DEVELOPMENT OF A HYBRID MALWARE DETECTION SYSTEM USING BEHAVIORAL FEATURES

BY

ESTHER UBI
GOU/U22/CSC/779

DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS, FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY, GODFREY OKOYE UNIVERSITY, ENUGU STATE.

JUNE, 2026.

DEVELOPMENT OF A HYBRID MALWARE DETECTION SYSTEM USING BEHAVIORAL FEATURES

BY

ESTHER UBI
GOU/U22/CSC/779

DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS, FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY, GODFREY OKOYE UNIVERSITY, ENUGU STATE.

IN PARTIAL FULFILLMENT OF THE AWARD OF BACHELOR OF SCIENCE (B.Sc), DEGREE IN COMPUTER SCIENCE.

SUPERVISOR: DR. FRANK OKEBANAMA

JUNE, 2026.

APPROVAL PAGE

This research, titled "**Development of a Hybrid Malware Detection System Using Behavioral Features**," has been assessed and approved by the Department of Computer Science and Mathematics Committees in the Faculty of Natural Sciences and Environmental Studies, Godfrey Okoye University, Enugu, Nigeria.

Dr. Frank Okebanama	**.....**	**.....**
Supervisor	Signature	Date
External Examiner	**.....**	**.....**
	Signature	Date
Dr. Frank Okebanama	**.....**	**.....**
HOD, Department of Computer Science and Mathematics	Signature	Date
Prof. I.A Ajah	**.....**	**.....**
Dean, Faculty of Computing And Information Technology.	Signature	Date

CERTIFICATION

I certify that i completed the research included therein and that it was primarily based on my observations and investigation. Consequently, I will fully accept the University's decision if I am found to have cheated on the assignment.

_____

ESTHER UBI
GOU/U22/CSC/779

DEDICATION

To God, who gave me the strength to begin, the grace to continue, and the faith to finish. Every page of this work is a reminder that none of it was by my might alone.

And to my parents, for every sacrifice made quietly, every prayer whispered on my behalf, and every word of encouragement that reached me exactly when I needed it most. You believed in me before I believed in myself, and this achievement belongs to you just as much as it belongs to me.

I hope this makes you proud.

ACKNOWLEDGEMENTS

First and foremost, I give all glory and gratitude to God, whose grace carried me through every moment of this journey, especially during periods of uncertainty and challenge. This project stands, above all, as a testament to His faithfulness.

Deep appreciation is extended to my supervisor and Head of Department, Dr. Frank Okebanama, whose guidance shaped not only this project but also my entire approach to thinking carefully and working diligently. Having a supervisor who also served as Head of Department was a privilege, as the support received extended beyond research supervision to a genuine investment in academic and personal growth. Sincere gratitude is expressed for his patience, encouragement, and direction.

Appreciation is also extended to all lecturers in the Department of Computer Science and Mathematics at Godfrey Okoye University. The knowledge, skills, and values imparted throughout the years provided the foundation upon which this work was built. Their dedication to teaching and mentorship is deeply appreciated.

University life was not without its challenges. Balancing academic responsibilities with the realities of everyday life required perseverance and determination. During periods of pressure, exhaustion, and self-doubt, the unwavering support of my family remained a constant source of strength. Heartfelt gratitude is extended to every member of the family for their encouragement, sacrifices, and belief in the successful completion of this journey.

Special thanks are also due to friends whose encouragement, laughter, and steadfast support provided comfort and motivation during difficult times. Their presence made challenging moments more manageable and memorable moments even more meaningful.

Finally, sincere appreciation is extended to everyone who contributed to this journey, whether through words of encouragement, acts of kindness, attentive listening, or prayers. Their support played an invaluable role in the successful completion of this work.

ABSTRACT

Malware detection remains a critical issue in the cybersecurity domain. As the prevalence of obfuscated, polymorphic and unknown malicious attacks grow, traditional signature-based methods struggle to keep pace. This work addresses some of these shortcomings of single-model malware detection techniques and lack of interpretable explanations of their behaviors by proposing and developing a hybrid system based on behavioral analysis and stacking ensembles in machine learning. A total of 49,179 malicious Windows PE samples were gathered from the Avast-CTU CAPEv2 dataset and a collection of benign samples curated by a researcher to train the system. Forty-two static PE and behavioral features were extracted from sandbox execution reports utilizing a custom feature extraction mechanism. Two complementory base learners, an XGBoost gradient boosting classifier and a PyTorch Multilayer Perceptron (MLP) were used and combined using a stacking ensemble architecture with Logistic Regression as the meta-learner. Models were trained and tested on identically same settings. A deployment threshold was fixed at 0.96. The results confirmed that the proposed hybrid ensemble model outperforms its base learners and reached the accuracy of precision, recall and F1-score of 1.0000, 0.9999 and 0.9999 with the false positive rate 0.0000 for a test data size of 9,836 samples. Alongside the high detection performance, the system also provides rule-based interpretable explanations of the detected malware by translating model prediction into simple human understandable behavioral observations. The obtained system was implemented as a web-based application incorporating a FastAPI backend, CAPE sandbox, and React frontend. The proposed work successfully shows that a hybrid stacking ensemble model can enhance the detection performance of malware detection system while maintaining interpretability.

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

The fast-growing use of ICT has brought about transformation in the ways in which individuals, organizations, and government conduct their affairs. Information systems are used in education, banking, health care, communications, business, and administration among other domains. With increased dependence on such systems, the importance of information security, system integrity, and availability has become more significant. However, the increased dependency has led to increased chances for cybercriminals, thus increasing cybersecurity challenges across developed and developing areas (Saeed *et al*., 2023). In Africa, threat assessments reveal that cybercrime is increasing with digitization, with Nigeria being one of the countries with cyber threats (INTERPOL, 2025).

Malware is one of the greatest risks to cybersecurity. The term "malware" stands for malicious software and means any piece of software deliberately developed to interfere with functions, corrupt data, collect private information, spy on users, and illegally access computers and networks. Malware includes viruses, worms, Trojan horses, ransomware, spyware, and rootkits. Examples that are famous include Emotet, Trickbot, and Qakbot, which have made immense damage in the last few years, highlighting how a single malware family can affect thousands of computers and lead to substantial financial and operational losses. Regardless of the techniques used by the malwares, the threats associated with them are quite similar and pose major risks to the security, integrity, and availability of data. Detection of malwares is an important aspect of computer science research (Ferdous *et al*., 2025).

Malware detection is necessary for minimizing the risks of damage that could be caused to systems. Signature-based detection is one of the most common forms of traditional detection

and works by matching files or software against a collection of known malicious patterns.Signature-based approaches offer a number of advantages since they operate quite fast and are quite efficient in terms of identifying already encountered malware. But at the same time, these approaches suffer from numerous shortcomings. They depend greatly on database updates, and perform poorly against modified malware. Mordern day malware creators are increasingly employing the application of various obfuscation methods, such as polymorphism that alters the look of malware each time it is spread, as well as metamorphism that alters the code of malware but retains its functionality (Louthánová *et al*., 2024). Therefore, signature-based detection mechanisms cannot be considered adequate for use against rapidly evolving malwares.

To counter such shortcomings, many researchers have recently resorted to behavior-based detection of malware. Behavior-based detection does not depend on the static nature of the file but rather on what happens when the program is run. This usually includes watching system and API calls, registry access, file operations, memory usage, process creation, and network connectivity (Li *et al*., 2022). Behavioral attributes are useful because no matter how much any malware tries to change its code structure to evade detection, there are certain behaviors it has to execute for the same purpose. Thus, behavioral analysis provides the actual intent of any program. Research on dynamic malware analysis reveals that API call sequences along with other runtime characteristics are highly relevant in representing program behavior and differentiating between malicious activities and legitimate ones (Maniriho *et al*., 2023).

The increased application of machine learning in malware detection has been attributed largely to this emphasis on improved feature representation. Machine learning enables a system to learn from data and predict results independently from human-programmed rules. In the case of malware detection, machine learning models can use the feature extraction process for both malicious and non-malicious applications to identify the differences between them (Singh &

Singh, 2021). Their flexibility makes them more efficient compared to only rule-based techniques, particularly when dealing with big amounts of data and different malware variants. According to recent studies, the use of machine learning has become an area of considerable importance for malware research because of their capacity to recognize complex patterns and enable automation and detection of new types of malware. On the other hand, the available literature indicates that the results of using models in malware research depend significantly on the quality of features used, the chosen learning algorithms, and the quality of the training data (Rafique *et al*., 2025). However, using only one machine learning algorithm may not always provide the best results, because all the algorithms have their pros and cons. The aforementioned statement is what has led many scholars to investigate the possibility of using hybrid and ensemble methodologies, in order to get improved results through combining the output from several algorithms or methods (Onyedinma *et al*., 2025). As hybrid machine learning techniques become more common in the detection of malware, it is worth noting that the increased complexity of such hybrids needs proper justification. From the work carried out by Alhashmi *et al.* (2023), the effectiveness of a hybrid approach does not arise from the combination of various algorithms; instead, it relies on how the combinations of these algorithms complement each other in prediction performance beyond the performance of a single classifier. Any proposed hybrid approach should be properly evaluated against baseline single techniques to show its advantage (Rodrigo *et al*., 2021).

In light of this context, it is evident that there is a necessity of adopting malware detection techniques that go beyond traditional signatures and leverage behavioral data along with advanced learning approaches. The research proposes a hybrid malware detection approach using behavioral characteristics through an ensemble of two machine learning algorithms in a stacking framework. The goal is to enhance the detection performance beyond the existing techniques of using one model and create a system that would be able to detect both the old

malware and the new malware variants based on its runtime behavior. This research contributes to the wider process of creation of more flexible malware detection systems.

1.2 Statement of the Problem

Though there have been advances made in the use of behavioral malware detection techniques, there still remain some very important challenges which prevent the detection system from being effective. The challenges apply to both the efficiency of the detection system as well as its usability by non-technical end-users.

Firstly, although behavioral analysis is more robust than signature-based methods for detecting novel and evasive malware attacks, machine learning models fail to generalize well to the variety of behavior patterns presented by modern malware families (Jurečková *et al.*, 2024). However, a single model tends to be good at detecting specific forms of malware activity while poor at others based on the training set and nature of the learning algorithm involved. This is especially important considering the quick appearance of new types of malware where using one model to detect all types will be difficult across different malware families.

Secondly, while hybrid and ensemble methods have been suggested to overcome the shortcomings of individual models for detecting threats, their performance differs significantly based on how they are combined and evaluated (Dhanya *et al.*, 2022). However, many previous research works focus only on overall accuracy rate and do not provide any convincing proof that the proposed hybrid system indeed shows better performance than the individual component models used separately, on the same data. Therefore, there is an evident need for hybrid malware detection systems to be developed and evaluated properly, in particular by comparing them with the base component models.

Thirdly, most current malware detection algorithms are aimed at security experts, and their outputs are not understandable for laymen. For example, an output of such algorithm can be that the file is detected as malware with a certain probability, and there will be no explanation

of this decision provided at all. With the rise of malware attacks being aimed at people and organizations who lack any particular knowledge in cybersecurity, the requirement for malware detectors that can present information about their analysis in a way that is understandable has become vital. Otherwise, the user may mistrust the system, disregard its alerts, or be unable to act upon them.

These three challenges, namely the drawbacks of using single-model detection methods, the necessity for having hybrid techniques which are thoroughly evaluated, and the lack of comprehensible explanations, serve as the major drivers for developing a hybrid malware detection system that uses complementary machine learning algorithms with proper evaluation and comprehensible explanations for non-technical users.

1.3 Aim and Objectives of the Study

The aim of this research is to build a hybrid malware detection system using behavioral features that can efficiently distinguish between malicious and benign software.

In order to attain the above stated aim, the research will be guided by the following specific objectives:

1. Perform a review of existing malware detection systems which include traditional signature based detection systems, individual machine learning models, hybrid and ensemble malware detection systems, and behavioral feature based malware detection systems.
2. Get a set of behavioral features from the malware and benign software execution reports generated using a sandbox environment.
3. Creation and development of a hybrid malware detection model using Gradient Boosting Machine and Multilayer Perceptron within a stacking ensemble framework using logistic regression as the meta-learner.
4. Evaluating the hybrid model against its individual components through performance metrics, significance tests, calibration tests, and per-family error analysis.

5. Creation of an explainability module based on rules for converting model output into human-understandable language.

6. implement the trained models and explainability component within a web interface for file analysis and communication of detection results to the end-users.

1.4 Significance of the Study

This study is significant since it provides a solution to an issue that is persistent and ever-evolving in the field of cybersecurity, the detection of malware that is new and changing. Signature-based techniques work well in detecting known malware but fall short when malware changes its structure to evade detection. By concentrating on behavioral characteristics extracted from the runtime execution process, the study makes a contribution to the detection method, which is based on the real actions taken by malware, as opposed to their appearance, thus providing better protection against polymorphic and metamorphic evasion tactics.

The proposed hybrid approach in this research is beneficial for the wider scope of machine learning for cyber security. In contrast to using one classifier, the study shows how to use the benefits of both Gradient Boosting Machine and Multilayer Perceptron by means of stacking ensemble approach. Notably, the model is critically tested in relation to its individual base learners through statistical significance test, calibration test, and per-family error rate analysis. This methodology helps to address a noted gap in the literature, in which many hybrid models fail to show any clear advantage over their constituent parts, reporting only overall accuracy.

One unique aspect of the study is the integration of an explainability module that explains the predictions of the model in language understandable to ordinary people. Current malware detection models are developed for security experts, with the output being complex and challenging for users without security skills to understand. With this unique aspect, the output of this study can benefit not only cybersecurity experts but also individuals, and small organizations without any knowledge in cybersecurity. With malware becoming more

prevalent in users who lack technical knowledge, the importance of such accessibility becomes clear since it will help establish trust in these detection systems.

In terms of academia, this paper provides a sample of how problems related to cyber security can be resolved by using machine learning, behavioral analysis, and explainable artificial intelligence. The paper serves as a source of information for students and researchers who work on hybrid detection systems, behavior-based security systems, and intelligent systems. In terms of practical applications, the created system can serve as a sample of how malware detection can be used to inform about cyber security where there are no professional resources available.

1.5 Scope of the Study

The scope of this research paper focuses on building and assessing a hybrid system for malware detection of Windows executable files. This malware detection framework approaches the problem as a binary classification task that classifies whether the software is malicious or not based on their behaviors extracted from sandboxing execution reports. Malware data is extracted from the open source dataset Avast-CTU CAPEv2, while benign data is produced from running genuine software within a controlled CAPE sandbox environment. The proposed model employs the Gradient Boosting Machine and Multilayer Perceptron with a stacking technique, where the logistic regression is utilized as the meta-learner, and evaluated on the basis of accuracy, precision, recall, F1-score, ROC-AUC, significance testing, and calibration methods. An explanation module explains the predictions in natural language, and the entire system can be accessed through a web-based interface that is user-friendly for both technical and non-technical individuals.

The scope of this research is restricted to the behavioral feature-based detection of Windows executable malware. It does not include any malware designed for mobile devices (like Android or iOS), Linux or MACOS machines. Furthermore, it does not include any other type of cyber threats like phishing, spamming, intrusions, or network based attacks. The program

has been created only as a prototype in order to conduct some research and is not intended for wide commercial use.

1.6 Limitations of the Study

There are certain limitations that this study is subjected to. First, the dataset used in this case, the Avast-CTU CAPEv2 dataset, comprises malware samples belonging to ten specific malware families ranging from 2017 to 2019. It is true that the dataset is publicly accessible and quite suitable for use in research purposes; however, the most recent malware families emerging after 2020 are not included in the database, nor is there representation of all the other kinds of malware that exist in reality.

Second, the behavioral characteristics employed in this research are obtained from the execution of a malware sample within a CAPE sandbox environment. Some modern malware uses sandbox-detection techniques to change their behavior if they detect that they are being tested. This means that such malware might generate incomplete or incorrect behavioral characteristics. Additionally, the set of features that can be extracted is constrained by the amount of information collected by CAPE during execution since there are more potentially valuable behavior signals that can be gathered but are not included in the list of features used in this paper.

Thirdly, the benign samples employed in the research process have been obtained through running selected Windows programs within the same CAPE environment as the one where the malware was analyzed. While it ensures consistency in the feature generation process, the set of benign samples that is thus generated does not capture all aspects of legitimate software as it exists in the real world.

Finally, hardware and time limitations restrict the range of hyperparameter tuning and the testing of different models. In this study, the tested hybrids are considered a rational choice but not an exhaustive examination of all potential base learners and meta-learners. In addition, the

testing of the system takes place within an experimental setting and not in real-life situations when adversaries can make attempts to escape being detected.

1.7 Operational Definition of Terms

1. **Malware:** Malicious software that is specifically designed to interfere with the functioning of computer systems, to destroy data, spy on users' activities, or to gain unauthorized access to computers.

2. **Malware Detection**: A procedure aimed at detecting if software samples are malicious or harmless by employing certain characteristics and machine learning techniques.

3. **Hybrid Malware Detection Model:** An approach to detection that utilizes two or more learning algorithms to enhance the classification of software into malware or benign.

4. **Behavioral Features:** The actions of a software program as it executes, such as its system calls, API calls, file interactions, registry manipulation, process interactions, and network behavior.

5. **Machine Learning:** It is an application of artificial intelligence where computers learn patterns from data and make predictions using it without explicit programming for that particular task.

6. **Dataset:** A set of malware and benign software samples, along with their extracted feature values, used for the purpose of developing the model.

7. **Feature Extraction:** The process of transforming raw data collected through observation of a program’s behavior into numeric form usable by machine learning algorithms.

8. **Benign Software:** Harmless software whose actions do not affect a computer in any negative way.

9. **Behavior-Based Detection:** A technique to detect malware based on actions of a program while running rather than depending only on static analysis or signature of the program.

10. **Signature-Based Detection:** A conventional approach for malware detection where programs are checked against malicious signatures in a database.

11. **Polymorphic Malware:** A kind of malware that can change its appearance every time it spreads, without changing its inherent malicious behavior to avoid signature-based detection.

12. **Metamorphic Malware:** A kind of malware that can rewrite the code structure of its software program without changing its overall functionality.

13. **Sandbox:** An environment wherein the execution of a program is performed safely without causing any harm to the system.

14. **Training:** The phase where the machine learning algorithm learns the patterns from the selected features of the dataset.

15. **Testing:** It is the process of evaluating the ability of a trained machine learning model to classify new data as malware or benign.

16. **Detection Accuracy:** It is the percentage of software samples that are classified by the model as either malware or benign.

17. **Precision:** The ratio of samples classified as a particular category by the model that actually belonged to that category. Malware accuracy refers to the ratio of files classified as malware that actually were malware.

18. **Recall:** The ratio of samples belonging to a particular category that were correctly identified by the model to be from that category. Malware recall refers to the ratio of malware files that were correctly classified as malware.

19. **F1-Score:** Harmonic Mean of Precision and Recall, computed for a particular class to harmonize the two values as one value.

20. **False Positive Rate:** The percentage of data samples that do not belong to a particular class but get classified into the class by the machine learning algorithm. In the case of binary

malware classification, this usually refers to the percentage of benign software data samples that get misclassified as malware.

21. **Model Generalization:** Refers to how well the machine learning model is able to generalize its performance on unseen samples outside the training sample set.

22. **Stacking Ensemble:** Is a machine learning approach where predictions from several base learners are combined using another learner known as a meta-learner that produces the final classification.

23. **Gradient Boosting Machine (GBM):** A machine learning technique which constructs a sequence of decision trees where each successive tree learns from the mistakes made by the preceding trees.

24. **Multilayer Perceptron (MLP):** A kind of artificial neural network consisting of an input layer, several hidden layers, and an output layer with each layer being completely connected to its successor layer.

25. **Meta-Learner:** A model used in a stacked classifier that uses outputs of the base learners in order to generate the final classification.

26. **Explainability:** An ability of a machine learning algorithm to give human-understandable explanations to its predictions.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The present chapter is concerned with the literature review relevant to the hybrid malware detection using behavioral features and machine learning. The review analyzes previous works in the field, highlights their strengths and weaknesses, and explains why a gap exists which requires the current study. Furthermore, the theoretical background for the design of the study is also included in the review.

Four sections are used to structure this chapter. The section 2.2 discusses the theory related to the topic including malware and detection approaches, behavioral analysis and feature extraction through sandbox, use of machine learning for malware classification, the base learning algorithms used in this study, hybrid and stacking ensemble learning and explainable artificial intelligence in cybersecurity. The section 2.3 discusses related literature about malware detection through machine learning approach including their methodology, results and limitations. The section 2.4 discusses the summarized related literature through tables and section 2.5 discusses the gaps in the existing literature.

The literature review shows that there are a number of issues even though great progress has been made in using machine learning to detect malicious software. Some of these problems are lack of comparative analysis between hybrid models and their base classifiers, lack of focus on user-friendly explainability of the model and limited research done on the particular stacking technique discussed in this research paper.

2.2 Theoretical Background

2.2.1 Malware and Detection Approaches

Malware can be defined as the software that disrupts the proper operation of the system, steals valuable information, or exploits the resources of computers without any kind of permission.

There is a wide range of malware in terms of viruses, worms, trojan horses, ransomware, spyware, rootkits, and bots. Modern forms of malware belong to several families of malware including Emotet, TrickBot, QakBot, Zeus, and Dridex, which evolve through time by developing new behavioral patterns (Bensaoud *et al.*, 2024). Various kinds of methods have been created to detect malicious software. The signature-based approach uses some patterns or signatures to detect malware that has already been known. This approach does not work when it comes to detection of new forms of malware. The heuristic approach relies on rules for detecting suspicious activities. Behaviour detection relies on various activities undertaken by the software while being executed like calling API, file I/O operation, changes in registry, and network operations. This malware analysis technique has been found to be far better due to the reason that this is based on the behavior of the program rather than the signature of the program. Behavior analysis forms the core part of this research (Rahima Manzil & Naik, 2023).

2.2.2 Behavioural Analysis and Feature Extraction using Sandbox Technology

Behavioural analysis is a process that entails conducting an examination of how the software behaves when it is executing. The activities include calling APIs, accessing files, changing the registry, creating processes, as well as communication over the internet. The method performs efficiently because there will be a time when the malware will have to do something harmful despite its obfuscated code (Torres *et al.*, 2023).

To achieve this purpose, the malware sample is executed within sandboxed platforms such as CAPE, where events taking place at the system level are recorded in form of report files. Since the reports generated by these sandbox environments may be extensive and complex, there is a need to extract features from them and convert them to numerical form for feeding to the machine learning algorithms. The examples of these features are API calls frequency, file and registry manipulations, network operations, and other behavioral features (Maniriho *et al.*, 2023).

Behavioral analysis is efficient; however, there are some issues with it. The malware program is able to detect its behavior being analyzed in virtualized/sandboxed mode and alter its behavior to ensure that it cannot be detected. Moreover, time constraint during execution can hinder the process of capturing all the behaviors of the malware program. (Aboaoja *et al*., 2023)

2.2.3 Machine Learning for Malware Classification

Machine learning algorithms help in malware detection through pattern recognition from the labeled data without applying any manual rules. In the case of supervised learning, the classifiers are trained on the labeled data set and then classify the new instances based on the learned relationships (Bensaoud *et al*., 2024).

Malware detection problem can be described in terms of a binary classification problem, whereby the instances can either be classified as malicious or benign. The performance of the model is based on splitting the dataset into a training and testing set. This is an ideal way of solving the problem of malware detection through machine learning due to the complex nature of malware behavior, which changes constantly, hence rule-based systems are not able to detect new forms of malware (Azeem *et al*.,).

2.2.4 Gradient Boosting Machines

GBM is an ensemble learning technique wherein the algorithm generates better predictions by constructing decision trees incrementally where each new tree corrects the mistakes of the earlier trees. It is different from random forests in the sense that it trains trees independently, gradient boosting enhances the performance of the model through iterative error reduction, enabling the capture of complex non-linear patterns in data (Alhashmi *et al.*, 2023).

The XGBoost (Extreme Gradient Boosting) technique is used in this research, which is an efficient version of gradient boosting that includes regularization, parallel processing, and dealing with missing values. The XGBoost technique, due to its high accuracy and efficiency, is extensively used in malware detection and classification tasks (Chitteti, 2024).

2.2.5 Multilayer Perceptrons (MLP)

Multilayer Perceptron (MLP) refers to an artificial neural network that consists of an input layer, one or several hidden layers, and an output layer. The learning mechanism entails assigning weights to various inputs and then using non-linear activation functions such as ReLU and sigmoid to process the input data through hidden layers (Singh *et al*., 2023).

Training MLPs employs the concept of backpropagation, in which the errors made during predictions are reversed back into the network to adjust the weights and increase performance. MLPs' capability of identifying non-linear relations renders them ideal for behavioral malware detection, which involves recognizing malicious activities through combinations of actions like file, registry, and network activity (Almaleh *et al*., 2023).

2.2.6 Ensemble Learning and Stacking

Ensemble methods are techniques that utilize a combination of machine learning algorithms in order to enhance prediction accuracy and robustness by minimizing errors from single models. Out of all the different types of ensembles, stacking involves combining the results of many base learners to train a meta-learner on how to combine their outputs (Mienye & Sun, 2022).

In this research work, XGBoost and Multilayer Perceptron (MLP) models will be used as the base classifier model and logistic regression as the meta-learner model. This particular approach is able to exploit the benefits of both models as XGBoost model can understand the structure of features and MLP learns the complex non-linear relationships among features, resulting in improved malware classification performance.

2.2.7 Explainable AI in Cybersecurity

Explainable Artificial Intelligence (XAI) is a type of artificial intelligence which makes machine learning predictions interpretable by humans. In cybersecurity, explainability is important since security analysts and end users need to know the reason for classifying the file or behavior as malicious or benign.

XAI techniques can be classified into post-hoc approaches, which involve providing explanations for decisions made by models after predictions, and intrinsic approaches, which take interpretability into consideration while developing models. This research uses an intrinsic rule-based approach that translates detected behaviours into human-readable explanations. By describing actions such as suspicious file modifications, registry changes, or abnormal network activity, the system provides clear and consistent explanations that are accessible to non-technical users (Charmet *et al*., 2022).

2.3 Review of Related Literature

There have been tremendous developments in the process of malware detection from the conventional signature based technique to using machine learning and deep learning techniques that incorporate both static and dynamic analysis. Most current studies are concerned with hybrid solutions that incorporate several feature extraction techniques, as well as ensembles and deep sequence architectures.

Various researches have been conducted on hybrid approaches of malware detection which employ both static and dynamic behavioral features. Alhashmi *et al*. (2023) developed a malware detection approach called HAPI-MDM which employs static features from the Import Address Table with N-gram and TF-IDF methods along with dynamic features generated via sandbox environment in order to detect Windows PE malware. Cosine Similarity was used for generating an extra discriminative feature from the static and dynamic feature vectors while the classification process was done with the help of sequential pipeline which includes XGBoost followed by ANN. The proposed approach obtained a 97.91% accuracy with 10-fold cross-validation and performed better than multiple conventional classifiers. Nevertheless, the paper does not provide clear information about the dataset used and how the output from XGBoost is included into the neural network. Furthermore, the approach is based on sandbox execution and is hard to interpret.

Even more sophisticated ensemble methods have been investigated based on deep feature learning. Maniriho, Mahmood, and Chowdhury (2024) introduced the MeMalDet method, which utilizes the deep autoencoder approach to compress features together with stacking ensembles of random forest, XGBoost, LightGBM, and CatBoost models via logistic regression meta-learner. The proposed model made use of a temporal assessment method in order to simulate the evolution of malware and attained an accuracy of almost 99%. However, there is no specification of temporal splitting criteria or any stacking technique such as out-of-fold learning within the study. This introduces an element of risk with regard to data leakage.

One recent trend that is common among research papers is the application of ensemble learning methods in the classification of malware. Some researchers, such as Dhanya *et al*. (2022), have compared various ensemble models like Bagging, Random Forest, AdaBoost, Gradient Boosting, Stacking, XGBoost, and LightGBM, in both Windows PE malware and IoMT data sets. XGBoost model showed maximum accuracy of 99%. There is lack of methodological transparency in the research because there is no information about the nature of the dataset, splitting between training and test samples, and stacking structure used. There is also lack of any statistical significance tests.

Apart from the effectiveness of detection, explainability has become a new research topic in the field of malware analysis. In this regard, Galli *et al*. (2024) have tested explainability techniques like SHAP, LIME, LRP, and Attention models on LSTM and GRU malware classifiers. The results indicate that SHAP and LRP produce more accurate explanations compared to attention models. Nevertheless, the work is centered around numeric scores for explanation while ignoring the aspect of producing understandable explanations that would help actual decision making of analysts.

Besides ensemble techniques, another machine learning technique extensively utilized for the analysis of API calls in sequence is the deep learning technique. Maniriho, Mahmood, and

Chowdhury (2023) developed an approach called API-MalDetect that uses a hybrid of CNN-BiGRU. This model learns both local behavioral trends and long dependencies and produces outstanding results on various data sets. Nevertheless, this system is not computationally efficient and does not have high interpretability.

Li *et al.* (2022) have developed this idea even further through a proposed deep learning approach, which combines the representation of API phrases, API calls' semantic decomposition, and modelling of relationships between sequence operations. The architecture includes the embedding layer, CNN, BiLSTM, and the multilayer perceptron classifier, attaining 97.31% accuracy. Even though it shows remarkable accuracy, the model is quite complex and does not have the explainability required to be deployed in security-sensitive areas.

A key area of research deals with coping with malware evolution and unseen malware families. In that regard, Jurečková *et al.* (2024) have come up with an ensemble approach where a classifier such as a Multilayer Perceptron is paired with Self-Organizing Maps for clustering of low-confidence samples. The evaluation scheme used by the researchers clearly takes into account temporal splits and unseen malware families, giving an accuracy of 97.21% using the EMBER dataset.

Inspired by such concepts, Zhang *et al.* (2023) presented a hybrid model of CNN-BiGRU that combines contextual semantics embedding, API structure decomposing, and TF-IDF statistics features. The research found out that contextual semantics play a major role in detection. Nevertheless, the proposed method requires manual API representations engineering and does not use API parameters, limiting its adaptability to changing malware behavior.

In like manner, Onyedinma *et al.* (2025) created a hybrid malware detection architecture using a combination of signature based screening, static and dynamic behavioral techniques. This architecture uses a mixture of different kinds of features such as byte n-grams, opcodes, API

calls, registry accesses and network communications in addition to using the concept of weighted voting using Random Forest and XGBoost models. It attained 98.6% accuracy with low false positive rate. Even though successful, the model does not use any other form of ensembling but weighted voting besides lacking explainability module. In contrast to this study, it also uses signature filtering that could affect its efficiency when dealing with zero-day malware. XGBoost obtained the greatest accuracy rate of 99%. Nevertheless, this research lacks methodological clarity concerning data specifics, splitting into training and testing sets, and stacking approach specifics. Moreover, there is no information about statistical significance test and reproducibility which questions the robustness of the reported results.

Almaleh *et al*. (2023) presented a malware detection system using a combination of logistic regression with recurrent neural networks (RNNs) for API call-based behavioral analysis. In the method proposed by the authors, the dynamic sequences of APIs are first modeled with logistic regression and the obtained parameters are then used as input weights to train the RNN classifier. This weight transfer approach was developed with an aim to enhance convergence and take advantage of linear separability while performing initial feature learning. The classifier managed to achieve a level of accuracy close to 98% while working on an unbalanced data set, showing good performance in behavioral malware classification. But the research is lacking some information about the process of dealing with class imbalance in training and its robustness for other malware types.

The MalHyStack hybrid stacked ensemble method was proposed by Roy *et al*. (2023) to perform obfuscated malware detection. Preprocessing methods such as standardization and feature selection using Pearson correlation were applied on the CIC-MalMem-2022 dataset before the classification process. The architecture utilizes a combination of different classical machine learning algorithms such as Random Forest, XGBoost, and Extra Trees along with a deep learning-based meta-learner to make the predictions. The framework obtained an

outstanding binary classification accuracy score of 99.98%, suggesting high discriminatory ability. On the other hand, the near-perfect accuracy results raise issues about the possibility of overfitting or biased data, especially considering the lack of discussion on cross-validation techniques used.

Al-Andoli *et al.* (2023) introduced a stacking ensemble deep learning model based on a combination of heterogeneous neural network learners and a Fuzzy ARTMAP as the meta learner for detecting malware. The learners undergo training through a combination of Backpropagation-Particle Swarm Optimization. The output from all these models is merged using the stacking technique based on fuzzy logic to enhance the classification process. This research shows that there is efficiency in the use of different neural network structures to detect malware. However, the high computational cost associated with merging different deep networks makes it unsuitable for real-time applications.

The MAlign approach was proposed by Saha, Afroz and Rahman (2024), which is based on explainable malware family classification using the principles of sequence alignment methods used in bioinformatics. The method involves transforming raw byte sequences of executable code into nucleotide representations and then analysing those through multiple sequence alignment. These conserved patterns are then used to construct interpretable signatures for classification and explanation. In contrast to other deep learning methods, MAlign is designed to be more interpretable in that it provides human-understandable explanations regarding the similarity of malware samples. The use of sequence alignment methods could hinder scalability when dealing with massive datasets.

Card *et al.* (2024) have studied the use of explainable deep learning models for malware detection through dynamic analysis features alongside post-hoc explanation methods like SHAP, LIME, and Permutation Importance. This research assesses feedforward neural networks (FFNNs) and convolutional neural networks (CNNs) that are trained on behavioral

execution of malware data and provide explanations. Their results indicate that SHAP and Permutation Importance give more reliable results than LIME in some cases. Nonetheless, there is one main limitation that remains in the study – even though the importance of features has been evaluated successfully, the results obtained are still technical and cannot be applied in practice.

2.4 Summary of Literature Review

This section summarizes the key findings from the literature. Presented in tabular format

Table 2.1 Summary of Literature Review

S/N	Author(s)	Contribution	Methodology	Key Limitations
1	Alhashmi *et al.* (2023)	Proposed HAPI-MDM, a hybrid malware detection framework combining static and dynamic API-call analysis using XGBoost and ANN, achieving 97.91% accuracy.	Static and dynamic API features were extracted and combined using cosine similarity before classification through an XGBoost-to-ANN pipeline.	Dataset details not disclosed; limited to Windows malware; vulnerable to sandbox-evasion techniques; limited interpretability; no real-world deployment evaluation.
2	Maniriho, Mahmood & Chowdhury (2024)	Proposed MeMalDet, a memory-based malware detection framework using deep autoencoders and stacked ensemble learning, achieving approximately 99% accuracy.	Memory forensic features were reduced using a deep autoencoder and classified through a stacked ensemble with Logistic Regression as the meta-learner.	Limited reproducibility; possible stacking data-leakage concerns; contribution of individual base learners not rigorously quantified; requires memory dumps; limited explainability.
3	Dhanya, Chitra & Bamini (2022)	Compared multiple ensemble learning techniques and identified XGBoost as the best-performing model with 99% accuracy.	Evaluated Bagging, Random Forest, AdaBoost, Gradient Boosting, Stacking, XGBoost, and LightGBM on malware datasets.	Dataset details insufficient; contribution of stacking architecture not rigorously analysed; behavioural features excluded; no explainability analysis.
4	Galli *et al*. (2024)	Developed an explainability evaluation framework for behavioural malware detection systems.	Behavioural API-call features were analysed using deep-learning models and explained through SHAP, LIME, LRP, and Attention mechanisms.	Technical explanations requiring expert interpretation; no natural-language explanations; no user-centred evaluation; mainly targeted at security analysts.
5	Maniriho, Mahmood &	Proposed API-MalDetect, a CNN-BiGRU framework for malware	Behavioural API-call sequences were processed through	Computationally intensive; robustness against adversarial attacks not

	Chowdhury (2023)	detection from API-call sequences.	CNN and BiGRU layers for automated feature extraction and classification.	evaluated; Windows-only; lacks interpretability.
6	Li *et al*. (2022)	Proposed a deep-learning framework combining API semantics and behavioural patterns for malware detection.	Dynamic API-call sequences were processed using embeddings, semantic modelling, CNN, BiLSTM, and MLP classification.	Requires dynamic execution; susceptible to sandbox evasion; computationally expensive; lacks explainability; limited to Windows malware.
7	Jurečková *et al*. (2024)	Proposed an online malware family classification and clustering framework for emerging malware families.	Static PE features were analysed through Multilayer Perceptron classification and Self-Organizing Map clustering.	Focused only on malware samples; relied solely on static features; moderate clustering performance; limited generalisation.
8	Zhang *et al*. (2023)	Proposed a semantic-aware CNN-BiGRU framework that combines contextual, structural, and statistical API information, achieving 98.28% accuracy.	API-call sequences were represented using Word2Vec embeddings, semantic decomposition, and TF-IDF features before CNN-BiGRU classification.	Manual semantic encoding; ignored API parameters; limited interpretability; Windows-focused; concept drift not fully addressed.
9	Onyedinma *et al*. (2025)	Proposed a hybrid malware detection framework integrating static and dynamic analysis with ensemble machine learning, achieving 98.6% accuracy.	Static and dynamic behavioural features were fused and classified using Random Forest and XGBoost combined through weighted voting.	Contribution of voting mechanism unclear; lacks explainability; Windows-focused; no malware family analysis; deployment challenges remain.
10	Almaleh, Almushabb & Ogran (2023)	Proposed a hybrid malware detection model that integrates Logistic Regression and RNNs for API-call-based behavioural malware detection, achieving 98% accuracy on an imbalanced dataset.	Dynamic API-call sequences were processed using Logistic Regression, whose learned coefficients were used as initial weights for an RNN classifier.	Hybrid model not rigorously compared with constituent learners; relied solely on dynamic analysis; susceptible to sandbox evasion; class imbalance may affect evaluation validity; lacks explainability mechanisms and user-centred interpretation.
11	Roy *et al*. (2023)	Proposed MalHyStack, a hybrid stacked ensemble framework for obfuscated malware detection that combines multiple machine learning classifiers with a deep-	CIC-MalMem-2022 features were preprocessed using standardization and Pearson-correlation feature selection before classification through a	Limited reproducibility due to insufficient architectural details; contribution of individual stacking components not rigorously analysed; lacks explainability; evaluated on

		learning meta-learner, achieving 99.98% binary classification accuracy.	stacking architecture comprising Random Forest, XGBoost, Extra Trees, and a deep-learning meta-learner.	a single dataset; real-world deployment effectiveness not assessed.
12	Al-Andoli *et al*. (2023)	Proposed a stacked ensemble deep-learning framework using heterogeneous neural networks and a Fuzzy ARTMAP meta-learner for malware detection.	Multiple deep neural networks were trained using a hybrid Backpropagation–Particle Swarm Optimization strategy and combined through a Fuzzy ARTMAP stacking layer.	Limited reproducibility due to insufficient architectural details; contribution of individual learners not rigorously analysed; lacks explainability; complex meta-learning architecture may hinder deployment; no support for non-technical interpretation.
13	Saha, Afroz & Rahman (2024)	Proposed MAlign, an explainable malware family classification framework that applies bioinformatics sequence-alignment techniques to raw executable bytes and generates interpretable malware-family signatures.	Raw executable bytes were encoded as nucleotide-like sequences and processed using multiple sequence alignment to identify conserved code regions for malware family classification and explanation generation.	Limited to static analysis; technical explanations requiring expert interpretation; focused on malware family classification rather than malware detection; lacks support for non-technical users; does not investigate hybrid or stacking-based architectures.
14	Card *et al*. (2024)	Investigated explainable deep-learning models for malware classification using SHAP, LIME, and Permutation Importance to improve transparency in dynamic and online malware analysis.	FFNN and CNN models were trained on dynamic malware-analysis features and explained using SHAP, LIME, and Permutation Importance.	Technical explanations requiring expert interpretation; lacks natural-language explanations for non-technical users; does not evaluate hybrid architectures against constituent learners; no stacking framework involving tree-based and neural learners.

2.5 Research Gap

The literature reviewed in Section 2.3 demonstrates that machine-learning-based malware detection has achieved high levels of performance across a wide range of approaches, with many studies reporting detection accuracies above 97%. Despite these advances, three important gaps remain.

The first gap relates to the evaluation of hybrid malware detection models. Although hybrid approaches are increasingly common in the literature, their superiority over individual component models is not always rigorously demonstrated. In many cases, emphasis is placed on overall detection performance rather than on a systematic comparison between the hybrid model and its constituent learners under identical experimental conditions. Consequently, the specific contribution of the hybrid architecture to performance improvement remains insufficiently established.

The second gap concerns explainability for non-technical users. Most studies focus primarily on prediction accuracy and provide little or no explanation for classification decisions. Where explainability is considered, explanations are typically presented through feature importance scores, relevance maps, or other technical outputs intended for security analysts and machine-learning practitioners. As a result, there remains a lack of explainability mechanisms that translate model decisions into simple, human-readable descriptions that can be easily understood by non-technical users.

The third gap relates to the architectural design of hybrid malware detection systems. The reviewed studies employ various ensemble and hybrid strategies; however, none combines a tree-based learner and a neural learner within a stacking framework using logistic regression as the meta-learner. Such an architecture offers the potential to exploit the complementary strengths of different learning paradigms through a learned combination strategy rather than relying on fixed aggregation rules.

Taken together, these gaps, the limited rigorous evaluation of hybrid models against their base learners, the absence of plain-language explainability for non-technical users, and the lack of a stacking architecture that combines tree-based and neural learners through a logistic regression meta-learner, provide the motivation for the present study.

CHAPTER THREE

SYSTEM ANALYSIS AND DESIGN

3.0 Introduction

This chapter discusses the analysis and design of the hybrid malware detection system proposed in this thesis. It first analyzes some existing malware detection methods, its limitations and highlights the necessity of the proposed system. System requirements are then identified followed by system design methodology and UML diagram based system models. Detailed design specification including input and output design, database design, system design is discussed in the final part of this chapter.

3.1 System Analysis

3.1.1 Analysis of the Existing System

Presently, malware identification depends mainly on the use of signature-based anti-virus software and manual rule-based techniques. Signature-based software works through comparing files to databases containing signatures of malicious programs, which include file hashes, byte patterns, and other behavioral signatures. These tools are commonly used in both domestic and corporate settings. The use of manual static and dynamic analysis of suspicious files is another technique often employed in the field of cybersecurity. This involves either analyzing file content without executing them or studying their behavior in a controlled environment.

3.1.2 Limitations of the Existing System

The main problem with signature-based approaches is that they are reactive and can detect only those threats that have been seen before, leaving the system susceptible to new forms of malware. Nowadays, malware authors use obfuscation, polymorphism, and metamorphism to modify the appearance of the malware without altering its functionality. Improvement is provided by the rule-based heuristic method through identifying any suspicious patterns of behavior, but it demands manual creation of the rules by security professionals and results in false positives, which lead to

alert fatigue. The manual inspection process is comprehensive; however, it does not cope with large volumes of new samples generated every day and is not an option for those without necessary skills and expertise. Moreover, available software products lack explanation for their decisions and give only numeric judgments or verdicts.

3.1.3 Analysis of the Proposed System

This malware detection system is able to overcome these shortcomings through three major contributions. Firstly, the detection is achieved using behavioral features obtained from dynamic sandboxing analysis, which allows detection of obfuscated and new variants of malware based on their runtime behavior and not on their static features. Secondly, it utilizes the combination of gradient boosting machine and multilayer perceptron using stacking ensemble that allows for higher accuracy detection without false positives. Third, it offers plain language descriptions based on rules that translate behavioral observations into descriptions understandable by a nontechnical user. The system takes a Windows PE file as input, runs it inside the CAPE sandbox, creates a 42-dimensional feature vector, applies the hybrid ensemble for classification, and outputs the results using a web based interface.

3.2 System Requirements Specification

3.2.1 Functional Requirements

1. The system will be able to take Windows PE file submissions using a web-based interface.
2. The system will be able to analyze submitted files using CAPE’s sandbox for dynamic behavioral analysis.
3. The system will be able to extract a 42-dimensional feature vector from CAPE's report, which consists of 25 behavioral features and 17 static PE features.
4. Each submitted file will be classified as either malicious or benign using the hybrid stacking classifier with a deployment threshold of 0.96.

5. The system shall produce results from GBM, MLP, and hybrid meta-classifier in terms of their estimated probabilities of being malicious.

6. The system shall produce natural language behavioral explanations for each classification result.

7. The system shall maintain all analyses as a job record containing the file name, SHA256 hash value, timestamp, verdict, and reasoning.

8. The system shall include an endpoint for checking the system's health status.

3.2.2 Non-Functional Requirements

1. Performance: Feature extraction and inference shall take less than one second per sample, not counting sandbox analysis time.

2. Reliability: The system shall manage CAPE timeouts and analysis failures gracefully without crashing.

3. Usability: Observations in plain language shall be written at a level that is accessible to non-technical users without security knowledge.

4. Reproducibility: A fixed random seed of 42 will be used for all model training with the artifacts being versioned and kept separate from the code base.

5. Maintainability: The system design shall be modular such that each component can be upgraded independently.

6. Security: Files uploaded into the system will be run in a sandboxed environment and then deleted from the host environment.

3.3 System Design Methodology

3.3.1 Justification for Methodology

This design has been done using Object-Oriented Analysis & Design (OOAD), where the entire system is represented in terms of interacting objects with well-defined roles and interfaces. The reason for choosing OOAD is that the different elements of the system have clear object-based

implementations in the form of the FeatureExtractor which generates vectors from reports, ModelService which implements inference functionality, CAPEClient which takes care of sandbox communication, and Job which models the persisted state of analysis. This modular structure supports independent development and testing of each component and makes the architecture straightforward to extend.

3.3.2 Method of Data Collection

The malware dataset was collected from the freely accessible Avast-CTU CAPEv2 dataset, containing 48,976 labeled Windows PE malware instances belonging to ten malware families, along with sandbox analysis reports for each of them. Benign instances were collected by the researcher by running 203 benign Windows executables in the CAPE sandbox environment, ranging from productivity tools, system utilities, development tools, media players, document viewers, to software installers. Each instance of the benign dataset was confirmed to be clean using VirusTotal.

3.4 System Modelling Using UML

3.4.1 Use Case Diagram

The figure 3.1 shows the use case diagram. There are two actors in the system: the End User, who is responsible for uploading files and getting results, and the System Administrator, who checks the status of the system. Uploading file causes a sequence of included use cases including sandboxing, feature extraction, classification, and explanation generation.

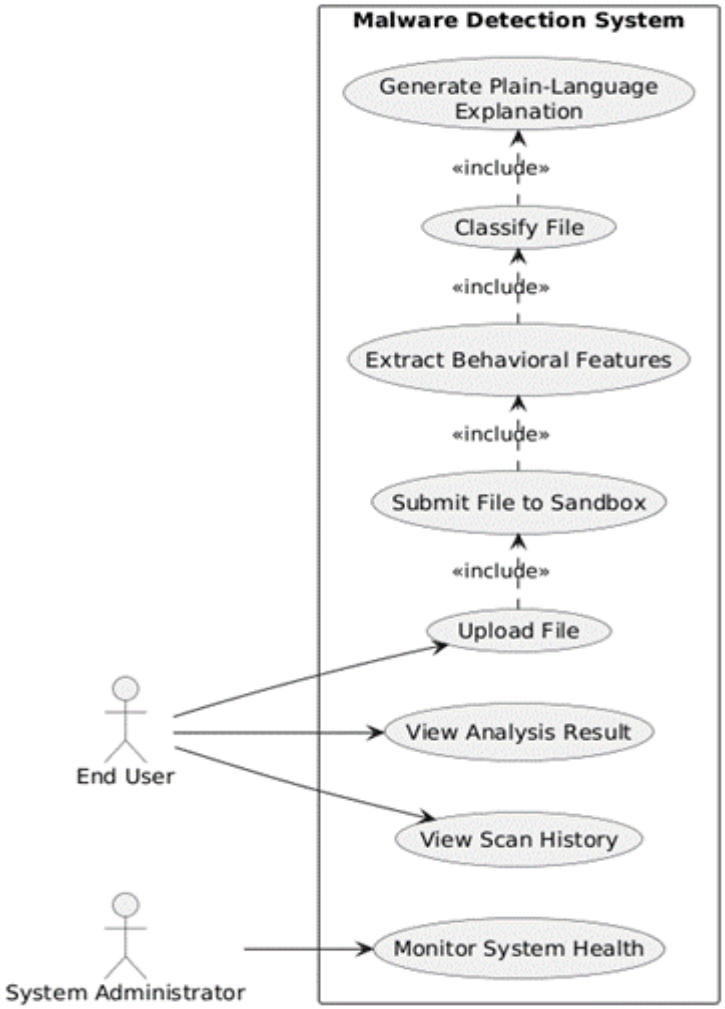

Figure 3.1: Use Case Diagram

3.4.2 Activity Diagram

Activity diagram of file analysis process is shown in Figure 3.2. Workflow starts from user uploading a PE file. SHA256 hash is calculated by API, which also creates a pending job entry and sends the file to CAPE. Task completion is periodically checked and after that, JSON report is downloaded along with the 42-dimensional feature vector. The malicious probabilities produced by the GBM and MLP are combined together by the logistic regression model meta-learner to create a hybrid probability value. The 0.96 threshold is used to make the binary decision, the explainability module creates language-based findings, and then the output is given back to the frontend.

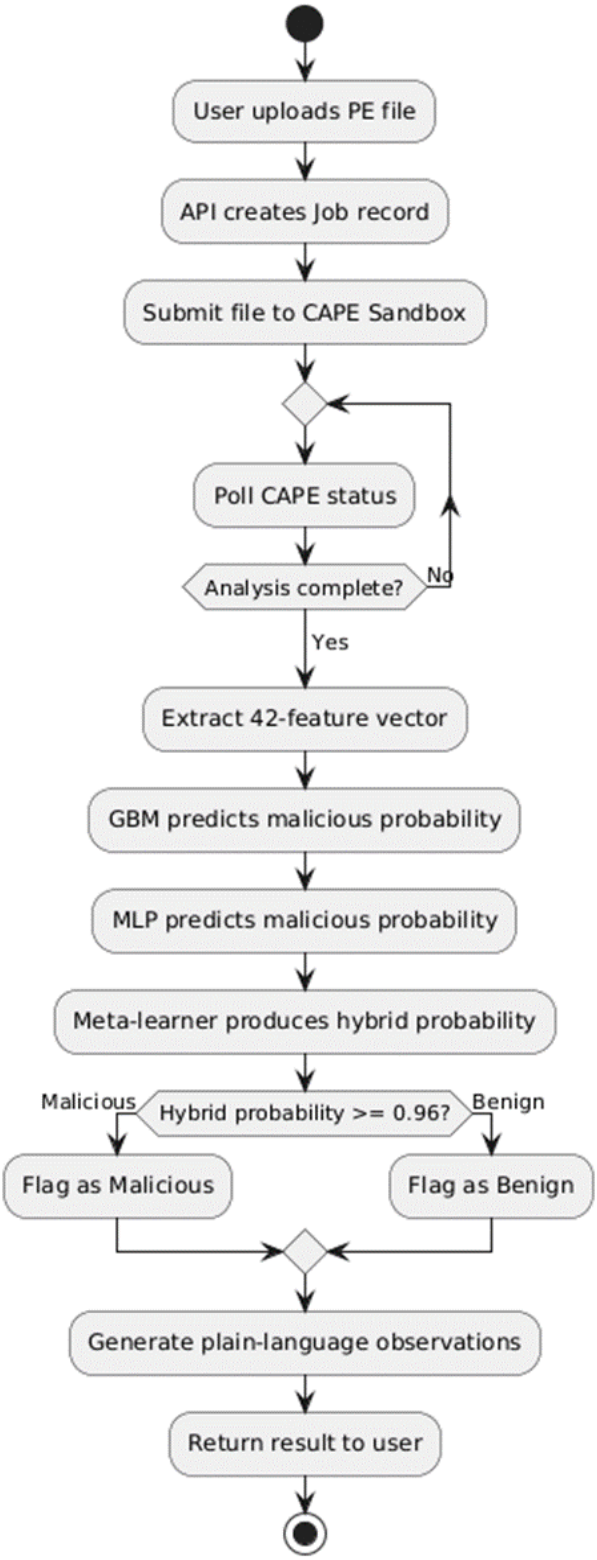

Figure 3.2: Activity Diagram

3.4.3 Class Diagram

The class diagram is shown in Figure 3.3. ModelService is the main inference module, where there are links to the FeatureExtractor and MLP. It controls the entire process of prediction. CAPEClient provides JSON files for ModelService. The Job is the database model that holds every analysis done, and PredictionResponse is an API response model consisting of verdict and a set of Observation objects from the explainability module.

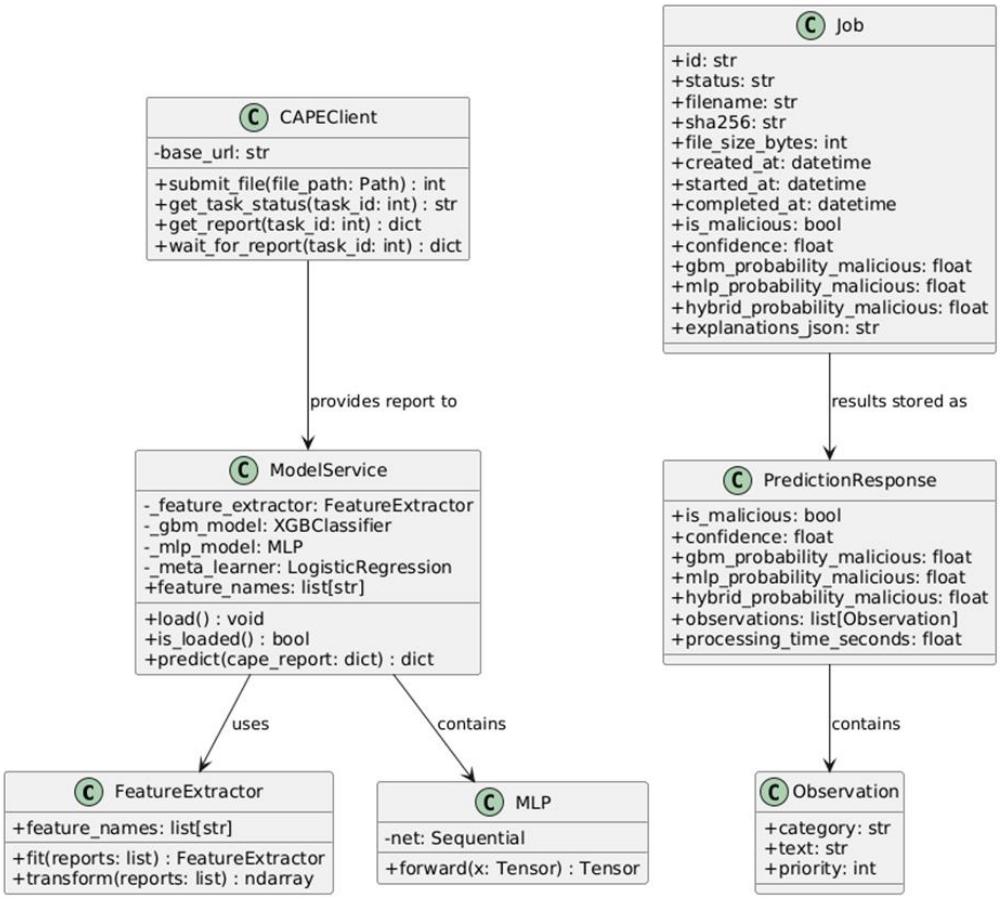

Figure 3.3: Class Diagram

3.5 System Design

3.5.1 Input Design

There are two possible inputs accepted by the system. First of all, there is a Windows PE file that should be uploaded using either drag-and-drop or file picker in the web user interface. Once it is received, the API calculates its SHA256 hash value and stores it in the temporary file for CAPE submission that will be removed after prediction is completed. Second of all, there is a raw CAPE JSON report submitted to the /predict endpoint using POST request.

3.5.2 Output Design

The first form of output is the structured prediction output, which consists of the binary verdict, hybrid confidence score, probability of each model's predictions of being malware, and a list of observations sorted according to their category as either suspicious, benign or something else. This is presented in the frontend results pane with color-coded verdict and models breakdown. The second output form is the saved job data into the SQLite database.

3.5.3 Database Design

The system makes use of a SQLite database which is managed using SQLAlchemy 2.0. All analysis entries are stored in a single table called jobs which contains the following columns: a UUID-based primary key, status representing the state of the job, filename, SHA256 hash, size of the file, three timestamps, error message column, binary verdict, confidence value, three model probability values, and a list of explanation strings in JSON format.

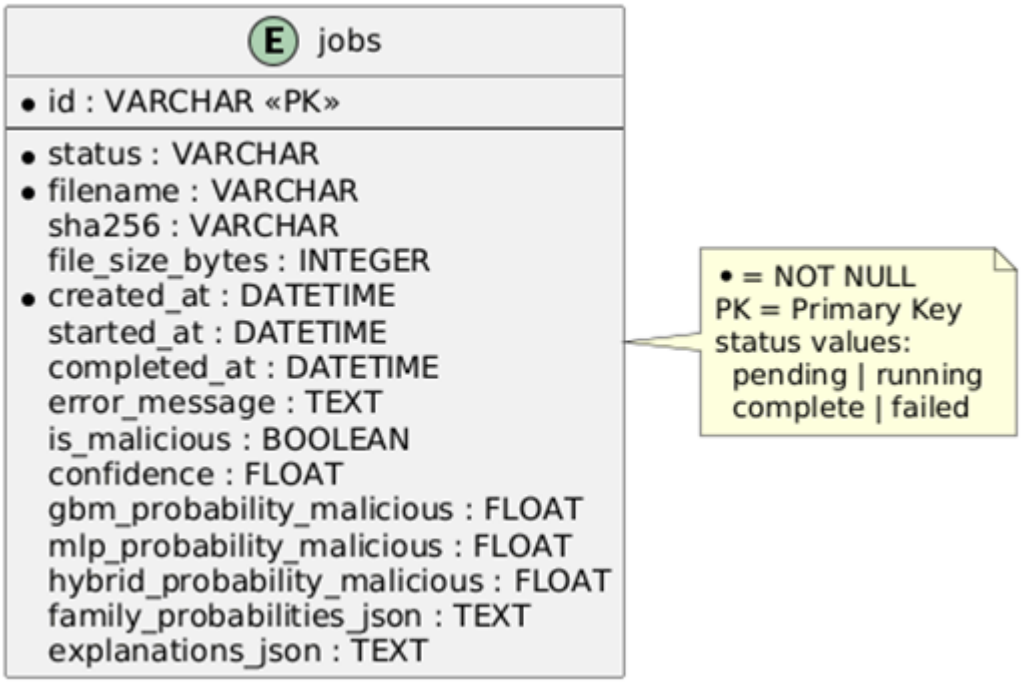

Figure 3.4: Database Design

3.5.4 System Architecture Design

The system is implemented using a three-tier client-server architecture which consists of React front-end, FastAPI back-end and CAPE sandbox services. Communication between the front-end and back-end occurs using HTTP protocol. The back-end performs file upload processing, model inference, database persistence, and CAPE communication. The sandbox tier is deployed in an independent Ubuntu machine that provides dynamic analysis using CAPE services via its REST API.

In the backend, the design of the system follows the structure of modular architecture, where the router layer is responsible for parsing the HTTP request, the ModelService layer does the machine learning inference, the CAPEClient layer does the communication with the sandbox, the explainability layer produces the observations, while the database layer uses SQLAlchemy for the data persistence.

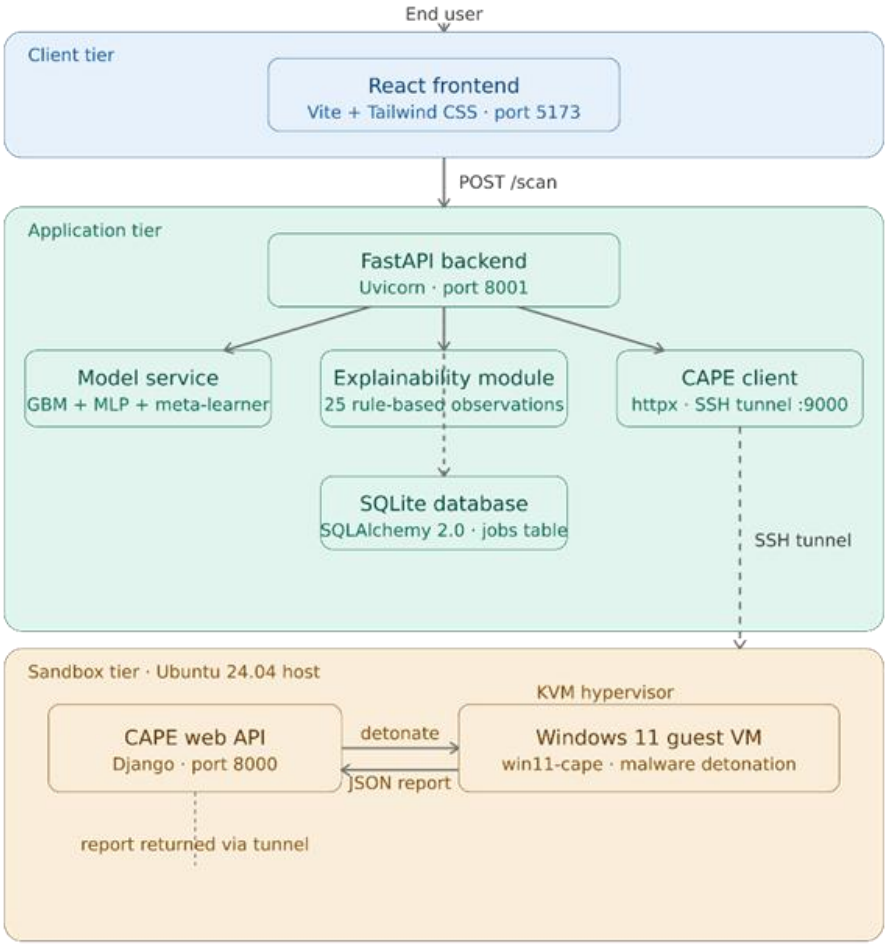

Figure 3.5: System Architecture Design

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.0 Introduction

This chapter describes how the hybrid malware detection system developed in this research was created and implemented. It explains the process of collection and preparation of malware and benign program samples, extraction of Portable Executable features related to their behavior and static properties, and construction of machine learning models for malware detection.

The chapter also elaborates on the training and deployment of the three classifiers namely XGBoost Classifier, MLP Classifier, and the stacking ensemble classifier. The chapter also describes how these classification models were incorporated in a web-based detector that was built using the FastAPI framework. Furthermore, the chapter describes the incorporation of a rule-based explainable artificial intelligence module to generate user-friendly detection reports. Finally, the chapter discusses the testing, performance evaluation, and analysis of the system developed. Comparing the results from the various models helps in assessing the effectiveness of the proposed hybrid method in identifying malware, with minimal instances of false detection.

4.1 Development Environment and Tools

4.1.1 Programming Language and Libraries

The hybrid malware detection system was implemented using Python as it is widely supported for machine learning, data analysis, and the development of web applications. For the machine learning part, it utilized XGBoost for gradient boosting classifier, PyTorch for Multilayer Perceptron (MLP) model, and Scikit-learn for data pre-processing, model evaluation, cross-validation, and logistic regression meta-learner in the stacking ensemble approach.

Feature extraction and data processing were carried out with the help of Pandas and NumPy, while the processed data sets were saved in Parquet file format. The web-based detection tool was built with the help of FastAPI, with SQLAlchemy acting as the database library and Uvicorn as the application server. Other libraries such as Matplotlib and Seaborn were also used for visualization and performance evaluation.

4.1.2 Development Platform and Hardware

Model training and development were done using a Dell Latitude 5491 with Microsoft Windows 11 Pro operating system. The computer had an Intel Core i7-8850H processor and 8 GB RAM. The machine Learning Models were all developed using CPUs without GPUs.

The behavioral data collection was done using the CAPE sandbox which was located in an isolated environment running Ubuntu 24.04 LTS with the help of KVM/QEMU virtualization. The malware and benign binaries were run in a secure Windows virtual machine environment, creating the behavioral reports required for the features extraction.

4.2 Dataset Description and Preprocessing

4.2.1 Dataset Description

In this research work, the use of the Avast-CTU CAPEv2 malware dataset was employed. This is an open-source dataset that has a total of 48,976 samples of Windows Portable Executables (PE) malware belonging to ten different malware families such as Emotet, Trickbot, Qakbot, Dridex, Agent Tesla, Formbook, Remcos, Lokibot, Nanocore, and Guloader. These samples were examined in the CAPEv2 sandbox, and their information is saved in the form of JSON reports, containing information about file system activities, registry operations, network communications, and process behavior.

In order to facilitate the analysis of malware binaries, 203 samples of benign software were collected and studied using the same CAPE sandboxing setup. The samples consisted of legitimate Windows software including office suites, system tools, media players, developer

tools, and software installation utilities. Every sample was confirmed to be benign prior to analysis to ensure the reliability of the dataset.

The total number of samples in the dataset was 49,179, including 48,976 malware samples and 203 benign samples. An 80:20 split was performed to form the training and testing datasets while preserving the original class distribution. The number of samples in the training dataset was 39,343 and that of the test dataset was 9,836.

4.2.2 Feature Extraction

A set of features was extracted from the CAPE sandbox JSON reports that described the behavior as well as structure of each sample. There were 42 features extracted from each sample.

The first set of features comprised 25 behavioral features that describe the actions carried out during execution. They include file operations, registry activities, network connections, DNS request, and process creations. Some additional features such as log and ratio features were included to better capture activity patterns.

The second set of features used was made up of 17 static PE features which were derived from data present in the sandbox reports. Some of these features were the file size, section characteristics, entropy values, digital signature details, and other PE header properties.

The final feature representation contains 42 fully populated behavioral and static features altogether.

4.2.3 Preprocessing

Following the feature extraction phase, the dataset was stored in the Parquet file format so that the data could be easily loaded and processed while training the models.

Since the data had an extreme imbalance, class weighting was used during the model training process rather than resampling. This ensured that both malware and benign samples contributed appropriately to the learning process. Scaling of features was done for the Multilayer

Perceptron (MLP) model using standard scaling techniques . Scaling of features was not done for the XGBoost model since the decision tree models are usually not sensitive to feature scales. The preprocessed dataset was used to train and test the XGBoost, MLP, and ensemble stacking models.

4.3 Model Architecture and Training

4.3.1 XGBoost Classifier

The first model that was employed for this experiment is XGBoost classifier. This type of model was chosen due to its high efficiency of dealing with structured data and its capacity to establish complex dependencies between variables. The model was trained on the basis of 42 features that were extracted from the training dataset.

In order to deal with the class imbalance problem of malware and benign samples, class weighting was used in the training process. The algorithm was set up using multiple decision trees to classify malware from benign software data instances. This trained model acted as the first base learner of the proposed hybrid detection system.

4.3.2 Multilayer Perceptron (MLP)

The second model that was tested was the Multilayer Perceptron (MLP), which utilized a feed-forward neural network approach. It comprised one input layer, two hidden layers, and an output layer to perform binary classification.

The extracted features were normalized prior to the training process to improve learning performance. The network was trained using the Adam optimizer with the binary cross-entropy loss function. The class weights were employed in order to minimize the effect of the unbalanced data distribution. Once trained, the MLP became the second base learner of the hybrid detection system.

4.3.3 Hybrid Stacking Ensemble

The hybrid malware detection model designed in the proposed work used the stacking ensemble method with the XGBoost and MLP learners. The output produced by the base learners was used as an input to the logistic regression meta-learner.

In order to avoid data leakages and enhance generalization, five fold cross validation was employed in creating out of fold predictions in order to train the meta-learner. The logistic regression model learnt how to integrate both learners' strengths in making the final classification prediction.

In the test phase, each sample was first evaluated by the XGBoost and MLP algorithms. These predictions were fed into the meta-learner, which returned the final detection result for the malware. This approach was designed to leverage the complementary strengths of tree-based learning and neural network learning, thereby improving overall detection performance.

4.4 System Implementation and Modules

The malware detection system implemented in this paper comprises five main components: the machine learning training pipeline, the back-end application, CAPE sandbox integration, the explainability module, and the graphical user interface. Together, these modules provide the functionality of automated malware analysis, classification, and results visualization. The general architecture of the solution is based on a client-server approach, where the client interacts with the software via a web interface, whereas the back-end manages feature extraction, classification, model inference, and communication with the sandbox.

4.4.1 Training Pipeline

The training pipeline is tasked with producing the machine learning models that will be used by the detection algorithm. It takes in the extracted features dataset and uses it to train the XGBoost and Multilayer Perceptron (MLP) algorithms before forming the stacking ensemble model with logistic regression meta learner.

Once the training phase is complete, the feature extractor along with the trained models and other supporting components, is stored and becomes available for the backend application. These components get loaded during system initialization and are used to predict the malware during operation.

4.4.2 Backend Application

The backend application was built using FastAPI architecture. This acts as the main processing entity in the system and controls the interaction between the user interface, the machine learning algorithms, and the CAPE sandbox environment.

Once a malware sample is uploaded, the backend obtains the analysis report from it, selects the necessary features, runs the prediction algorithm on them, and finally produces the classification outcome. It is also responsible for maintaining scan data and system output information for future use.

4.4.3 CAPE Sandbox Integration

The CAPE sandbox integration module allows the automation of analysis of the executable files. The file samples that are uploaded are analyzed using the CAPE sandbox by executing them in an isolated virtual environment. As the sandbox executes them, it collects information about their behavior, which includes actions performed during their operations like file manipulation, registry modification, network communications, and process creation.

Upon completion of the analysis, the generated JSON report is sent to the machine learning pipeline for analysis.

4.4.4 Explainability Module

In order to enhance transparency and understandability for the user, an explainability module was added to the system. This module makes use of a set of rule-based observations that are formulated on the basis of the extracted feature values to generate human-readable explanations of the model's decision.

The explanations generated point out the features which are either suspicious, benign, or informational. This enables the user to identify the behavior that made the decision without any need for in-depth cybersecurity knowledge.

4.4.5 User Interface

The development of a web-based user interface provided the means of accessing the system in an easy way. The user interface enables the users to upload executable files for analysis and view the output in a clear and organized manner.

Once the processing phase is finished, the user interface shows the classification outcome, confidence details, and the reasoning produced by the explainability module. This allows users to make a quick judgment on whether the file is classified as a threat or not, and understand the reasons behind the system's decision.

4.5 Testing and Evaluation

4.5.1 Evaluation Metrics

The performance of the developed malware detection system was evaluated based on five standard criteria for binary classification: Accuracy, Precision, Recall, F1-score, and False Positive Rate (FPR).

The Precision metric evaluates the percentage of files identified as malicious that really were, whereas Recall shows how well the model is able to detect the malware. F1-Score is the measure used to balance Precision and Recall. The accuracy is used to measure the overall proportion of correctly classified samples, while the False Positive Rate is the proportion of benign files incorrectly classified as malware.

The above metrics were chosen the detection system for malware should be capable of detecting the malware effectively while ensuring that there are no false alarms that may disrupt legitimate users and system administrators.

4.5.2 System Testing

The designed models were tested on a different testing dataset which was not part of the training process. This dataset contained 9,836 samples with malware and benign applications and was created using an 80:20 stratified split of the entire dataset.

In order to determine the most suitable threshold for operation, probability thresholds were evaluated at various levels. The process revealed that the most suitable threshold for deployment was 0.96 as it provided the best performance in terms of malware detection and reducing false positives. This threshold was used during the final evaluation of the XGBoost, MLP, and the hybrid ensemble model.

Additional analyses were done to investigate the behavior of the models, to compare the performance of the individual base learners with the hybrid ensemble model, and to support the interpretation of the experimental results in the next section.

4.5.3 User Interface and Functional Testing

Functional testing was performed to ensure proper working of the whole system. The test was performed for the following operations:

File uploading, interaction with the CAPE sandbox environment, report generation, feature extraction, model prediction, explanation generation, and result display via the user interface.

The system successfully processed submitted files and generated classification results along with human readable explanation of the detected behaviors. Testing proved that communication between the system components worked well and that the prediction output was accurate for the user.

Performance observations revealed that most of the processing time was spent on sandbox analysis, whereas feature extraction and machine learning predictions were performed within a very short time. This indicates that the developed system is able to classify malware efficiently after behavioral analysis is performed.

Figure 4.1: Upload Page

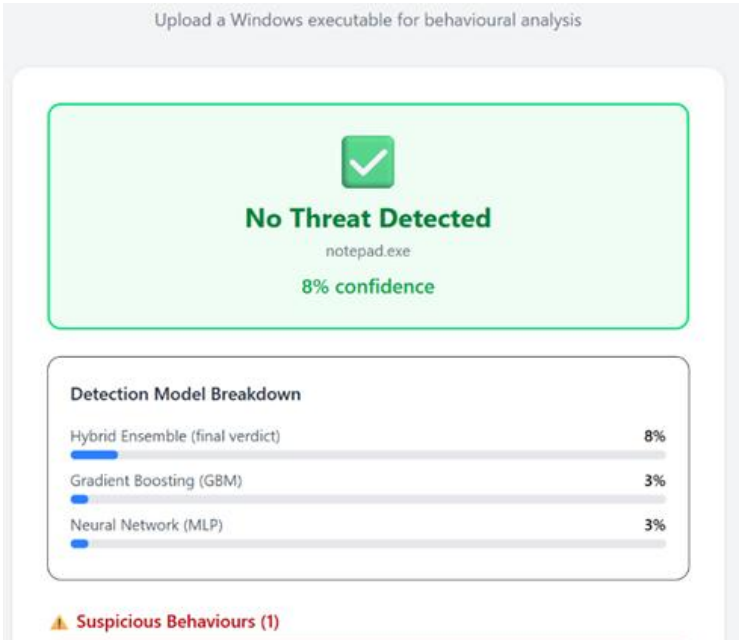

Figure 4.2: Result Page

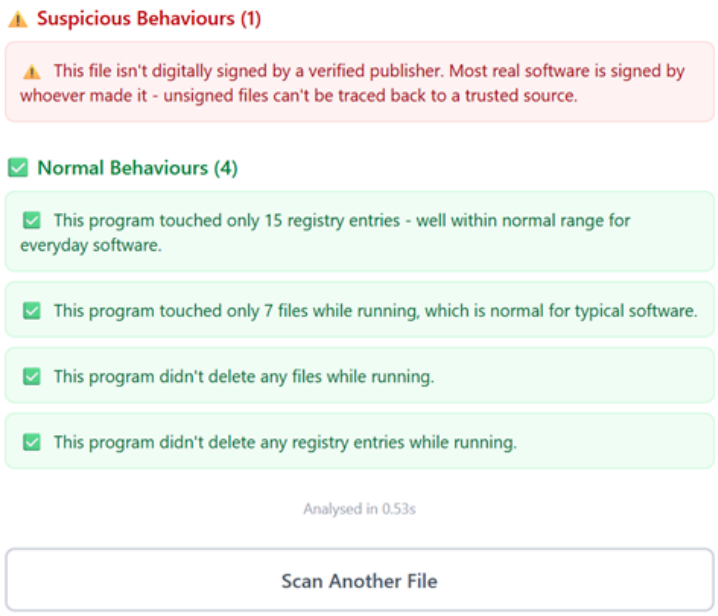

Figure 4.3: Explainability Page

4.6 Results Presentation and Analysis

4.6.1 Evaluation Results

The performance of the models, including the XGBoost, Multilayer Percetron(MLP), and Hybrid Stacking Ensemble model, was assessed using the test data set, which includes 9,836 samples. Table 4.1 shows the classification results obtained for the two different thresholds: 0.50 and 0.96.

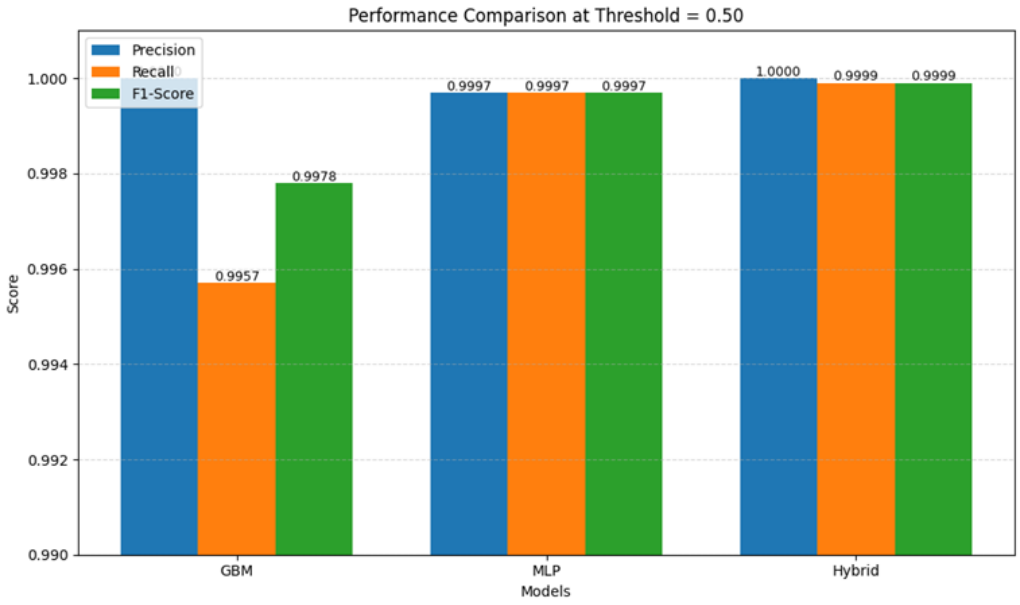

Figure 4.4: Performance Comparison at 0.50 Threshold

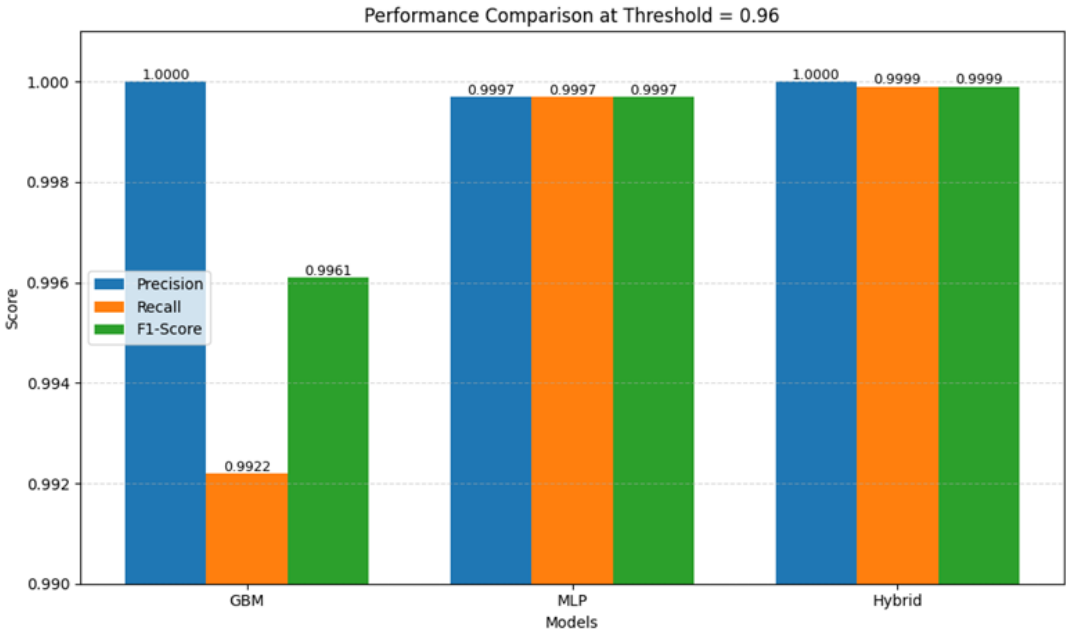

Figure 4.5: Performance Comparison at 0.96 Threshold

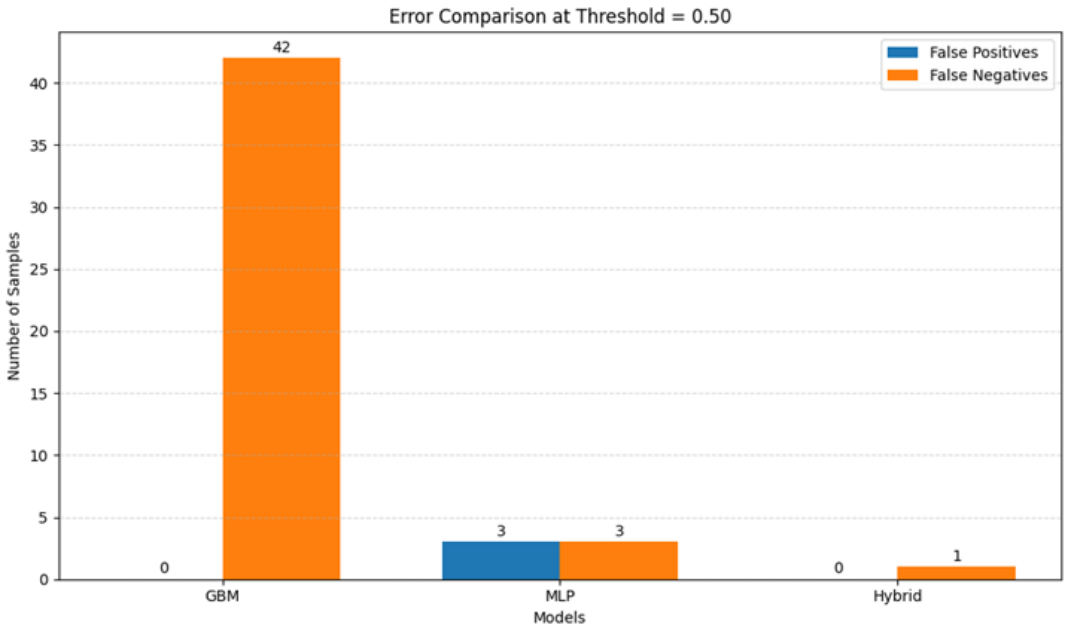

Figure 4.6: Error Comparison at 0.50 Threshold

According to the results, the three models had good performance in detection. Nevertheless, the Hybrid Stacking Ensemble was able to give the best performance of the three models. With the deployment threshold at 0.96, the hybrid model achieved a Precision of 1.0000, Recall of 0.9999, an F1-score of 0.9999, and False Positive Rate of 0.0000. The model did not generate any false positives and just one false negative on the test dataset.

The hybrid algorithm performed better compared to the individual algorithms used because it combined the strengths of the two algorithms. The XGBoost algorithm had no false positives, but it could not detect further malware samples. The MLP model had a very good recall rate, but it generated some false positives. This model balanced the strengths of the two algorithms and performed well.

4.6.2 Discussion of Results and System Performance

From the results obtained, it can be seen that the suggested hybrid approach performs very well in detecting malware. In fact, the stacking ensemble performed better than each model alone

by being able to achieve near-perfect malware detection without any false positives at the selected deployment threshold.

These results demonstrate that XGBoost and MLP learned complementary information from the behavioral and static extracted features. The combination of outputs of these two models by means of logistic regression meta-learner, the system was able to reduce missed detections while preserving the reliability of classification decisions.

In addition, the threshold analysis also confirmed the significance of choosing the right decision threshold. The threshold of 0.96 yielded the best results when weighing up detection of malware and reduction of false positives, making it suitable for practical deployment where minimizing false alarms is essential.

All in all, the results show that the suggested hybrid malware detection system satisfies all the goals of this study. In particular, the processes of feature extraction, the architecture of the hybrid model, the framework used for evaluation, and the part related to the explainability of the results have led to the creation of an extremely precise malware detection algorithm. The results also confirm that combining multiple machine learning techniques can improve malware detection performance compared to using a single classifier alone.

CHAPTER FIVE

SUMMARY, RECOMMENDATIONS AND CONCLUSION

5.1 Introduction

In this chapter, the important aspects of the study are brought out concisely in the form of a summary, conclusions are drawn from the evaluation done in Chapter 4, and recommendations are made for further research. The aim and objectives set out in Chapter 1 are revisited and the level of achievement of each is analyzed based on what was done in previous chapters.

5.2 Summary

Chapter 1 set the background on malware being a rising concern in the domain of cybersecurity and pointed out three challenges faced by the current methods of detection: the drawbacks of single model behavioral detection, lack of evaluation for hybrid models, and unavailability of understandable explanation for such hybrid models for non-technical users. The aim of this research was to develop a hybrid malware detection system using behavioral features, with six specific objectives operationalizing this aim.

Chapter 2 covered the theoretical background of behavioral analysis, gradient boosting machines, multilayer perceptron, stacking ensemble, and explainable artificial intelligence, followed by a critical analysis of fourteen literature reviews. Three main research gaps were identified: lack of rigorous comparative performance evaluations against base learners, lack of plain-language explanations for non-technical users, and absence of a stacking model that integrates tree-based and neural learners using logistic regression as a meta-learner.

Chapter 3 described the system analysis and design where the limitations of current detection approaches, functional and non-functional requirements, and the system model using use cases, activity diagram, and class diagrams were discussed. Input/Output designs, database design, and system architecture designs were specified.

The implementation and evaluation of the system is discussed in Chapter 4, including the processing of the data, 42 feature extraction process, training of the models, implementation of the system, and the results of the evaluation. At the threshold of 0.96, the hybrid stacking ensemble model yielded a precision of 1.0000, recall of 0.9999, and a false positive rate of 0.0000 resulting in zero false positives and one false negative among the 9,836 samples in the test set, strictly outperforming both base learners under identical experimental conditions.

5.3 Recommendations

The following are the recommendations made for future work based on the results of this research.

First, future work should test the effectiveness of the system against malware that has been adversarially altered and sandbox evasion techniques. This work has used samples of malware collected from the Avast-CTU dataset collected between 2017 to 2019, and the system has not been tested against malware that was created in such a way as to be able to detect the presence of sandbox or to delay malicious behavior. Assessing robustness under adversarial conditions is a natural and important extension of this work.

Second, the benign data sampling should be increased both in quantity and variety. In the current research, there were 203 benign data samples used, resulting in an extreme ratio of 241:1. A larger and more representative benign dataset would improve the reliability of false positive estimates and better reflect the distribution of legitimate software encountered in real-world deployment.

Third, the explainability component could be enhanced by adding post-hoc explanations based on SHAP and LIME on top of the current rule-based one, thereby allowing for a comparison between the two approaches in terms of quality of explanation and ease of understanding. The current work deliberately focused on rule-based explanations for accessibility, but a user study

comparing rule-based and post-hoc explanations for non-technical users would provide empirical grounding for this design decision.

Fourth, it is necessary to study how well the system generalizes to other malware families that are not present in the Avast-CTU database. These ten malware families, although common, are not an exhaustive list of all possible malware types out there. A test on a more extensive malware database would offer a better understanding of the system's practical efficiency.

Fifth, the MLP calibration problem presented in Section 4.6 – where the neural network gives overconfident probability scores to a few benign samples. This can be corrected using probability calibration methods like Platt scaling or isotonic regression on MLP's raw outputs.

5.3 Conclusion

This research aimed to design a hybrid malware detection system using behavioral features, with the purpose of bridging the gaps outlined in the current literature. The evaluation results reported in Chapter 4 demonstrate that this aim was achieved. Each of the six specific objectives was met as follows.

Objective 1, to review existing malware detection techniques, was accomplished by conducting the literature review provided in Chapter 2, which reviewed fourteen relevant studies pertaining to hybrid approaches, deep learning, evaluation techniques for ensembles, and explainable methods, and identified the three research gaps that have motivated this study.

Objective 2, which involved the extraction of behavioral features from sandbox execution reports, was achieved through the development of a Custom FeatureExtractor which generates a 42-dimensional feature vector made up of 25 behavioral features and 17 static PE features derived from the CAPEv2 JSON report, used uniformly on 49,179 malware and benign samples.

Objective 3, to design and develop a hybrid detection model by stacking GBM and MLP, was achieved using the binary classification pipeline, where the two base learners, namely XGBoost

and PyTorch MLP, were combined using a logistic regression meta-learner. The model was designed in such a way that its architecture takes advantage of the complementary inductive biases of tree-based and neural learners.

Objective 4, to assess the hybrid model relative to its base learners in a rigorous manner, was achieved via an extensive assessment process involving per-model assessment at two different thresholds, threshold tuning assessment, calibration assessment, and per-family recall assessment. The hybrid ensemble identified 75 out of 76 misclassified malicious samples of the GBM at a deployment threshold of 0.96, with zero false positives, thereby showing significant enhancement over the two base models.

Objective 5, to develop a rule based explainability module, was achieved by developing 25 rules in the explainability module that interpret the 42-dimensional feature vector as suspicious, normal, or other simple language observations. The module creates prioritized and understandable explanations that do not require any security knowledge to comprehend.

Objective 6, to integrate the trained models into a web application, was accomplished by implementing the backend using FastAPI, where binary prediction is available at REST API endpoints, CAPE sandbox integration to process file submissions and report retrieval, and frontend using React for drag-and-drop file uploads and results presentation.

Overall, the hybrid stacking ensemble constructed in the current research shows high performance on the Avast-CTU binary classification task, with the results of the evaluation proving the superiority of the stacking approach to its base classifiers empirically. The rule-based explainability module solves the problem recognized in the existing literature, providing explanations of the detection process in layman's terms. Thus, it is proven that hybrid behavioral malware detection can be both highly accurate and interpretable within the constraints of an undergraduate research project.

REFERENCES

Aboaoja, F. A., Zainal, A., Ali, A. M., Alsolami, F. J., & Rassam, M. A. (2023). Dynamic extraction of initial behavior for evasive malware detection. *Mathematics, 11*(2), 416.

Al-Andoli, M. N., Tan, S. C., Sim, K. S., Goh, P. Y., & Lim, C. P. (2023). An ensemble deep learning classifier stacked with fuzzy ARTMAP for malware detection. *Journal of Intelligent & Fuzzy Systems, 44*(6), 10477–10493.

Alhashmi, A. A., Darem, A. A., Alanazi, S. M., Alashjaee, A. M., Aldughayfiq, B., Ghaleb, F. A., Ebad, S. A., & Alanazi, M. A. (2023). Hybrid malware variant detection model with extreme gradient boosting and artificial neural network classifiers. *Computers, Materials & Continua*, *76*(3), 3483–3498.

Alhashmi, A. A., Darem, A. A., Alashjaee, A. M., Alanazi, S. M., Alkhaldi, T. M., Ebad, S. A., Ghaleb, F. A., & Almadani, A. M. (2023). Similarity-based hybrid malware detection model using API calls. *Mathematics, 11*, 2944.

Almaleh, A., Almushabb, R., & Ogran, R. (2023). Malware API calls detection using hybrid logistic regression and RNN model. *Applied Sciences, 13*(9), 5439.

Azeem, M., Khan, D., Iftikhar, S., Bawazeer, S., & Alzahrani, M. (2023). Analyzing and comparing the effectiveness of malware detection: A study of machine learning approaches. *Heliyon, 10*(1), e23574.

Bensaoud, A., Kalita, J., & Bensaoud, M. (2024). A survey of malware detection using deep learning. *Machine Learning with Applications, 16*, 100546.

Card, Q., Simpson, D., Aryal, K., Gupta, M., & Islam, S. R. (2024). Explainable deep learning models for dynamic and online malware classification. *arXiv Preprint arXiv:2404.12473*.

Charmet, F., Tanuwidjaja, H. C., Ayoubi, S., Gimenez, P.-F., Han, Y., Jmila, H., Blanc, G., Takahashi, T., & Zhang, Z. (2022). Explainable artificial intelligence for cybersecurity: A literature survey. *Annals of Telecommunications, 77*(11–12), 789–812.

Chitteti, S. R., Vamshi, S., & Kanaparthi, L. (2024). Malware executables detection using XGBoost. *International Journal for Research in Applied Science and Engineering Technology, 12*(4), 56–60.

Dhanya, L., Chitra, R., & Anusha Bamini, A. M. (2022). Performance evaluation of various ensemble classifiers for malware detection. *Materials Today: Proceedings, 62*, 4973–4979.

Ferdous, J., Islam, R., Mahboubi, A., & Islam, M. Z. (2025). A survey on ML techniques for multi-platform malware detection: Securing PC, mobile devices, IoT, and cloud environments. *Sensors, 25*(4), 1153.

Galli, A., La Gatta, V., Moscato, V., Postiglione, M., & Sperlí, G. (2024). Explainability in AI-based behavioral malware detection systems. *Computers & Security*, *141*, 103842.

INTERPOL. (2025). *INTERPOL Africa cyberthreat assessment report 2025*. INTERPOL.

Jurečková, O., Jureček, M., Stamp, M., Di Troia, F., & Lórencz, R. (2024). Classification and online clustering of zero-day malware. *Journal of Computer Virology and Hacking Techniques, 20*(4), 579–592.

Li, C., Lv, Q., Li, N., Wang, Y., Sun, D., & Qiao, Y. (2022). A novel deep framework for dynamic malware detection based on API sequence intrinsic features. *Computers & Security, 116*, 102686.

Louthánová, P., Mišák, M., Bílý, M., & Horák, K. (2024). A comparison of adversarial malware generators. *Journal of Computer Virology and Hacking Techniques*.

Maniriho, P., Mahmood, A. N., & Chowdhury, M. J. M. (2023). API-MalDetect: Automated malware detection framework for Windows based on API calls and deep learning techniques. *Journal of Network and Computer Applications, 218*, 103704.

Maniriho, P., Mahmood, A. N., & Chowdhury, M. J. M. (2024). MeMalDet: A memory analysis-based malware detection framework using deep autoencoders and stacked ensemble under temporal evaluations. *Computers & Security, 142*, 103864.

Mienye, I. D., & Sun, Y. (2022). A survey of ensemble learning: Concepts, algorithms, applications, and prospects. *IEEE Access, 10*, 99129–99149.

Onyedinma, E. G., Asogwa, D. C., & Onyenwe, I. E. (2025). Towards resilient malware detection: A hybrid framework leveraging static-dynamic features and ensemble models. *World Journal of Advanced Engineering Technology and Sciences, 15*(3), 634–639.

Rafique, N., Asghar, A., & Kanwal, A. (2025). Impact of machine learning algorithm choice and data quality on model accuracy. *Spectrum of Engineering Sciences, 3*(8), 793–803.

Rahima Manzil, H. H., & Naik, S. M. (2023). Detection approaches for Android malware: Taxonomy and review analysis. *Expert Systems with Applications, 232*, 122255.

Rodrigo, C., Pierre, S., Beaubrun, R., & El Khoury, F. (2021). BrainShield: A hybrid machine learning-based malware detection model for Android devices. *Electronics, 10*, 2948.

Roy, K. S., Ahmed, T., Udas, P. B., Karim, M. E., & Majumdar, S. (2023). MalHyStack: A hybrid stacked ensemble learning framework with feature engineering schemes for obfuscated malware analysis. *Intelligent Systems with Applications, 20*, 200283.

Saeed, S., Altamimi, S. A., Alkayyal, N. A., Alshehri, E., & Alabbad, D. A. (2023). Digital transformation and cybersecurity challenges for business resilience: Issues and recommendations. *Sensors, 23*(15), 6666.

Saha, S., Afroz, S., & Rahman, A. H. (2024). MAlign: Explainable static raw-byte based malware family classification using sequence alignment. *Computers & Security, 139*, 103714.

Singh, J., & Singh, J. (2021). A survey on machine learning-based malware detection in executable files. *Journal of Systems Architecture, 112*, 101861.

Singh, P., Borgohain, S. K., Sarkar, A. K., Kumar, J., & Sharma, L. D. (2023). Feed-forward deep neural network (FFDNN)-based deep features for static malware detection. *International Journal of Intelligent Systems, 2023*, Article 9544481.

Torres, M., Álvarez, R., & Cazorla, M. (2023). A malware detection approach based on feature engineering and behavior analysis. *IEEE Access*.

Zhang, Y., Yang, S., Xu, L., Li, X., & Zhao, D. (2023). A malware detection framework based on semantic information of behavioral features. *Applied Sciences, 13*(22), 12528.