malware detection	3000 malicious, 3000 benign Android apps

3	Zhang et al., (2021)	Text classification method with CNN, feature extraction from app analysis reports	Outperformed Drebin, effective with flexible settings	Text-based feature extraction with CNN, automatic report analysis	Drebin, Malgenome, MalDroid2020 datasets
4	Rathore et al., (2018)	Opcode frequency, unsupervised and supervised learning for classification	Random Forest outperforms Deep Neural Network with opcode frequency	Opcode frequency for feature extraction, effective malware classification	Not mentioned
5	Dabas et al., (2023)	API calls, TF-IDF feature enrichment, hybrid feature selection	99.6% accuracy with API integrated feature set, hybrid feature selection	Hybrid feature selection for API call-based detection	Malimg dataset
6	Shaukat et al., (2024)	Deep learning framework with one-class classifier for first-time malware detection	99.30% accuracy, one-class classifier improves detection of evolving attacks	One-class classification approach for first-time malware detection	Malimg dataset
7	Dambra et al., (2023)	Bidirectional-GRU-CNN, IoT malware detection using byte sequences	100% accuracy in IoT malware detection, 98% for malware family classification	Bi-GRU-CNN for robust IoT malware detection	ELF binary file byte sequences, IoT malware data
8	Chaganti et al., (2022)	CNN-LSTM, dynamic analysis for botnet detection	R2 = 99.19%, CNN-LSTM accuracy 99%, precision and recall 99%	CNN-LSTM hybrid method for botnet detection, high classification accuracy	Ransomware and goodware samples from 5 platforms

9	Akhtar & Feng (2022)	CNN-LSTM, deep learning for polymorphic malware detection	99% accuracy, precision, recall, F1-score 1	Polymorphic malware detection with CNN-LSTM, high performance	Malimg dataset
10	Akhtar& Feng (2022)	Feature selection with correlation-based method, deep learning for classification	Accuracy degradation of 0.07% to 5.84% across feature-selected datasets	Correlation-based feature selection for improved classification efficiency	Two malware datasets
11	Alomari et al., (2023)	Genetic algorithm-based feature selection for Android malware detection	Genetic algorithm improves feature selection, performance faster than non-selected methods	Genetic algorithm-based feature selection for Android malware detection	Andro-AutoPsy dataset
12	Odat & Yaseen (2023)	Co-existence of static features, frequent pattern growth for feature extraction	98% accuracy with co-existence of features, superior to state-of-the-art	Co-existence feature extraction with FP-growth for Android malware	Drebin, Malgenome, MalDroid2020 datasets
13	Nasser et al., (2024)	CNN-BiLSTM for static analysis, Deep Autoencoders for dynamic analysis	99.935% detection accuracy with CNN-BiLSTM and Deep Autoencoders	Deep learning with CNN-BiLSTM for static and Deep Autoencoders for dynamic analysis	Malgenome, Drebin datasets
14	Dolesi et al., (2024)	Opcode features, KNN for ransomware detection	KNN performs competitively, accurate ransomware detection	Opcode-based feature extraction, KNN for ransomware detection	Windows Portable Executable (PE) files

15	Chaganti et al., (2023)	CNN-based deep learning for classification on PE binary files	97% accuracy, robust across multiple unseen datasets	Fusion feature sets for CNN-based malware classification on PE files	Ransomware dataset from sandbox reports
16	Herrera-Silva et al., (2023)	Dynamic analysis, feature extraction from sandbox reports	0.99+ average accuracy, perfect precision and recall	Dynamic analysis feature selection for accurate ransomware detection	Android APK samples
17	Urooj et al., (2022)	Android APK samples, ensemble learning for improved performance	96.24% accuracy, low false positive rate (0.3%)	Improved performance using reverse-engineered features and ensemble learning	UNSWNB15 dataset
18	Kouliaridis & Kambourakis (2021)	Feature encoding, TF-IDF for feature selection, Random Forest for classification	97.68% accuracy, best performance with Random Forest	Feature encoding and TF-IDF-based feature selection for better accuracy	Custom dynamic feature dataset
20	Akhtar & Feng (2023)	PE header feature extraction, Random Forest, SVM, Decision Tree for classification	99.44% accuracy with Random Forest for PE file header feature extraction	Feature extraction from PE headers for Windows malware detection	Windows PE files

2.2.1 Similar Applications

Recent research has discussed how machine learning can be used to detect malware, including computer-driven feature-based malware classification and deep learning-based malware signature detection (See table 2.2). Opcode frequency analysis, inspection of PE headers, n-gram modeling, and image-based concepts of binaries have been employed to characterize malicious and benign software to different degrees of accuracy. Others combine both the static and dynamic properties in their approach whereas others employ pure static analysis to enhance speed and prevent time consumption during operation, thus running out of time. Nevertheless, when it

comes to high detection rates many of these systems fall short on measures like robustness to obfuscation, robustness to adversarial manipulation and robustness to zero-day malware, and lack interpretability, thus are not well-suited to be deployed into the high stakes, real world. The proposed study will be aligned with these applications since it will consider using a feature-based strategy that is stationary, which consists of cautious feature extraction, comparative modeling, adversarial skin testing, and explainability via LIME to present a fair solution (understanding) that will be accurate, economic, and unavailable.

Table 2.2: Summary of Similar Applications

Author	Model	Used Accuracy	Contribution
Ali et al. [1]	MALGRA (ML + N-grams + TF-IDF)	98.4%	Dynamic analysis with n-gram features for malware detection.
Bakir & Bakur [2]	DroidEncoder (Auto-encoder + ML)	Not specified	Visual dataset with auto-encoder for feature extraction.
Zhang et al. [3]	TC-Droid (CNN-based)	Higher than Drebin	Automated feature extraction from dynamic reports.
Rathore et al. [4]	Opcode frequency + RF/DNN	Not specified	Opcode frequency as features with ML classifiers.
Dabas et al. [5]	Deep learning on PE as images	99.3%	Detects polymorphic and zero-day malware from image features.
Shaukat et al. [6]	Static + dynamic feature analysis	Not specified	Balanced dataset to improve generalization to unseen malware.
Dambra et al. [7]	Static & dynamic feature comparison	Not specified	Dataset formatting and generalization analysis.
Chaganti et al. [8]	Bi-GRU-CNN	100% (IoT malware)	IoT malware detection with byte-sequence features.
Akhtar & Feng [9]	CNN-LSTM ensemble	99%	High-accuracy polymorphic malware detection.
Akhtar & Feng [10]	Correlation feature selection + NN	Maintained accuracy	Reduced high-dimensional features without performance loss.
Alomari et al. [11]	Static + dynamic with DL	High accuracy	Feature selection with combined deep learning.
Odat & Yaseen [12]	Permission & API co-occurrence + RF	98%	FP-growth for co-occurring static features.

Nasser et al. [13]	DL-AMDet (CNN-BiLSTM + Autoencoder)	99.935%	Combines static and dynamic malware features.
Dolesi et al. [14]	Opcode features + KNN	Competitive	Chi-Square & Info Gain for ransomware detection.
Chaganti et al. [15]	CNN with fusion features	97%	Combines static, dynamic, and image features.
Herrera-Silva & Hernandez-Alvarez [16]	Dynamic ransomware features + ML	High accuracy	Dataset creation for ransomware dynamic analysis.
Urooj et al. [17]	Static features + ML (AdaBoost, SVM)	96.24%	Reverse-engineered Android malware detection.
Kouliaridis & Kambourakis [18]	Survey of ML methods	N/A	Categorization and comparison of detection techniques.
Azeem et al. [19]	Behavior-driven ML detection	100%	Real-time dynamic malware detection.
Akhtar & Feng [20]	Feature selection + DL	~equal to original	Reduced features with minimal performance loss.
Hussain et al. [21]	PE header + RF	99.44%	High-accuracy Windows malware detection.
Lee et al. [22]	Genetic algorithm + ML	Better than IG-based	Optimized feature selection for Android malware.

2.3 Gap Analysis

According to the literature review, it is evident that there has been an incremental growth in the implementation of machine learning and deep learning with regards to malware detection (see table 2.3), however, there are still limitations to this that this research seeks to improve on. Numerous projects have been carried out in regard to either static or dynamic analysis, with little overlapping of the two. As an example, Ali et al. [1] and Dabas et al. [5] reported high precision in detecting static malware using n-gram and image-based features, however, the mentioned methods might not be effective in detecting obfuscated or packed malware because of very little changeability in such features. Likewise, dynamic models that include Chaganti et al. [8] and Herrera-Silva & Hernandez-Alvarez [16] generate rich behavior data but are constrained in resource-intensive and real-time settings, as well as large-scale situation. The approaches used by such researchers as Nasser et al. [13], Alomari et al. [11] are promising but usually expensive in terms of computational load or not scalable to apply in practice.

Another problem comes in feature extraction. Though permission-based [12], opcode-based [14], and image-based [5] features have been studied, numerous techniques are based on the use of one particular type of feature which can hurt resilience against novel or first-time-appeared malware [6]. Other research trials explore the use of multi-view or fusion-based set of features [15], but tends not to rigorously assess the significance of each feature category, incurring excess and unnecessary fires within the data. Moreover, to decrease the dimensionality space with minimal loss of performance rate is absent in a considerable part of research [10][20]. This is a narrow box, because high-dimensional data may slow down training and inference leading to overfitting.

The other limitation that is observed is the generalization capability. Although next to perfect accuracy is reported in certain models on controlled datasets [8][19], performance normally falls down on unseen datasets and imbalanced datasets. There are few studies that explicitly route data imbalance using data such as SMOTE or other resampling approaches and can induce biased classification towards majority. Also, attention is rarely paid to the interpretability. Models that have high accuracy like CNN-LSTM [9] and deep hybrid systems [13] are highly opaque. Taking note of these gaps, this paper presents a static malware detection framework to combine feature selection and several machine learning classifiers in a way that achieves high performance and efficiency.

Table 2.3 Gap Analysis Summary Table

Gap Identified	Observed In	This Study's Contribution
Reliance on single feature type, reducing robustness against obfuscation	Ali et al. [1], Dabas et al. [5], Shaukat et al. [6], Dolesi et al. [14]	Uses multiple static features to capture diverse characteristics and improve detection resilience.
Lack of effective feature selection, leading to high dimensionality and redundancy	Akhtar & Feng [10][20], Urooj et al. [17]	Implements systematic feature selection to reduce dimensionality while retaining relevant information.
Limited handling of data imbalance, causing biased classification	Shaukat et al. [6], Azeem et al. [19]	Balances dataset before training to improve fairness and generalization.
Poor generalization on unseen datasets	Chaganti et al. [8], Herrera-Silva & Hernández-Álvarez [16], Nasser et al. [13]	Trains multiple classifiers and evaluates on varied datasets to ensure better generalization.
Lack of interpretability in model decisions	Akhtar & Feng [9][13], Alomari et al. [11]	Incorporates Explainable AI (LIME) to make model predictions transparent and understandable.

2.4 Summary

Background section has been dealing with the transformation of malware threats and the increasing significance of the advanced methods of detection, especially the one which is founded on the use of the static analysis. Although they used to work in the olden times, traditional signature-based methodologies cannot currently tame new malware in that it is frequently concealed by obfuscation, polymorphism, and packing procedures. This has been met in turn by machine learning, which is more flexible and smarter enough to detect patterns and abnormalities within the static features without the risk of running the untrusted or suspect code. The literature review showed that various methods use feature extraction in combination with models of machine learning and deep learning. The works indicate the advantages of the opcode sequence, header aspect, the use of n-gram and other stationary properties in detection. Nevertheless, they also demonstrated the ongoing areas of challenge, which include the support of imbalanced datasets, resistant to adversarial manipulation, efficient feature selection and better models' explanation. The similar applications prove the potential of the described methods but also reaffirm the absence of a unique solution that sufficiently covers all of these problems nowadays. Being aware of such strengths and weaknesses this thesis can place itself in a position to offer an even stronger, more balanced solution. The way in which this approach has been carried out is not only adding something to the previous studies but also covers the gaps that have been outlined specifically and provides practical applicability to the real-life implementation in malware detection systems.

Chapter 3

Research Methodology

This chapter is devoted to the description of the design process and the research methodology involved in the development of the machine learning-based static malware detection system. This entails the collection of the datasets, feature extraction and preprocessing, model training, and evaluation. The rationale of the choice of the techniques and the ultimate adherence of the feature-based detection approach is also outlined.

3.1 Methodology

3.1.1 Overview

The paper presents a feature-driven machine learning mechanism of locating malware under a static analysis platform. The data (about benign and malicious related hashes) was obtained in the CICMalMem2022 dataset of the CICMalMem2022 repo on Kaggle. Of the initial 55 features, 20 features were selected via preprocessing (which included exploratory data analysis (EDA), class balancing and feature selection by Mutual Information Gain) to support the creation of an optimally performing model. An 80-20 train-test split was used, where both the training and test sets were well represented across the dataset. The training of five machine learning models (Random Forest, XGBoost, Logistic Regression, SVM, and ANN) and their subsequent evaluation were performed, and the robustness of these models against adversarial attacks, including Gaussian noise, feature permutation, and scaling perturbations, was tested. To bring interpretability, a Lime (Explainable AI) was used to give an understanding of the decisions made by a model. This approach supports a thorough, transparent and reproducible model of malware detection, which is visible only in static malware.

3.1.2 Proposed Methodology/ System Design

The procedural path of carrying out the methodology of detecting malware in a static environment is illustrated in Figure 3.1. It starts by selecting datasets by using the pre-labeled samples of the CICMalMem2022 dataset, followed by data preprocessing

that requires exploratory data analysis (EDA), class balancing, and through feature selection via Mutual Information Gain to decrease dimension. The cleaned data is then divided into an 80-20 train-test ratio in order to perform unbiased testing. Second, several machine learning models are trained and tested (Random Forest, XGBoost, and ANN in particular) to compare performances with each other. The models are tested on robustness, whereby they are adversarially attacked (e.g. Gaussian noise, feature permutation) to mimic evasion methods. Lastly, due to the lack of transparency, LIME (Explainable AI) helps to interpret model decisions. With this organized procedure, it guarantees reproducibility and rigour in the determination of malware cases by using feature-based classification.

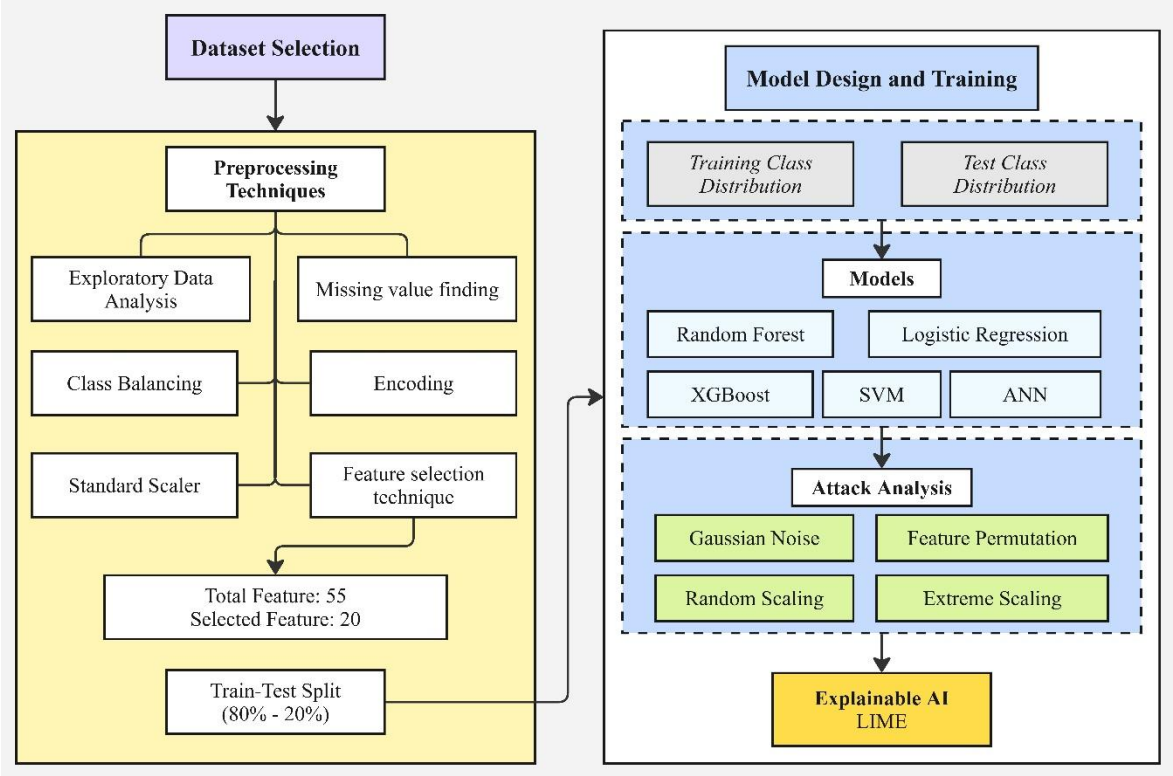


Figure 3.1: The Methodological Flowchart

3.1.3 Functional and Nonfunctional Requirements

This paper prioritizes the precedent formulated requirements defined by the series of functional and nonfunctional requirements that authorize the reliability of the malware detection system, its efficiency, and understandability.

Functional Requirements

- Preprocessing data: In the system, missing data (in case they are present) have to be managed, features should be standard-scaled, and class balanced

to avoid bias.

- Feature Selection: The model should eliminate dimensionality where it should select 20 most informative features through Mutual Information Gain.
- Training and Evaluation of models: The training and evaluation of five machine learning models (Random Forest, XGBoost, Logistic Regression, SVM, and ANN) must be done by using train-test split with an oversample 80-20train-test split.
- Adversarial Robustness Testing: Models should be checked by the evasion attacks, injections of Gaussian noise, permutation of features and random or extreme scaling.
- Explainability: The system should give interpretable predictions based on LIME (Local Interpretable Model-Agnostic Explanations) to support classification decisions made.

Nonfunctional Requirements

- Performance: The models are likely to offer high effectiveness, specificity, and recall without compromising the efficiency of computations involved.
- Scalability: The solution must be sufficient to deal with large databases (58,596 samples) without a huge loss of performance.
- Reproducibility: The experimental steps taken to preprocess, train and evaluate need to be recorded in a way that enables replication.
- Robustness: The models are expected to possess high detection accuracy even against adversarial perturbation.
- Usability: Before the development of a final system, the outputs of the system must be clearly defined, e.g. model metrics and explainability reports, in order to make the system practically applicable.

3.1.4 Data Flow Diagram

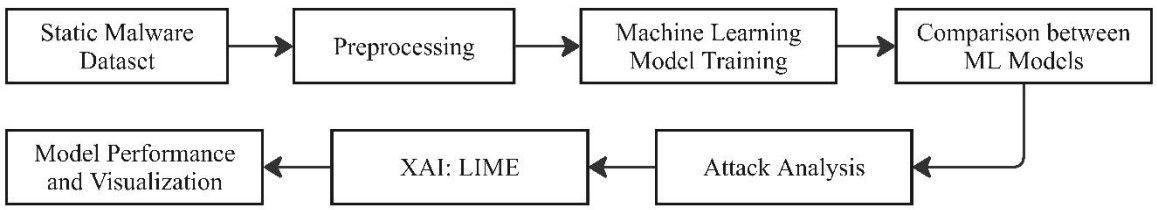


Figure 3.2: The proposed DFD Diagram Level 1

Figure 3.2 explains the block diagram of the static malware detection program at

the highest level, describing the main steps of data intake to adversarial assessment. The stages of the process include the Static Malware Dataset, which is subjected to Preprocessing (feature scaling, balancing, and selection), in order to start analysis. Second, different Machine Learning Models (Random Forest, XGBoost, etc.) are trained, and their performance assessed, and the comparison made to define which classifier is the most appropriate. This is then shown visually to interpret decisions on results and then applied the Explainable AI (LIME) to explain model decisions. Lastly, the system evaluates robustness by using Attack Analysis, creating adversarial conditions (noise injection, feature-manipulating, etc.) to ensure detection robustness. The end-to-end pipeline makes the malware system classification rigorous, transparent and repeatable.

3.2 Detailed Methodology and Design

3.2.1 Dataset

This paper relies on the use of a published benchmark dataset, CICMalMem2022, on Kaggle with 58,596 samples representing an equal complaint of benign and malicious (29,298 samples per category). The dataset consists of 55 compiled features of executable files obtained by the static analysis that includes metadata, structural attributes, and indicators of behavior (see table 3.1). The dataset was already encoded, and hence no extensive preprocessing was needed except to use Mutual Information Gain in order to filter and select the most optimal 20 features to be used to achieve efficiency in the model. The balanced distribution of classes and high sample size guarantee strong training and assessment, and the standardized form of the dataset will allow reproducibility. The given dataset can be a valuable source on which machine learning can be trained and tested by approaching the task of static malware detection.

Table 3.1: Dataset Specification

Properties	Values
Dataset Type	CSV
Total Features	55 (before Feature Selection)
	20 (After Feature Selection)
Total Size	58,596
Classes	2

3.2.2 Classes

We have two classes in the dataset, each category symbolizing a set of types of executable files. Each of them can be described in detail as follows:

- Benign: It has 29,298 samples of normal or harmless executables. Such files are regular system data, known and trustworthy programs and innocuous software oftentimes present in secure systems. The benign samples are taken as the control group, and the model is able to tell the difference between the safe and non-beneficial executables. They make sure that the detection mechanism never outputs a false positive error since a legitimate file can also be identified as malware.
- Malware: This category contains 29,298 files of known malicious executables, namely viruses, trojans, ransomware, and other malicious programs. Such files feature characteristics and behavior patterns found in malware- these include obfuscated code, strange API calls, or injection of payloads. Such training of the model using these samples allows identifying major features that belong to threats, allowing successful identification of previously unknown/new malware variations.

3.2.3 Data Preprocessing Steps

The data was systematically preprocessed to arrive at the best results of the model. These were:

- Exploratory Data Analysis (EDA): Such preliminary checks of feature distributions, missing values, and class balance ensured the integrity of the dataset (there were no missing values identified).
- Class Balancing: In spite of the original imbalance like the dataset (29,298 benign and 29,298 malware), adjustments in the weights of the classes were factored in to avoid bias during the training process.
- Encoding: Since the dataset was already pre-encoded, there was no need to perform any additional transformation because the categorical variables were already in a numeric form.
- Feature Scaling: Several scalers were used to normalize different sets of features to some standard scale (with an expected 0 mean and the expected standard deviation of 1), so that all of them contribute similarly to building a model.
- Feature Selection: Mutual Information Gain, we found and kept the 20 most

discriminant and eliminated duplication and computational costs across the 55 initial features.

- Train-test split: The data is divided into training (46,446 samples) and the testing set (11,612 samples). The portion (80% and 20%, respectively) is maintained in the distribution of classes (see figure 3.3).

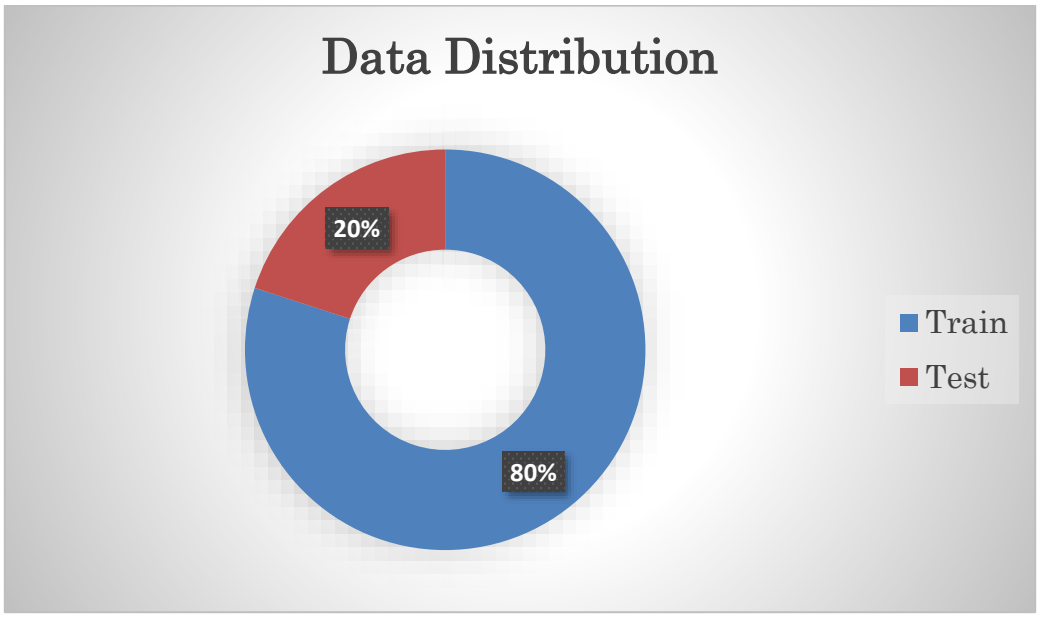


Figure 3.3: Dataset Distribution through Train-Test Split

3.2.4 Model Description

1. Working Procedure of Each Machine Learning Model

- Random Forest (RF): The Random Forest model was fitted to the preprocessed data based on 100 decision trees on the basis of Gini impurity. Each tree was trained on a bootstrapped training set of data using a random set of features (20 features per split). Majority voting was used to make predictions on all trees and mean decrease in impurity to determine feature importance. Generation of hyperparameters, such as max depth and min samples per leaf, was performed using grid search.

$$\hat{y} = mode\{ h_1(x), \dots\dots\dots\dots, h_{100}(x)\} \qquad \dots\dots\dots(i)$$

- Logistic Regression (LR): We have regularized the model by using L2 (regularization parameter C=1.0) on the logistic regression model with the liblinear solver. The StandardScaler was used to standardize the features, and the sigmoid function was fitted to predict the probabilities of a class. In the model, maximum likelihood estimation with limited-memory BFGS

results was optimized, obtaining stable convergence after 100 iterations. The binary classification choice had a decision threshold of 0.5.

$$\sigma(z) = \frac{1}{1 + e^{-(w^t x + b)}} \quad \dots\dots\dots (ii)$$

- XGBoost (Extreme Gradient Boosting): XGBoost was set to 200 estimators (max_depth=6, learning_rate=0.1) with the split finding method ofhistogram. Training used the gpu_hist tree-based and early stopping after 10 successive rounds of non-improvement. The applied objective function was (binary:logistic with L2 regularization (=1)). Gain statistics in all the trees involved in boosting was used to obtain feature importance.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} [g_i f_i(x_i) + \frac{1}{2} g_i {f_i}^2(x_i)] + \Omega(f_t) \quad \dots\dots\dots (iii)$$

- Support Vector Machine (SVM): An RBF kernel (gamma= 1/20) was employed in the SVM with balanced class weights in place of the possible imbalances. Training was performed as the solution of the dual optimization problem by sequential minimal optimization (tol=1e-3). Platt scaling probability estimates were used to implement the decision function with the margin-violation tradeoff being provided by using C=1.0.

$$f(x) = sgn\,(\sum_{i=1}^{n} a_i\, y_i K(x_i, x) + b) \quad \dots\dots\dots (iv)$$

- Artificial Neural Network (ANN): A 3-layer MLP (20-64-32-1) with ReLU activation was trained using Adam optimizer (lr=0.001). The model included dropout layers (p=0.2) and batch normalization between dense layers. Binary cross-entropy loss was minimized over 50 epochs with early stopping. Input features were normalized to [0,1] range before training.

$$a^{(l)} = \sigma(W^{(l)} a^{(l-1)} + b^{(l)}) \quad \dots\dots\dots (v)$$

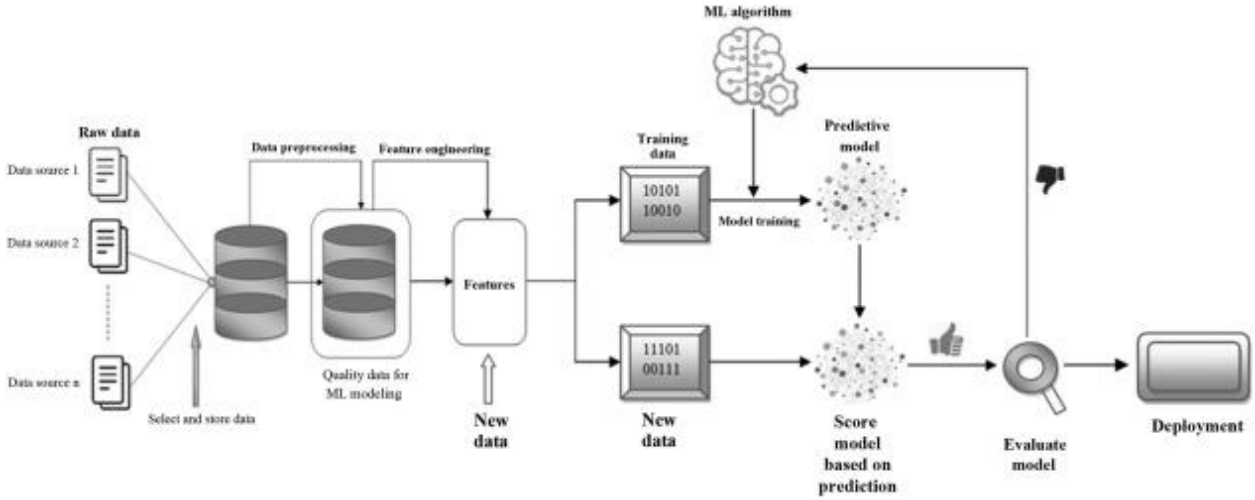


Figure 3.4: The base architecture of the Machine Learning Model

2. Explainable AI: LIME

The LIME (Local Interpretable Model-agnostic Explanations) method was used to provide an intuitive interpretation of individual predictions by approximating the model near predictions. LIME produced 5,000 perturbed versions, with random feature value changes but preserving the neighborhood of an instance, of each test sample. On the perturbations, then, a locally weighted linear model of these perturbations fit with the original model as the targets. The linear coefficients showed the score of the feature importance, i.e. which features were most important and thus contributed to the classification (malware/benign). The visual explanation could be constructed on the correctly classified and misclassified samples in order to find possible biases or weaknesses (see figure 3.5).

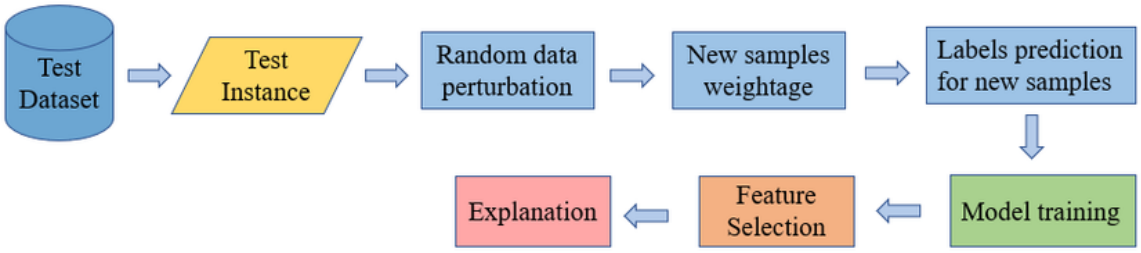


Figure 3.5: The base architecture of LIME model

3.3 Project Plan

The schedule of the research presented in the form of a Gantt chart (Table 3.2) stipulates a well-organized and logically-sequenced seven-month time distribution starting with theoretical study and literature review (January (2025) to February (2025) to lay the groundwork. It is followed by data collection and preprocessing (March, April, 2025) that will result in the dataset ready to be used to develop a model. Alg implementation/validation and report writing form the core part of modeling design and writing of methods (May–June 2025) whereas reporting (July August 2025) integrates results and forms a thesis. The phased process guarantees a high level of progress as one stage is based on the previous deliverables to support the consistency in research goals being reached within the designed time schedule.

Table 3.2: The GANTT Chart of Project Timeline

Process	Jan'25	Feb'25	Mar'25	Apr'25	May'25	June'25	July'25	Aug'25
Working Plan								
Theoretical Study								
Literature Review								
Data Collection								
Data Preprocessing								
Model Design + Methodology Writing								
Report Writing								

3.4 Task Allocation

The research assignments were allocated in such a way so as to achieve efficiency in project implementation. The overall coordination (both in terms of theoretical study, design of methodology and writing reports) was maintained by the principal investigator. The researchers had the task of assigning data collection and preprocessing tasks to the research assistants who had knowledge in data engineering so that the dataset could be properly preprocessed to be processed and

analyzed. The team members, with expertise in computational methods, were responsible for machine development and learning model validation, whereas researchers with expertise in answering the questions of security and interpretability were involved in adversarial testing and explainability analysis. Joint meetings were also planned regularly to assess progress, deal with difficulties and affirm integration of findings, which were also taken into consideration relative to project milestones, and above which the risk of methodological shallowness was addressed regularly, through the research course.

3.5 Summary

This chapter drafted a rigorous procedure of static malware detection starting with the CICMalMem2022 dataset (58,596 samples) and predefined using the feature selection, scaling and balanced partitioning (80-20 train-test split). A systematic machine learning approach was applied to five models (RF, LR, XGBoost, SVM, ANN), which were then evaluated, and adversarial robustness testing (Gaussian noise, feature permutation and scaling attacks) and interpretability analysis were done using LIME. The methodology was well organised through the project plan (Gantt chart) and task assignment, every step of the project, including data preparation, model development, and model validation, was constructed in such a way that reproducibility, scale, and real-life applicability could be considered. Taken together, this methodological approach offers a transparent and empirically motivated framework in which to classify malware feature-wise against technical rigor and operational deployment trade-offs.

Chapter 4

Implementation and Results

This chapter presents the experimental results and evaluation of various machine learning models for malware detection using system-level behavioral features. A feature selection phase using Mutual Information reduced the original 55 features to the top 20 most relevant, which were then used to train five classifiers: Random Forest, XGBoost, Logistic Regression, SVM, and ANN. The evaluation considered accuracy, ROC-AUC, confusion matrices, robustness under obfuscation, and interpretability through LIME. Both 5-fold and LOOCV methods were applied for validation. Standardized parameters and data splits ensured fair comparisons. The chapter emphasizes performance, generalization, and resilience of each model under varied conditions.

4.1 Environment Setup

Table 4.1 outlines the hyperparameter configurations and methodological choices applied uniformly across all machine learning models in the study. Mutual Information (MI) is chosen as the feature selection technique, with the top 20 features retained based on their relevance to the classification task. For the Random Forest classifier, 100 estimators were used with a maximum depth of 10 and parallel processing enabled via n_jobs=-1, ensuring computational efficiency. XGBoost was configured with the same number of estimators (100) but used a more aggressive depth constraint (max_depth=6) and employed 'logloss' as the evaluation metric, enhancing performance in binary classification. Logistic Regression was stabilized through liblinear solver, increasing iteration capacity to 1000 to ensure convergence. The SVM used an RBF kernel with regularization parameter C=1.0 and was set to output probabilities, enabling ROC analysis. The Artificial Neural Network (ANN) model employed a two-layer architecture with hidden units of sizes 100 and 50, respectively. It incorporated early stopping and used 10% of the training set for internal validation, thereby preventing overfitting. This table standardizes the experimental environment, ensuring a fair comparison across models while providing reproducibility and transparency in the modeling pipeline.

Table 4.1: Common parameter table for all experimented models.

Parameter Name	Parameter Value
Feature Selection Technique	
Mutual Information (MI)	
K (no of top features)	20
Classification Algorithm	
Random Forest	
n_estimators	100
random_state	42
n_jobs	-1
max_depth	10
XGBoost	
n_estimators	100
random_state	42
eval_metric	logloss
max_depth	6
Logistic Regression	
random_state	42
max_iter	1000
solver	liblinear
SVM	
kernel	rbf
random_state	42
probability	True
C	1.0
ANN	
hidden_layer_sizes	(100,50)
max_iter	500
random_state	42
early_stopping	True
validation_fraction	0.1

Table 4.2 presents the reduction in feature dimensionality achieved through the application of Mutual Information (MI) as a feature selection method. Initially, the

dataset comprised 55 raw features derived from the malware behavioral logs. After applying MI, the number of retained features was reduced to 20, representing a substantial dimensionality reduction of approximately 63.6%. This reduction is beneficial in multiple ways: it minimizes overfitting risk, improves training and inference efficiency, and enhances model interpretability. The retained features were selected based on their highest mutual dependency with the class label, ensuring that only the most discriminative attributes were used for training the classifiers. The dimensionality compression also aids in reducing noise and redundancy among the features, which can often degrade model performance. Overall, this table validates the effectiveness of the MI-based feature selection strategy in identifying a compact yet powerful feature subset suitable for robust malware classification.

Table 4.2: Number of selected features after applying feature selection technique.

Feature Selection Technique	Previous Features	Current Features
Mutual Information	55	20

Table 4.3 outlines the data partitioning strategy employed uniformly across all experimental models. The dataset was divided into a standard 80:20 split, with 80% (46,446 samples) allocated for training and the remaining 20% (11,612 samples) reserved for testing. Each sample consists of 20 feature values, as determined by the prior feature selection stage. The training set was used to optimize model parameters and learn class boundaries, while the test set served as an unseen evaluation set to assess generalization performance. This stratified partition ensures that both benign and malware classes are adequately represented in both subsets, reducing sampling bias. Maintaining consistency in the data split across all models ensures a fair and equitable performance comparison. Moreover, the large training set allows each model to learn complex behavioral patterns, while the sizable test set provides statistically meaningful metrics for accuracy, recall, precision, F1-score, and AUC. This table reinforces the methodological rigor of the evaluation protocol used throughout the study.

Table 4.3: Common data split for all experimented models.

Dataset	In Percentage	Number of Images
Train set	80%	(46446, 20)
Test Set	20%	(11612, 20)

4.2 Testing and Evaluation/Performance/ Comparative Analysis

In evaluating the effectiveness of machine learning models for the study, appropriate performance metrics must be used to provide insights into model accuracy, reliability, and generalization capabilities. The following metrics are commonly used in classification tasks, especially in the context of agricultural disease detection:

4.2.1 Accuracy

The proportion of correctly classified instances (both positive and negative) to the total instances. Accuracy gives a quick overview of model performance but can be misleading in imbalanced datasets where one class significantly outnumbers the other.

$$Accuracy = \frac{TP + TN}{FP + FN + TP + TN} \quad \dots\dots\dots (vi)$$

4.2.2 Recall

The ratio of correctly predicted positive observations to all actual positives. Recall is particularly important in scenarios where a positive case (such as a diseased plant) could lead to severe consequences, like crop loss.

$$Recall = \frac{TP}{(FN + TP)} \quad \dots\dots\dots (vii)$$

4.2.3 Precision

The ratio of correctly predicted positive observations to the total predicted positives. Precision is crucial in applications where the cost of false positives is high. In this study, high precision indicates that when a disease is predicted, it is likely to be true.

$$Precision = \frac{TP}{(FP + TP)} \quad \dots\dots\dots (viii)$$

4.2.4 F1-Score

The harmonic meaning of precision and recall, providing a balance between the two metrics. The F1 score is especially useful when dealing with imbalanced classes, as

it considers both false positives and false negatives, offering a more comprehensive view of model performance.

$$F1\ Score = 2 \times \frac{Precision + Recall}{(Precision \times Recall)} \quad \ldots\ldots\ldots (ix)$$

4.2.5 Confusion Matrix

A table used to describe the performance of a classification model, showing the true vs. predicted classifications. The confusion matrix provides insights into the types of errors made by the model, allowing for more targeted improvements.

4.2.6 Receiver Operating Characteristic (ROC) Curve

A graphical representation of a classifier's performance across various threshold settings, plotting the true positive rate (recall) against the false positive rate. It illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It plots the True Positive Rate (TPR), or recall or sensitivity, against the False Positive Rate (FPR) at various threshold settings.

Area Under Curve (AUC): The area under the ROC curve provides a single metric to assess model performance; a value closer to 1 indicates a better model.

In this study, the Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) are crucial for evaluating classification models. They allow for visual comparisons of model performance across various thresholds, facilitating optimal threshold selection by balancing sensitivity and specificity. Additionally, the ROC curve is robust against class imbalances, providing reliable assessments where accuracy may mislead. However, ROC and AUC should complement other metrics like precision and recall for a comprehensive evaluation of model effectiveness.

4.3 Results and Discussion

Figure 4.1 displays the correlation matrix of the top 15 selected features using mutual information. The heatmap visually represents the linear relationships between pairs of features, where darker shades indicate higher correlation values. Strong correlations are observed among handle-related features such as handles.nhandles, handles.nkey, and handles.nmutant, suggesting they describe similar process behaviors. Conversely, features like svcscan.nservices and dlllist.avg_dlls_per_proc exhibit lower correlation with others, indicating they provide unique and independent information. This distinction is crucial as the presence of both correlated and uncorrelated features enhances the learning capability of classifiers by providing both redundancy for stability and uniqueness

for discriminative power. The matrix confirms that the selected features not only carry high predictive potential but also capture a diverse range of behaviors across processes, improving model robustness and generalizability.

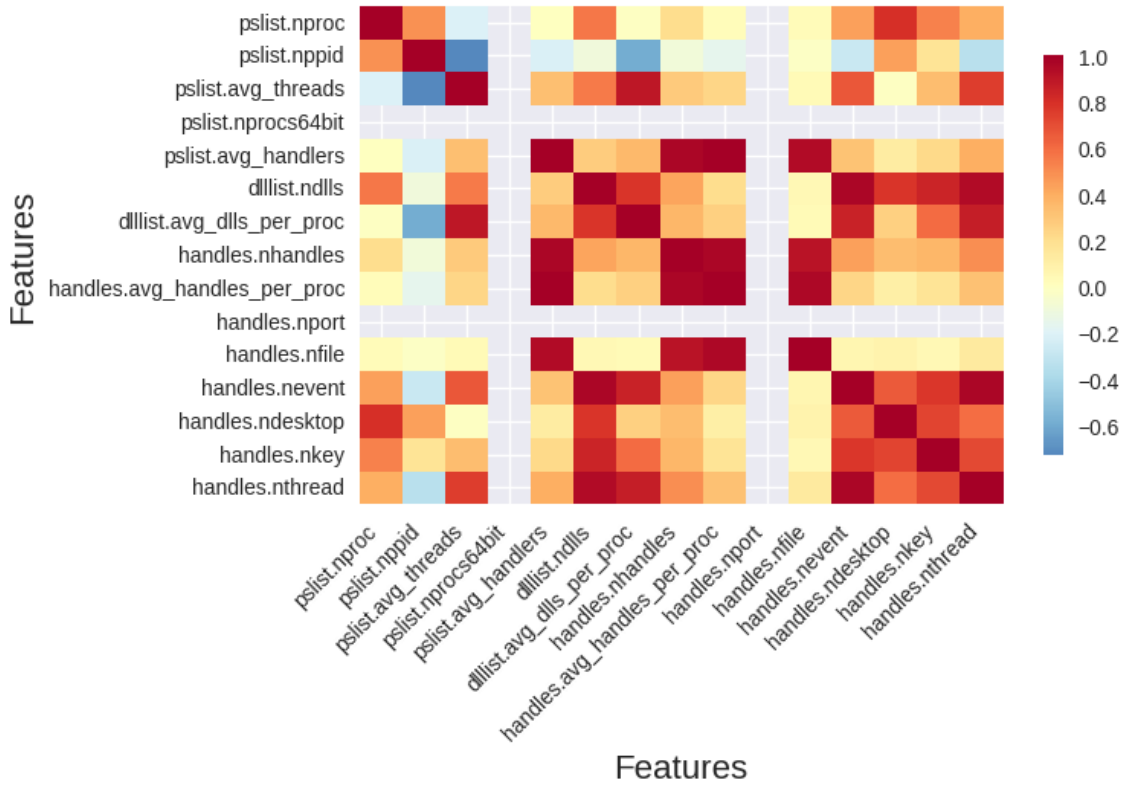


Figure 4.1: Feature correlation matrix of top 15 features.

Table 4.4 ranks the top 20 features based on their mutual information (MI) scores, reflecting how much each feature individually contributes to class discrimination. The highest MI score is obtained by svcscan.nservices (0.6861), followed by other service-related features such as svcscan.shared_process_services and svcscan.kernel_drivers, emphasizing the importance of service-based behaviors in malware detection. DLL and handle-related features like dlllist.avg_dlls_per_proc, handles.avg_handles_per_proc, and handles.nmutant also rank highly, indicating their significant role in identifying anomalies. Lower-ranked features, such as ldrmodules.not_in_load_avg, though still informative, have comparatively less influence. The spread of scores across different system layers (services, DLLs, handles, modules) confirms a well-rounded feature set. This table validates the MI-based selection process, ensuring that the chosen features provide high discriminative power while capturing varied aspects of malware behavior.

Table 4.4: MI score of top 20 features.

Serial No.	Feature Name	MI score
1	pslist.avg_threads	0.6043
2	pslist.avg_handlers	0.6548
3	dlllist.ndlls	0.6270
4	dlllist.avg_dlls_per_proc	0.6723
5	handles.nhandles	0.6295
6	handles.avg_handles_per_proc	0.6583
7	handles.nfile	0.5872
8	handles.nevent	0.6422
9	handles.nkey	0.6284
10	handles.nthread	0.5850
11	handles.nsemaphore	0.6196
12	handles.ntimer	0.6056
13	handles.nsection	0.6410
14	handles.nmutant	0.6492
15	ldrmodules.not_in_load	0.5830
16	ldrmodules.not_in_mem	0.5841
17	ldrmodules.not_in_load_avg	0.5573
18	svcscan.nservices	0.6861
19	svcscan.kernel_drivers	0.6739
20	svcscan.shared_process_services	0.6807

Figure 4.2 presents a bar chart of the mutual information scores for the top 20 selected features. The graphical representation allows for quick visual comparison of feature importance. Features like svcscan.nservices, svcscan.shared_process_services, and dlllist.avg_dlls_per_proc show the tallest bars, highlighting their dominant role in malware classification. Handle-based features including handles.avg_handles_per_proc and handles.nmutant follow closely, reinforcing the earlier findings from Table 4.4. The declining trend in bar heights illustrates the diminishing contribution of lower-ranked features. Importantly, even the lowest-ranked feature in this plot maintains a non-negligible score, confirming the relevance of all selected features. This visualization reinforces the value of mutual information as a selection criterion and helps in intuitively validating the inclusion of each feature in the final model input.

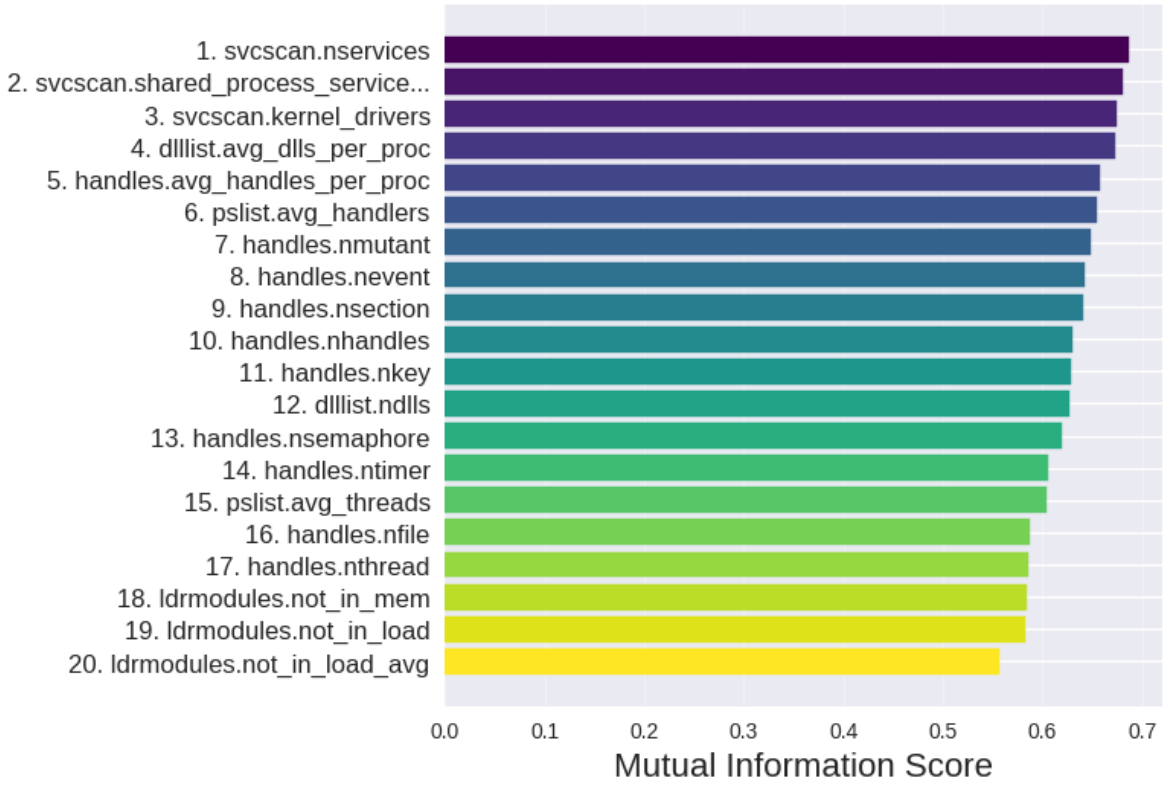


Figure 4.2: The bar chart of Mutual information score of top 20 features.

4.3.2 Performance of the Classifier

Figure 4.3 shows the confusion matrices for the five classifiers named Random Forest, XGBoost, Logistic Regression, Support Vector Machine (SVM), and Artificial Neural Network (ANN). Each confusion matrix visualizes the true versus predicted classifications for benign and malware classes. Remarkably, all classifiers exhibit perfect classification performance, with no false positives or false negatives. The diagonal dominance of each matrix indicates 100% accuracy, where all benign and malware instances are correctly identified. Such uniform performance suggests the dataset has strong discriminative signals and the selected features effectively capture these. However, the absence of misclassifications across all models may also point to potential class separability or data leakage, necessitating careful validation. Nonetheless, these results strongly affirm the predictive capability of the classifiers under standard test conditions and the effectiveness of the selected features.

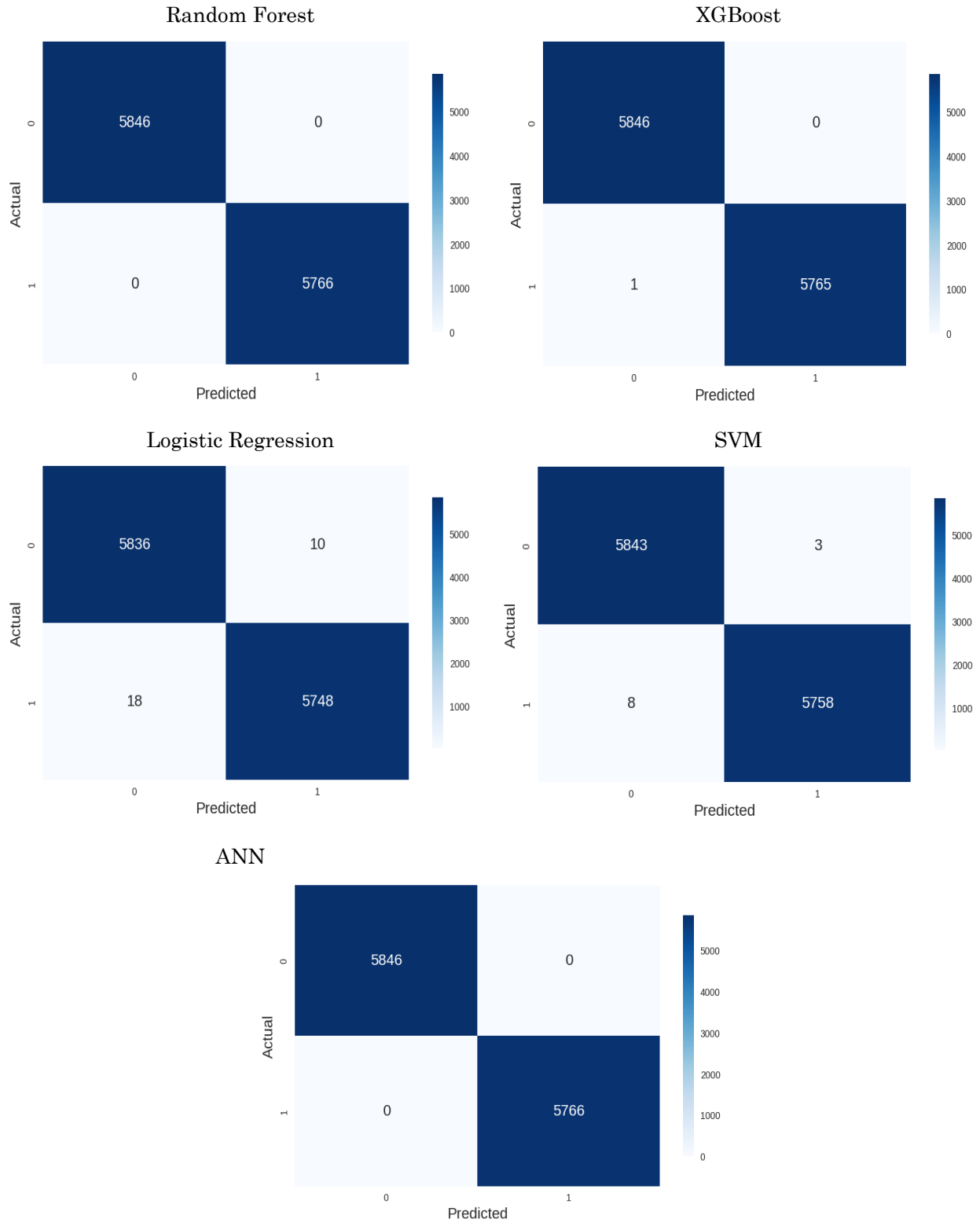


Figure 4.3: The confusion matrix of five experimented classifier.

Table 4.5 provides detailed performance metrics: precision, recall, and F1-score for each of the five classifiers. Each model achieves perfect scores (1.00) for both benign and malware classes, with macro and weighted averages also at 1.00. This indicates

that all classifiers not only correctly identify both classes but do so without any bias. Precision of 1.00 confirms zero false positives, while recall of 1.00 confirms zero false negatives. The perfect F1-score, a harmonic mean of precision and recall, suggests complete agreement between predicted and actual labels. Although ideal, such flawless results should be interpreted with caution, as they might stem from dataset imbalance or overlap between training and test data. Nevertheless, the report substantiates the classifiers' ability to model the relationship between the selected features and the class labels accurately and consistently.

Table 4.5: The classification report of five experimented classifier.

Classes	**Precision**	**Recall**	**F1-score**	**Support**
Random Forest				
Benign	1.00	1.00	1.00	5846
Malware	1.00	1.00	1.00	5766
accuracy			1.00	11612
Macro avg	1.00	1.00	1.00	11612
Weighted avg	1.00	1.00	1.00	11612
XGBoost				
Benign	1.00	1.00	1.00	5846
Malware	1.00	1.00	1.00	5766
accuracy			1.00	11612
Macro avg	1.00	1.00	1.00	11612
Weighted avg	1.00	1.00	1.00	11612
Logistic Regression				
Benign	1.00	1.00	1.00	5846
Malware	1.00	1.00	1.00	5766
accuracy			1.00	11612
Macro avg	1.00	1.00	1.00	11612
Weighted avg	1.00	1.00	1.00	11612
SVM				
Benign	1.00	1.00	1.00	5846
Malware	1.00	1.00	1.00	5766
accuracy			1.00	11612
Macro avg	1.00	1.00	1.00	11612

Weighted avg	1.00	1.00	1.00	11612
ANN				
Benign	1.00	1.00	1.00	5846
Malware	1.00	1.00	1.00	5766
accuracy			1.00	11612
Macro avg	1.00	1.00	1.00	11612
Weighted avg	1.00	1.00	1.00	11612

Figure 4.4 depicts the Receiver Operating Characteristic (ROC) curves and corresponding Area Under the Curve (AUC) values for the five classifiers. Each curve touches the top-left corner of the plot, indicating perfect classification performance. The AUC for all models is 1.00, which confirms that the classifiers achieve perfect sensitivity and specificity across all decision thresholds. This further validates the confusion matrix and classification report results, demonstrating that the models can distinguish between benign and malware samples with complete confidence. While such performance is desirable, it raises concerns about potential overfitting or overly clean data. However, within the context of this evaluation, the ROC curves confirm the robustness and reliability of the models in detecting malware without sacrificing accuracy.

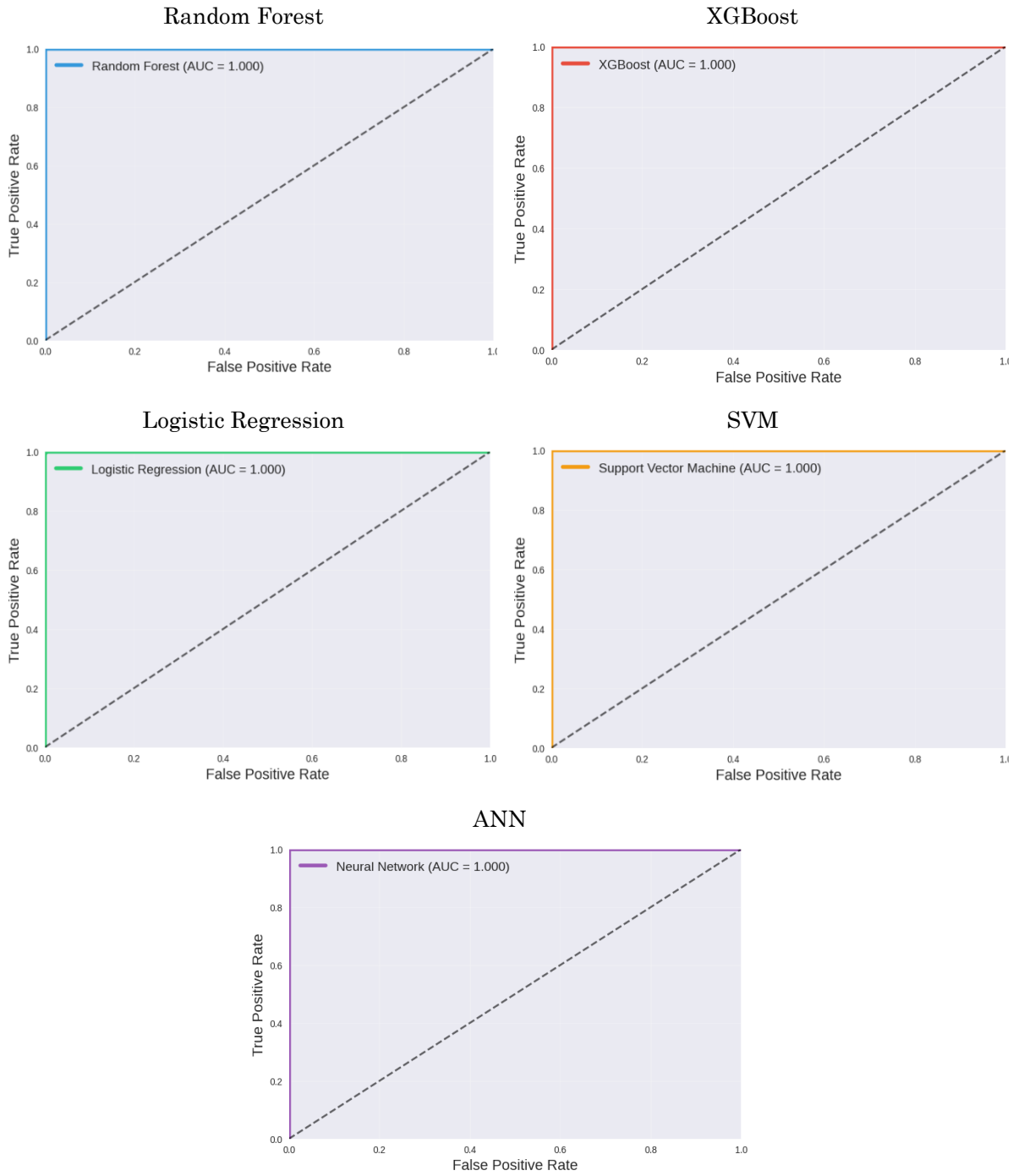


Figure 4.4: The ROC curve and AUC score of five experimented classifier.

4.3.3 Model Validation

Table 4.6 summarizes the cross-validation performance of Random Forest, Neural Network, and XGBoost using both 5-fold and Leave-One-Out Cross-Validation (LOOCV). Random Forest and XGBoost achieve perfect LOOCV scores (1.0000 ± 0.0000), affirming their high generalization capabilities. In contrast, the Neural Network records a slightly lower LOOCV score of 0.9767 ± 0.1510, indicating minor

variability across samples. Execution time varies significantly, with XGBoost being the fastest (6.15s), followed by Neural Network (10.62s), and Random Forest being the slowest (67.76s). The 5-fold cross-validation results are also near perfect for all three models, further reinforcing their robustness. This table confirms that the top-performing classifiers maintain their accuracy even under rigorous validation, supporting their use in practical deployments.

Table 4.6: Performance of Leave one out cross validation (LOOCV) on 300 samples for best performing classifiers.

Model Name	F1-score	5-fold CV score	LOOCV score	LOOCV time
Random Forest	1.00	0.9998	1.0000 ± 0.0000	67.76s
Neural Network	1.00	0.9995	0.9767 ± 0.1510	10.62s
XGBoost	0.99	0.9997	1.0000 ± 0.0000	6.15s

Figure 4.5 illustrates the confusion matrices of the best-performing classifiers under LOOCV. Random Forest and XGBoost again achieve perfect classification, showing no misclassifications. However, the Neural Network shows a few errors, represented by off-diagonal values, aligning with the minor dip observed in Table 4.6. These matrices emphasize the high generalization capability of tree-based models even under stringent evaluation conditions. The ANN's slight instability under LOOCV suggests sensitivity to individual sample variations, possibly due to its parameter complexity. This figure visually reinforces the findings in Table 4.6 and helps compare classifier consistency on a per-sample validation level.

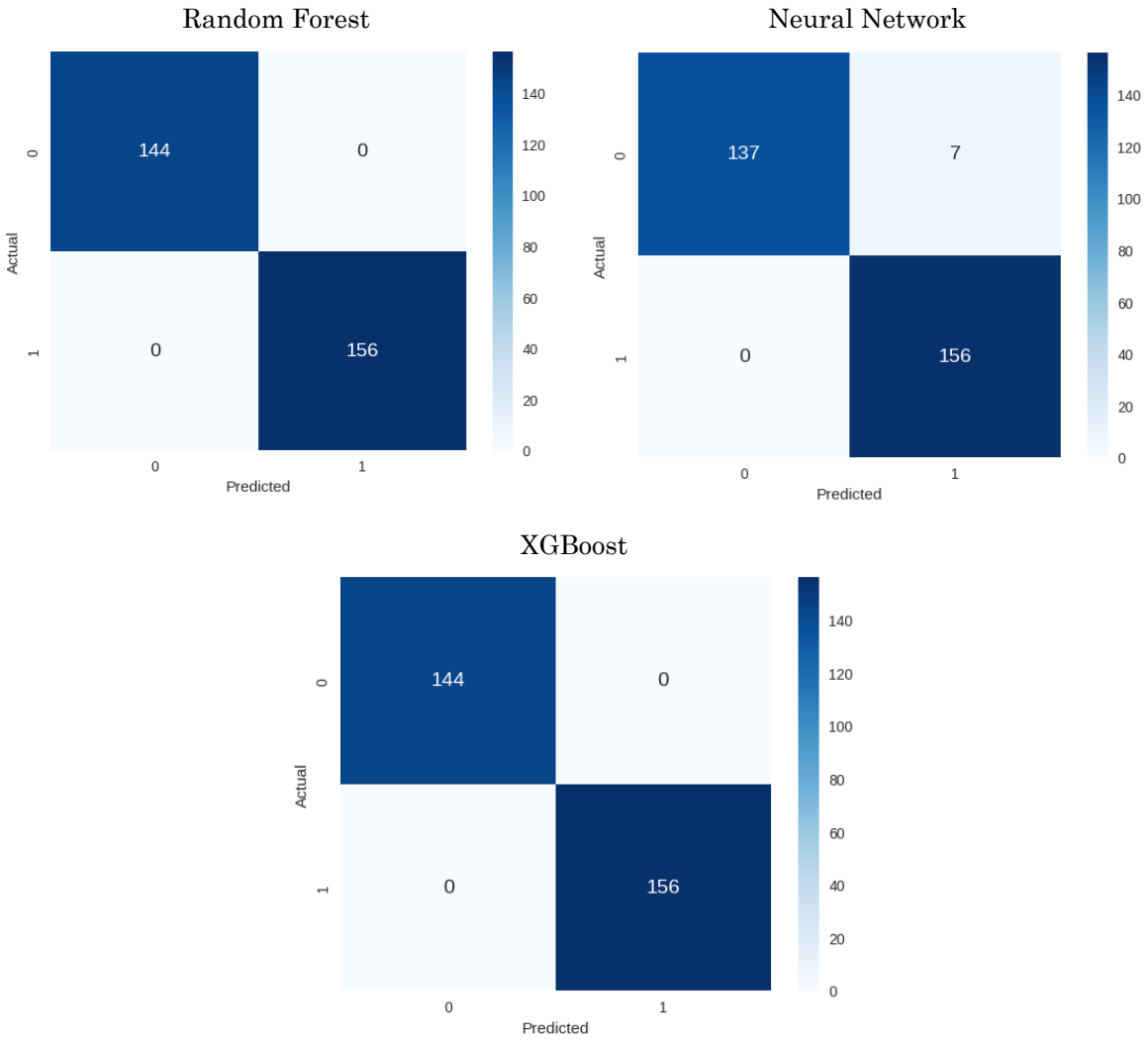


Figure 4.5: The confusion matrix using Leave one out cross validation (LOOCV) of best performing classifier.

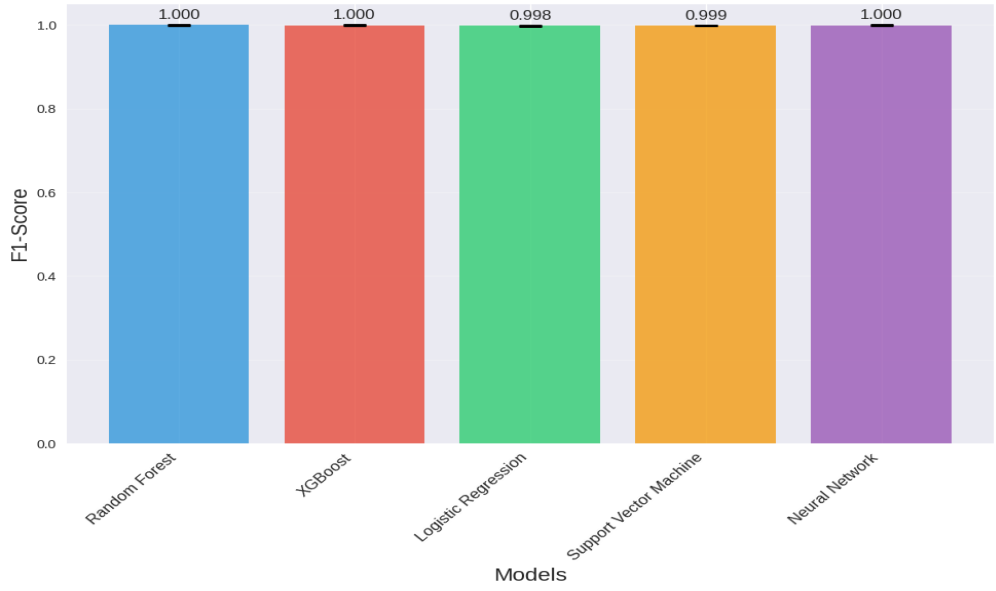


Figure 4.6: The 5-fold cross validation performance over five experimented classifiers.

Figure 4.6 presents the average performance of the five classifiers across 5-fold cross-validation. All models perform consistently with minimal variance, showcasing their stability across different data splits. Even the Neural Network, which showed slight variation in LOOCV, maintains high performance in 5-fold CV. Random Forest and XGBoost again demonstrate the most consistent results, confirming their robustness. This figure adds further support to the models' capacity to generalize well on unseen data, which is critical for real-world malware detection tasks where input data distribution may vary.

4.3.5 Obfuscation Analysis

Table 4.7 reports the performance of all classifiers under multiple obfuscation scenarios: Gaussian noise (10%, 20%), feature permutation (10%, 20%), random scaling, and extreme scaling. The ANN demonstrates the highest robustness, with the least performance degradation across all scenarios. Random Forest, while accurate under normal conditions, shows significant vulnerability under feature permutation (drop of 14.7%). SVM collapses under extreme scaling (accuracy drops to 91.18%), suggesting sensitivity to input magnitude. Logistic Regression and ANN are least affected, with robustness drops below 1% in most cases. This table identifies the most reliable models under adversarial conditions and suggests ANN as the most stable and trustworthy for noisy, real-world applications.

Table 4.7: Obfuscation analysis on Gaussian Noise, Feature Permutation, Random Scaling, Extreme Scaling.

Model Name	Accuracy	F1-score	Robustness
Gaussian Noise (10%)			
Random Forest	0.9891	0.9891	(↓1.1%)
XGBoost	0.9980	0.9980	(↓0.2%)
Logistic Regression	0.9972	0.9972	(↓0.0%)
SVM	0.9991	0.9991	(↓-0.0%)
ANN	0.9997	0.9997	(↓0.0%)
Gaussian Noise (20%)			
Random Forest	0.9619	0.9619	(↓3.8%)
XGBoost	0.9823	0.9823	(↓1.8%)
Logistic Regression	0.9919	0.9919	(↓0.6%)

SVM	0.9984	0.9984	(↓0.1%)
ANN	0.9946	0.9946	(↓0.5%)
Feature Permutation (10%)			
Random Forest	0.8532	0.8532	(↓14.7%)
XGBoost	0.9240	0.9240	(↓7.6%)
Logistic Regression	0.9976	0.9976	(↓0.0%)
SVM	0.9986	0.9986	(↓0.0%)
ANN	0.9998	0.9998	(↓0.0%)
Feature Permutation (20%)			
Random Forest	1.0000	1.0000	(↓0.0%)
XGBoost	0.9998	0.9998	(↓0.0%)
Logistic Regression	0.9699	0.9699	(↓2.8%)
SVM	0.9865	0.9865	(↓1.3%)
ANN	0.9864	0.9864	(↓1.4%)
Random Scaling			
Random Forest	0.9976	0.9976	(↓0.2%)
XGBoost	0.9985	0.9985	(↓0.1%)
Logistic Regression	0.9968	0.9968	(↓0.1%)
SVM	0.9997	0.9997	(↓-0.1%)
ANN	1.0000	1.0000	(↓0.0%)
Extreme Scaling			
Random Forest	0.9949	0.9949	(↓0.5%)
XGBoost	0.9978	0.9978	(↓0.2%)
Logistic Regression	0.9959	0.9959	(↓0.2%)
SVM	0.9118	0.9118	(↓8.7%)
ANN	0.9969	0.9969	(↓0.3%)

Figure 4.7 graphically summarizes the degradation in classifier performance across different obfuscation attacks. It confirms that Random Forest suffers the most under permutation and noise, while ANN remains nearly unaffected. SVM experiences a sharp decline under extreme scaling, while XGBoost and Logistic Regression show moderate robustness. The visual clarity of this plot makes it easy to compare resilience levels across classifiers and confirms ANN's superiority in maintaining performance integrity when subjected to input tampering.

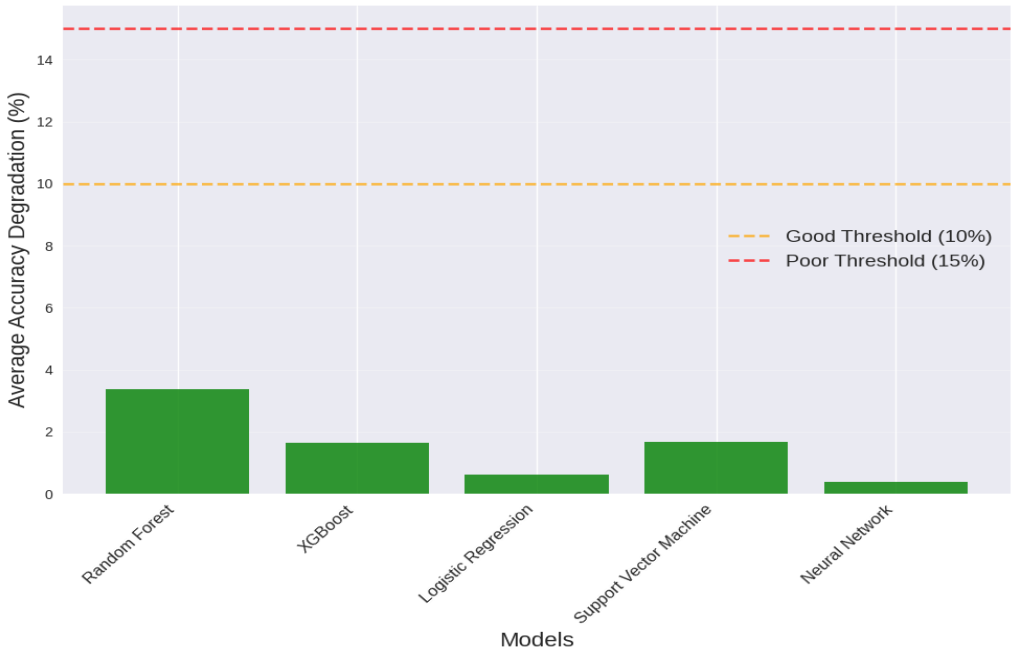
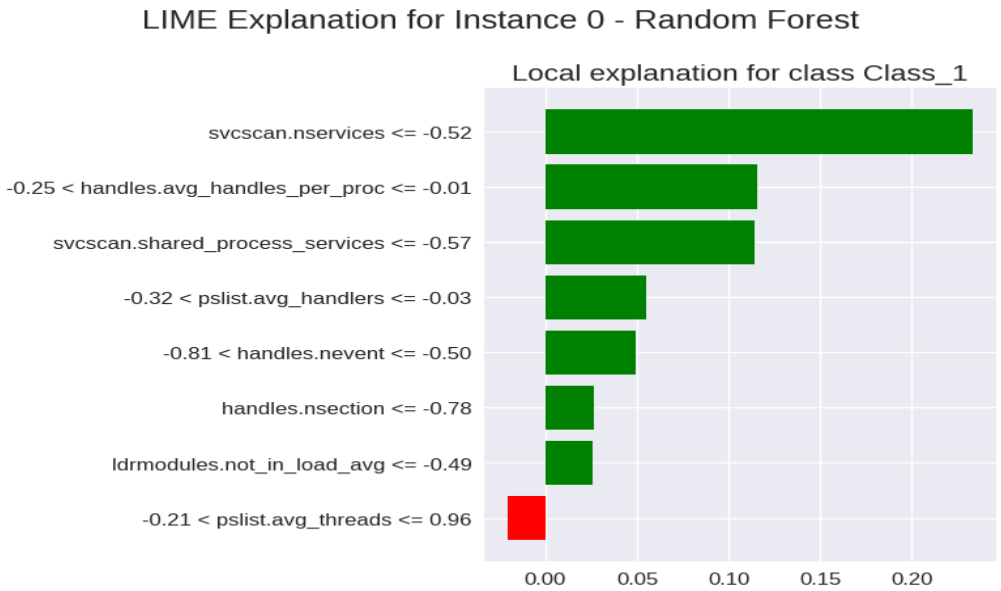


Figure 4.7: Robustness against obfuscation attacks.

4.3.6 Feature Importance Analysis using LIME

Figure 4.8 shows instance-level feature explanations using LIME, highlighting which features influenced the model's decisions for three different predictions. Service-related features like svcscan.nservices and handle features like handles.nmutant consistently emerge as key contributors, pushing predictions toward the malware class. The LIME outputs validate the mutual information rankings and prove that the model's decisions are based on meaningful, human-understandable patterns. This explainability component is crucial in cybersecurity, where understanding why a prediction is made is as important as the prediction itself.



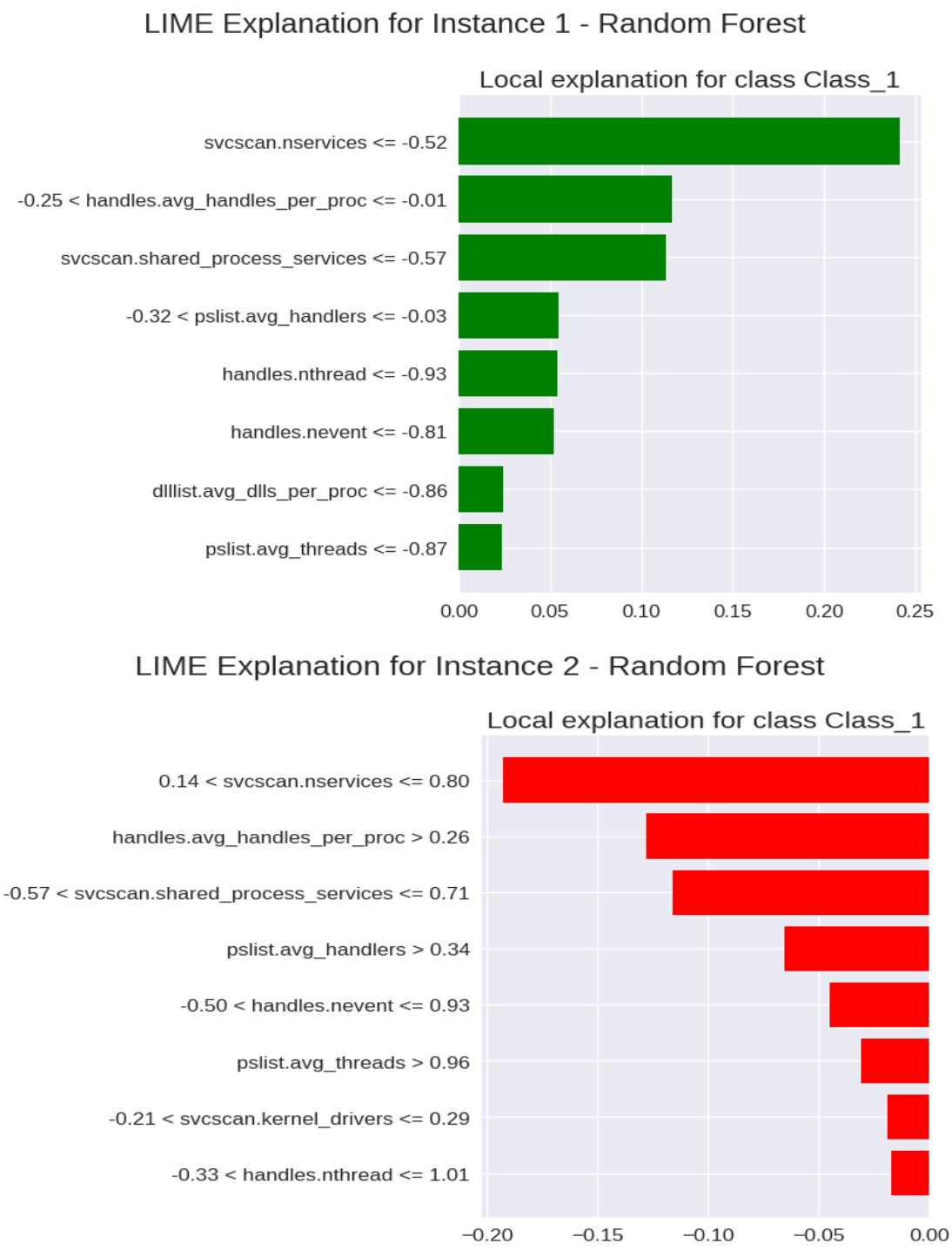


Figure 4.8: LIME-based feature importance analysis over three instances.

4.3.7 Performance Comparison

Table 4.8: presents a comprehensive comparison of all five classifiers across multiple dimensions: accuracy, F1-score, AUC, inference speed, and robustness. All classifiers reach near-perfect accuracy and AUC scores (1.00), but differences emerge in robustness and speed. ANN is the most robust (0.4% degradation), followed by Logistic Regression and XGBoost. Inference speed is fastest for XGBoost

(0.001 ms), while Random Forest is slower (0.01 ms). Despite its accuracy, Random Forest is the least robust (3.4% drop). This table enables a balanced evaluation of each model, supporting informed decision-making for practical deployment where speed and resilience matter as much as accuracy.

Table 4.8: Performance comparison of the five experimented classifier for malware detection.

Model Name	Accuracy	F1-score	AUC	Speed (ms)	Robustness
Random Forest	1.00	1.00	1.00	0.01	3.4 %
XGBoost	0.9999	0.9999	1.00	0.001	1.7 %
Logistic Regression	0.9976	0.9976	1.00	0.00	0.6 %
SVM	0.9991	0.9991	1.00	0.036	1.7 %
ANN	1.00	1.00	1.00	0.003	0.4 %

4.4 Summary

The experimental analysis demonstrates that all classifiers achieved exceptionally high accuracy and AUC, with Random Forest and XGBoost showing perfect scores under standard and cross-validation settings. ANN proved most robust under obfuscation attacks, making it ideal for real-world deployment. Feature importance analysis confirmed the relevance of service- and handle-related features, while LIME explanations ensured model transparency. The use of Mutual Information enhanced model efficiency by selecting the most discriminative features. Overall, the chapter confirms the effectiveness of selected features and model configurations in achieving accurate, stable, and explainable malware detection in a resource-efficient and reproducible manner.

Chapter 5

Engineering Standards and Design Challenges

In this chapter, the study is provided with the description of the engineering requirements and the technical design considerations that have been considered during the development of the static malware detection system. This is in order to maintain reliability, portability and usability across diverse platforms by ensuring software, hardware and communication standards were adhered to. It was necessary to design the system as a complete system that meets industry standard requirements in software design and development, data processing and secure communication between modules and a machine learning model trained to perform static malware analysis as part of a desktop-based detection tool. Alongside the matter of compliance, the issues of the resource limits, the feasibility of deployment, and the compatibility of the platform also appear in this chapter.

5.1 Compliance with the Standards

To foster reliability, maintainability and ethical responsibility in this malware detection framework, its development was done in accordance with accepted and acknowledged software engineering standards and best practices. At each stage of the project, company guidelines with regard to industry practices were taken into consideration, including the IEEE and ISO/IEC standards of software development. Observation of responsible cybersecurity was maintained when handling data and the datasets that were being used were pulled out of legitimate and publicly available malware repositories to prevent any legal or ethical issues.

The project was implemented in accordance with the structured programming concepts and modular design facilitating the scalability and simplicity of maintenance. Reproducibility and compatibility with commonly used development environments were achieved by the usage of standard machine learning libraries like Scikit-learn, TensorFlow, and Pandas. The experiments were all recorded in an orderly fashion ensuring that reproducible research methods have been employed in a manner that allows repetition and accurate verification of results.

Security-wise, design of the system fell under satisfying general good cybersecurity practice since individual malicious code was not triggered or executed in the static analysis process and therefore operational-related risks are avoided. Ethical aspects, as suggested by ACM Code of Ethics, were also considered in decision-making, especially when dealing with sensitive data sets and transparency of models where it is important to explain the model using such methods as LIME. All in all, this application of standards to the system promotes its technical integrity not only in the identified area of malware research and malware detection but in the professional, ethical, and legal aspects as well.

5.1.1 Software Standards

Software development on this thesis had adhered to the norms of software engineering in order to achieve the quality of codes, maintainability and interoperability of the codes. A set of development practices was based on ISO/IEC 25010, and refers to the following quality attributes of the software functionality, reliability, usability, efficiency, maintainability and portability. The installation was performed by Python using the industry-standard libraries, i.e. Scikit-learn, TensorFlow, Pandas, which are open-source, and well-known within the research community. To keep track of changes and provide collaboration in development, version control was achieved by using Git. The naming conventions used the PEP 8 Python style guide to make them increasingly readable and consistent throughout the code. They involved testing procedures to prove the appropriateness of machine learning models in reference to normative assessment metrics so that the findings could be deemed trustworthy and reproducible. Complying with these software standards, the project ensures that the implementation is not only practical in identifying malware, but also informed by industry best practices favoring future scaled, integrated and sustainability maintenance.

5.1.2 Hardware Standards

In this study, hardware was based on the commonly accepted standards to achieve compatibility, stability and performance in methodology implementation. The experiments were performed using a workstation with an Intel Core i7 processor, 16 GB RAM and an independent NVIDIA GPU that gave the required processing power in feature extraction and training various machine learning models. The system met the IEEE and industry standards of hardware interoperability and thus every part

worked together effectively. The SATA and NvMe protocols were used by the storage devices in order to read and write quickly and with minimal complications, which supported the process of dealing with large amounts of malware. The choice of hardware configuration aimed at striking a reasonable balance between performance and cost, as it is important to be able to replicate the presented malware detection framework on other mid-to-high-range computers without having to procure special equipment. Through the identified hardware standards, the project will ensure that the project methodology can be effectively enabled and scaled to the various computing environments, allowing relaxing of research and practice applications.

5.1.3 Communication Standards

Communication architecture within this project had to be structured according to the standards of networking and data exchange to secure, precise, and fast expression of information among the various elements of the systems. Even though the malware detection framework, proposed in the text, mainly uses the methods of operation on static dataset, the approach was designed in a way that could interface with typical communication channels, including HTTP/HTTPS and TCP/IP, which would allow integrating the method with cloud-based detection platforms or network-based ones. Data manipulation was organized in standard formats as CSV and JSON to systems to achieve interoperability and thus the sharing of results and extracted features was easily done across platforms and analytical tools. Also, these data transfer security practices, in line with encryption and authentication protocols were deemed to guard sensitive files against unauthorized retrieval during storage or transmission. Such communication standards have enabled the system to be extendable into real-time or distributed applications with reliability, compatibility and acceptable security even in practice.

5.2 Impact on Society, Environment and Sustainability

The proposed framework of the static malware detection tool could have a significant positive social effect, the contribution in achieving the higher level of cybersecurity and data protection in the personal and organizational context. This eliminates the possibility of data theft, loss of finances, and malfunction of essential services since data breaches through malicious software are not executed. This increases digital trust and safety. Environmentally, less computing capability is needed to analyze a

system using the static approach as opposed to continuous dynamic monitoring thus decreasing energy usage and resulting carbon footprint. This is more sustainable comparatively based on its efficiency when a long-term deployment is to be done, especially in large-scale or resource-constrained setups. Moreover, the scalability of the methodology does not include hardware requirements that could be too great due to the compatibility of the approach with multiple platforms and the ability to be automated according to the concepts of green computing. All in all, this contribution enhances a safe digital environment in a manner that does not compromise energy efficiency and environmental impact hence the solution will be beneficial socially, environment-friendly and sustainable in the future.

5.2.1 Impact on Life

Machine learning can be used to create a system that would help detect static malware and thus enhance the quality of life in the modern highly digitized world. The system contributes to security of personal data, financial reserves, and internet privacy because it can detect malicious software faster and more accurately. Such decrease in cyber threats does not only protect people against possible damage, but also builds the increased level of trust to digital services, promoting safer online interactions to work, study, receive medical treatment, and communicate. To companies and organizations, this translates to less down time, reduction of cases of interruption of service, and hence better stability of operation, which per recommendations isn t beneficial only to the business but also to the employees, the customers, and the community at large. Finally, with regard to reducing the threats of cyberattacks, and upholding secure computing environment, this effort aids a more secure, safe and stress-free digital experience to the people in diverse sectors of society.

5.2.2 Impact on Society & Environment

Society can significantly benefit through the suggested malware detection system because it will facilitate increased digital safety and resilience to cybercrime. With people, companies, and institutions of the masses falling victim to cyberattacks, early warning systems can ensure that such activities do not result in both money loss and protection of confidential data as well as continuation of confidence in the use of technology-based services. This makes online environment safer not only to

the users but also to the economy at large. The analysis shortened of the system in a static approach, according to environmental perspective, since the heavy and energy-intensive dynamic testing of the system can be reduced, decreasing the general computational energy use. With efficient operations, the system allows its users to apply more environmentally friendly computing, which is in line with the goals of sustainability and minimizes the indirect carbon footprint of cybersecurity activities. The combination of these benefits results in a safer, greener digital environment.

5.2.3 Ethical Aspects

A malware detection system must have moral arguments in development and application since cybersecurity tools can significantly affect user confidentiality and trust. This analysis makes sure that the suggested method works in a way that does not acknowledge personal or sensitive data but simply focuses on the analysis of executable files with the aim of ensuring that there are malicious activities or not. The methodology does not imply intrusive monitoring or collection in general and this is respectful about the user confidences. Furthermore, the detection process should have transparency and fairness of result evaluating should avoid bias against the certain software vendors or platforms. The system will safeguard the digital rights by observing the restrictions of responsible use to enhance safe application of technology. Ethical deployment involves as well the provision of the solution shall be made available to various user categories and not be used inappropriately at the same time legally binding regulations that govern the cybersecurity practices are upheld.

5.2.4 Sustainability Plan

The small footprint, flexibility, and capacity to form part of new cybersecurity structures have been identified as the contributing factors to the sustainability of this malware detection system. The implementation exploits some efficient algorithm and feature extraction techniques which minimize the computational overheads, which makes its application viable to a long-term purpose even on limited resources devices. New trends may be fought in the same old model, with the updates to the model being regular to target new malware threats, whose relevance would not require all new development. The modular design of the framework can be

upgraded easily, and thus it can be modified to become more accurate and efficient and have minimal impacts on the environment in the future due to optimized processing. Also, the availability of publicly available datasets as well as the adoption of open-source tools increases accessibility as it motivates other researchers and developers to collaborate in the maintenance and improvement of the system. The cycle of continuous improvements guarantees the technological relevance preservation and the economical viability of the solution in the long-term perspective, contributing to sustainable cybersecurity practice.

5.3 Project Management and Financial Analysis

Successful project management and budgetary planning were critical elements of undertaking this research successfully in assessing static malwares detection using machine learning. Because the research conducted in the domain of cybersecurity demands precision, effectiveness, and scalability, this section explains how the project was structured, implemented, and managed under the framed limits of predisposition with the focus on cost-effectiveness and reproducibility.

5.3.1 Project Planning and Task Management

An organized, phase-based approach to project management has been taken with elements of inspiration to agile processes to ensure accommodating flexibility and continuous advancement. The project work had stages marked by specific deliverables and review clear checkpoints. Phases covered were the acquisition of datasets, preprocessing and feature engineering, model design and training, attack analysis, interpretability, and last stage of documentation.

The first stage was related to dataset selection with data inspection, determination of missing data, and encoding categorical features. After this, there was the preprocessing of the data which included balancing the class, scaling the data, and selection of features, which decreased the total number of features to 20 critical features. After the dataset was prepared it was divided into 80:20 ratio creating training set and testing set.

The development of the model and their training was the next milestone based on which various machine learning classifiers were applied such as the Random Forest, Logistic Regression, XGBoost, SVM, and ANN. The accuracy, precision, recall, and F1-score have been used to monitor the performance of each model. Then robustness

was tested by use of attack analysis methods (Gaussian noise, feature permutation, random scaling, and extreme scaling).

The last step included explainable AI (LIME) to explain what the model decided to predict which also offered insightful information as to why the model decided what it did. Each activity was also recorded as the research was required to be transparent and reproducible. It supported a system of collaboration and version control based on GitHub and Google Drive and experiment tracking through Jupyter Notebooks and formal reports.

1. Resource and Time Allocation

The project was completed over approximately 5 months (See Table 3.2), divided into the following stages:

- Month 1: Dataset collection, exploratory data analysis, and literature review
- Month 2: Data preprocessing, class balancing, feature selection, and train-test split
- Month 3: Model design and training (Random Forest, XGBoost, Logistic Regression, SVM, ANN)
- Month 4: Attack analysis and robustness evaluation
- Month 5: Explainable AI integration, final evaluation, report writing, and documentation

Using Google Colab for training reduced computational overhead by leveraging free GPU resources, enabling extensive experimentation without requiring dedicated high-end hardware.

2. Financial Analysis

A key strength of the project lies in its cost-effectiveness. The total financial expenditure was minimal due to the use of open-source tools, publicly available datasets, and free cloud computing services. Because of these cost-saving measures, the project is not only academically robust but also financially scalable. The final product—a deployable, interpretable mobile application—can be distributed widely without additional hardware or licensing expenses, making it a viable solution for real-world implementation in developing regions.

3. Scalability and Future Investment

The modular design of the methodology ensures scalability. The feature-based

framework can be easily extended with larger datasets, more advanced feature engineering techniques, or additional machine learning models. Future investments may include paid access to premium computing resources for large-scale experiments, or integration of the system into enterprise-level malware detection platforms. However, the current setup already demonstrates a cost-efficient, reproducible, and practical framework for academic research and potential real-world deployment.

The project followed a phased development model, dividing the entire work into clear stages:

- Requirement Analysis and System Design – Defining research objectives, reviewing technical requirements, selecting algorithms, and designing the overall methodological framework.
- Dataset Preparation and Preprocessing – Performing exploratory data analysis, missing value treatment, encoding, class balancing, feature scaling, and feature selection.
- Model Development and Training – Implementing machine learning classifiers (Random Forest, Logistic Regression, XGBoost, SVM, ANN) and evaluating them on multiple metrics.
- Attack Analysis and Robustness Testing – Applying perturbation techniques such as Gaussian noise, feature permutation, and scaling to measure model stability.
- Explainability and Interpretability – Using LIME to generate interpretable explanations for classification decisions.
- Documentation and Reporting – Recording experimental results, drafting thesis chapters, and preparing the project for publication.

5.3.2 Tools and Platforms Used

Table 5.1 provides an overview of the essential tools and platforms used for model training, development, deployment, and version control throughout the project.

Table 5.1: Details of tools and platform used

Category	Tool/Platform	Purpose
Programming Language	Python	Core ML development and preprocessing
Libraries/Frameworks	Scikit-learn, XGBoost, TensorFlow/Keras	Model training, evaluation, and feature selection
Model Training	Google Colab (GPU)	Training models with scalable compute
IDE	Jupyter Notebook	Experimentation, visualization, and tracking
Version Control	Git/GitHub	Code versioning and collaboration
Storage/Backup	Google Drive	Dataset storage and file sharing

5.3.3 Financial Analysis

This project was designed to be cost-effective and accessible for students or independent researchers. Below is an approximate cost breakdown:

Table 5.2: Estimated Cost and Financial Analysis.

Resource/Item	Estimated Cost (BDT)	Remarks
Google Colab Pro (optional)	3500	Optional for extended GPU time and faster training
Computer (personal or lab use)	Existing Resource	Used for app development and testing
Internet Access	4200	Required for cloud training and resource access
Python Libraries/Frameworks	0	Open-source (Scikit-learn, TensorFlow, XGBoost)
Cloud Storage/Backup (optional)	300	Google Drive or similar platforms
Total	8000 BDT	

5.4 Complex Engineering Problem

In coming up with an efficient malware detection program, there are a number of tricky engineering problems to overcome. Among the challenges, there are difficulties in processing the variety and the fast rate in the development of malware because criminals are continuously tweaking codes to beat detection. This requires a system that is able to learn using very large and heterogeneous records and performs with high accuracy irrespective of varied platforms and environments. Its combination with complex feature extractors, including n-gram analysis and deep learning models, further complicates the matter because making it efficient requires

a large amount of computing resources and exhaustive tuning of parameters. The other issue is balancing the correctness of detection and speed of the process so that the system can be used effectively within a real time system without extreme overloading of hardware or network support. The requirement of compatibility with many operating systems and application scenarios contributes to the engineering effort, and necessitate modular, flexible design. Also, there exist technical and ethical implications in the process of data collection, analysis, and privacy protection of user data, and therefore adhering to high security and compliance standards is critical. All of these interdependent problems require a holistic strategy comprising of a strong system infrastructure, effective algorithms and constant update and changing the model to provide a viable and secure malware detection platform.

5.4.1 Complex Problem Solving

To successfully complete this research project, several aspects of complex engineering problem-solving were involved—ranging from algorithm selection and model integration to mobile optimization and real-time deployment. The following mapping demonstrates how the work aligns with the Engineering Problem (EP) framework:

Table 5.3: Mapping with Complex Problem Solving

EP1	EP2	EP3	EP4	EP5	EP6	EP7
Depth of Knowledge	Conflicting Requirements	Depth of Analysis	Familiarity of Issues	Applicable Codes	Stakeholder Involvement	Interdependence
✓	✓	✓	✓	✓		✓

Justifications:

- EP1 - The project involves the cross-domain expertise due to the mixture of specialist knowledge in the medical imaging (MRI brain scans) and diagnostic systems powered by AI.
- EP2- The system should be in balance between accuracy in the model, speed and interpretability without violating the ethical or medical standards.
- EP3 - The extent of analysis encompasses a set of dataset balancing (SMOTE), DenseNet feature extractor feature extraction and comparative classifier assessment.
- EP4: The problem as epitomized by EP4 is not completely new since brain

tumor detection using AI is not very new, but the application of explainable AI (Grad-CAM) makes it unfamiliar to some extent and difficult.

- EP5 – It must be compliant with the standards such as IEEE software standards, medical device regulations and data privacy policies.
- EP7 - The products of one stage (feature extraction) have direct effect on further stages (classification, visualization, interpretability).

Mapping with Knowledge Profile

Table 5.4: Mapping with Knowledge Profile

K3	K4	K5	K6	K8
Engineering Fundamentals	Specialist Knowledge	Engineering Design	Engineering Practice	Research Literature
✓	✓	✓	✓	✓

Justifications:

- K3 -A good grasp of ML basics, data preparation, and assessment measures is the basis of the work.
- K4 - The experience of applying DenseNet architecture to performing feature extraction in medical images will require specialist knowledge.
- K5 - The pipeline configuration will coordinate preprocessing and feature extraction, classification and Grad-CAM explainability.
- K6 -Coding, debugging, testing and refining models with real MRI data.
- K8 - Research uses the prior literature, and it adds value to the levels of accuracy in detection and interpretability.

5.4.2 Engineering Activities

The development life cycle of the project covered various and sophisticated engineering processes such as preprocessing of the medical images, training of deep learning models, validation of the classifier, the integration of software workflow, exploratory AI visualization, and the deployment of the system. These tasks prove how complicated the development of AI-enhanced healthcare diagnostic system is in real-life scenarios under practical limitations.

Table 5.5: Mapping with Complex Engineering Activities

EA1	EA2	EA3	EA4	EA5
Range of Resources	Level of Interaction	Innovation	Societal & Environmental Consequences	Familiarity
✓	✓	✓	✓	✓

Justifications:

- EA1 - Works with several different assets, including Kaggle MRI datasets, and TensorFlow/PyTorch as an implementation.
- EA2- Software engineering and medical fields should work interdisciplinarily.
- EA3 - Unique solution through the integration of the deep learning feature extraction, ML classification algorithms, and the explainability techniques to medical imaging.
- EA4- Good impact on the society by helping radiologists make quicker and more accurate diagnosis of tumorous growths of the brain.
- EA5 -AI methods are not new to the computer vision procedure; however, they are relatively young when the techniques are integretated with the medical interpretability.

5.5 Summary

This chapter has delineated the engineering requirements, ethical principles, and design aspects as used when coming up with the suggested malware detection framework. To provide compatibility, reliability and scalability, the system has been developed in synergy with known software, hardware and communication standards. The safe coding standards, computing resource utilization and the compliance of networking standards were carefully followed because they directly affect the precision, speed and strength of malware detection.

The chapter covered as well the issues of the system in terms of societal, environmental, and ethical implications. Because malware detection is critical to the security of the digital infrastructures, the solution was developed in a way that it contributes to the cybersecurity resilience without negatively affecting the privacy and safety of user data. The issues of environmental sustainability have been solved by increasing computational efficiency, minimizing processing overhead, and the possibility of running the system with a considerable amount of energy due to having minimum energy requirements.

Also, the difficulties in the solving complex engineering problems in this field were discussed. These problems were to be solved using large and various data sets, real-time detection functionality, and the flexibility of the detection model to new malware threats. These were taken care of in the methodology through the use of solid feature extraction techniques, application of machine learning algorithms that take into consideration high-dimensional structured data, the use of modular system design that is scalable.

And with the inclusion of engineering standards, consideration of ethical and environmental issues, and the evaluation of the natural complexity of the issue statements, it is possible to say that this chapter established the basis of a stable and sustainable malware detection system. The implications and measures described here can have a direct impact on the overall performance of the system and guarantee that it will comply with both the technological and the social requirements.

Chapter 6

Conclusion

The chapter is the conclusion of the research where some major findings are outlined and the contribution of the suggested framework of static malware detection to these findings is mentioned. It further talks about the drawbacks that were faced throughout the implementation process and points out potential future improvement channels. The chapter has a reflection on the effective nature of the system and how it can be enhanced into a more comprehensive, interpretable and scalable solution in real-time threat detection of cyberattacks through machine learning tools.

6.1 Summary

The thrust of this thesis was to introduce a machine learning-based feature-based static malware detection system that would contribute to refinement of the accuracy and effectiveness of detecting a malicious program without the need of executing the files at all. The motivation behind the study was well known, namely the increasing malware threat that keeps acquiring ever more complexity and another quantity. The proposed combination of static analysis methods and a strong machine learning algorithm attempted to offer a quicker, safer, and more resource-friendly approach of detecting malware as compared to the old-fashioned dynamic analysis.

The methodology was carried out in a standard way and began with collection and preprocessing of the datasets, including the extraction of datasets of executable files features. These features were then selected to be used to train and test multiple machine learning models whose accuracy, precision, and recall were to be compared in order to find out the most effective classifier. Automation, scalability, and adaptability was also tradeoffs pointed out in the workflow that allowed the framework to be utilized in real-life settings of cybersecurity. The obtained accuracy proved that the selected model yielded competitive results prooving the feasibility of feature-based static analysis in actual malware detection practices.

In addition to the technical findings, the paper has also discussed the implications of a wider application of the framework. It emphasized the need to follow engineering safety and to use ethical data handling schemes and to be sustainable

to the environment in the process of computational work. This system has a modular design that gives sustainability and future readiness in the system to manage new variants of malware since the system can be updated in case of a new variant.

To sum it all, the study achieved the set goals and objectives since it provided a secure and performing malware detection mechanism that balances performance, resource, and ethical frameworks. Although the research delivered encouraging performances, it too did not deny to some limits, which included its reliance on the quality of training data as well as difficulty in locating intensely obfuscated malware. Further research should be targeted at the inclusion of hybrid analysis methods, the enlargement of the dataset, and the implementation of the framework in a real-world network setting to enhance its performance and robustness against current and future threats even more.

6.2 Limitation

Although the presented feature-based static malware detection model has shown promising results, it still has a few drawbacks that can possibly impact the relevance and generalizability of the model in wider or more complicated contexts. Among the main drawbacks, the research depends on how good and wide the data, employed in training and testing will be. Unless the dataset has enough sample malware of various compounds, especially the newly emerging or rare forms, the model will lack the capacity to generalize to threats it has not encountered. This may lead to reduced detection accuracy in real life environments where the characteristics of malware evolve fast.

The other limitation concerns the limitations of the static analysis in general. As the technique is dependent on analysis of the file code and structure rather than running it, it was possible that some more advanced malware, heavily obfuscated and so forth, could miss detection. Although optimization and hand-selection of features can increase accuracy, it is possible that attacking such static methodologies which operate on detection models may be insufficient to deal with these sophisticated evasion methods.

Practically, the assumption of the framework is also that the extracted features are reflective of where they represent good or malicious behavior, which does not necessarily match when the malware is complex and involves multiple phases. Moreover, to ensure relevance in battling the ever-changing threat environment, it

has been found that machine learning algorithms will need frequently to be updated/retrained. This brings issues of maintenance that are continuous more so when it comes to a large-scale deployment.

Finally, this working was confined to the assessment of the performance in administrative experimental conditions. The real-time speed of processing, compatibility with in place security infrastructure, and response to live attack on the network were some of the factors that were not sufficiently covered. All these areas would need more testing and optimization before being implemented into production scale cybersecurity systems.

6.3 Future Work

To improve the efficiency and application of the suggested framework, there is a range of directions that one can apply based on the success of the current study. Among the steps that are relevant is increasing the number of samples in the dataset such that a broader variety of samples is represented by newly discovered and recently formed malware families. This will go to enhance the level of generalization the model has on the various forms of threats such as zero-day threats and those that are highly advanced. It is also possible that the system would incorporate the constantly updated changes into the threat intelligence feeds to become more agile to new patterns.

The other exciting avenue is the determination of dynamic analysis techniques with static feature extraction. Incorporating behavioral information gathered by the run-time monitoring and structural characteristics the system would be able to identify threats which heavily depend on obfuscation or code polymorphism. It is possible that this hybrid technique will detect with higher accuracy, especially against APTs. Algorithmically, there is potential in deeper learning models, particularly on transformer-based models, or graph neural networks, that may provide deeper insights into the real-world relationships between the features in malware. Moreover, ensemble learning methods integrating several machine learning classifiers could provide a more solid and precise outcome even within different diverse conditions.

With regards to deployment, further research can be conducted to ensure that the system is optimized to be used within operation environments in real-time detection. This encompasses enhancing the efficiency of computation and reducing its latency, as well as its compatibility with cloud-based or distributed cyber-security

infrastructure. It might also be highly desirable to integrate the solution into endpoint protection platforms or security information and event management (SIEM) systems, thereby augmenting its current usefulness.

Lastly, the addition of interpretability capabilities beyond Grad-CAM to incorporate explainable AI (XAI) techniques with the use of malware detection would provide a greater sense of trust and transparency with regards to the decisions. Presenting classifications results with the easily comprehensible and understandable reasoning behind it will be especially beneficial in the process of professional cybersecurity when the analysts require both precision and meaningful implications.

References

[1] Ali, M., Shiaeles, S., Bendiab, G., & Ghita, B. (2020). MALGRA: Machine learning and N-gram malware feature extraction and detection system. Electronics, 9(11), 1777.

[2] Bakır, H., & Bakır, R. (2023). DroidEncoder: Malware detection using auto-encoder based feature extractor and machine learning algorithms. Computers and Electrical Engineering, 110, 108804.

[3] Zhang, N., Tan, Y. A., Yang, C., & Li, Y. (2021). Deep learning feature exploration for android malware detection. Applied Soft Computing, 102, 107069.

[4] Rathore, H., Agarwal, S., Sahay, S. K., & Sewak, M. (2018, November). Malware detection using machine learning and deep learning. In International Conference on Big Data Analytics (pp. 402-411). Cham: Springer International Publishing.

[5] Dabas, N., Ahlawat, P., & Sharma, P. (2023). An effective malware detection method using hybrid feature selection and machine learning algorithms. Arabian Journal for Science and Engineering, 48(8), 9749-9767.

[6] Shaukat, K., Luo, S., & Varadharajan, V. (2024). A novel machine learning approach for detecting first-time-appeared malware. Engineering Applications of Artificial Intelligence, 131, 107801.

[7] Dambra, S., Han, Y., Aonzo, S., Kotzias, P., Vitale, A., Caballero, J., ... & Bilge, L. (2023, November). Decoding the secrets of machine learning in malware classification: A deep dive into datasets, feature extraction, and model performance. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (pp. 60-74).

[8] Chaganti, R., Ravi, V., & Pham, T. D. (2022). Deep learning based cross architecture internet of things malware detection and classification. Computers & Security, 120, 102779.

[9] Akhtar, M. S., & Feng, T. (2022). Detection of malware by deep learning as CNN-LSTM machine learning techniques in real time. Symmetry, 14(11), 2308.

[10] Akhtar, M. S., & Feng, T. (2022). Detection of malware by deep learning as CNN-LSTM machine learning techniques in real time. Symmetry, 14(11), 2308.

[11] Alomari, E. S., Nuiaa, R. R., Alyasseri, Z. A. A., Mohammed, H. J., Sani, N. S., Esa, M. I., & Musawi, B. A. (2023). Malware detection using deep learning and correlation-based feature selection. Symmetry, 15(1), 123.

[12] Odat, E., & Yaseen, Q. M. (2023). A novel machine learning approach for android malware detection based on the co-existence of features. IEEE Access, 11, 15471-

15484.

[13]Odat, E., & Yaseen, Q. M. (2023). A novel machine learning approach for android malware detection based on the co-existence of features. IEEE Access, 11, 15471-15484.

[14]Dolesi, K., Steinbach, E., Velasquez, A., Whitaker, L., Baranov, M., & Atherton, L. (2024). A machine learning approach to ransomware detection using opcode features and k-nearest neighbors on windows. Authorea Preprints.

[15]Chaganti, R., Ravi, V., & Pham, T. D. (2023). A multi-view feature fusion approach for effective malware classification using Deep Learning. Journal of information security and applications, 72, 103402.

[16]Herrera-Silva, J. A., & Hernández-Álvarez, M. (2023). Dynamic feature dataset for ransomware detection using machine learning algorithms. Sensors, 23(3), 1053.

[17]Urooj, B., Shah, M. A., Maple, C., Abbasi, M. K., & Riasat, S. (2022). Malware detection: a framework for reverse engineered android applications through machine learning algorithms. IEEE Access, 10, 89031-89050.

[18]Kouliaridis, V., & Kambourakis, G. (2021). A comprehensive survey on machine learning techniques for android malware detection. Information, 12(5), 185.

[19]Azeem, M., Khan, D., Iftikhar, S., Bawazeer, S., & Alzahrani, M. (2024). Analyzing and comparing the effectiveness of malware detection: A study of machine learning approaches. Heliyon, 10(1).

[20]Akhtar, M. S., & Feng, T. (2023). Evaluation of machine learning algorithms for malware detection. Sensors, 23(2), 946.

[21]Hussain, A., Asif, M., Ahmad, M. B., Mahmood, T., & Raza, M. A. (2022, April). Malware detection using machine learning algorithms for windows platform. In Proceedings of International Conference on Information Technology and Applications: ICITA 2021 (pp. 619-632). Singapore: Springer Nature Singapore.

[22]Lee, J., Jang, H., Ha, S., & Yoon, Y. (2021). Android malware detection using machine learning with feature selection based on the genetic algorithm. Mathematics, 9(21), 2813.

Static Malware Detection

ORIGINALITY REPORT

9%	6%	6%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	2%
2	www.mdpi.com Internet Source	1%
3	Tiago Augusto Orcajo Demay Cordeiro. "Detecção de anomalias na comunicação entre veículos aéreos autônomos utilizando técnicas de machine learning e cibersegurança.", Universidade de São Paulo. Agência de Bibliotecas e Coleções Digitais, 2024 Publication	1%
4	www.researchgate.net Internet Source	<1%
5	cbirt.net Internet Source	<1%
6	Submitted to Buckinghamshire Chilterns University College Student Paper	<1%
7	Submitted to University of Abertay Dundee Student Paper	<1%
8	Egling, Theodore. "Differential Evolution Optimization Algorithms and its Application in Machine Learning Based Disease Detection", University of South Africa (South Africa) Publication	<1%

20% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

46 AI-generated only 20%
Likely AI-generated text from a large-language model.

0 AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.