B.Tech. BCSE497J - Project-I

MALWARE DETECTION USING EXPLAINABLE AI

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

21BCE2291 RAUNAK JHA

21BDS0079 RAHUL MADDALI

Under the Supervision of

Dr. Naveenkumar Jayakumar

Associate Professor

School of Computer Science and Engineering (SCOPE)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

November 2024

DECLARATION

I hereby declare that the project entitled MALWARE DETECTION USING EXPLAINABLE AI submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Prof. / Dr. Naveenkumar Jayakumar

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 20 November 2024

[illegible] [illegible]

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled MALWARE DETECTION USING EXPLAINABLE AI submitted by Raunak Jha(21BCE2291), Rahul Maddali (21BDS0079), **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by him / her under my supervision during Fall Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 20th November 2024

[illegible]

Signature of the Guide

[illegible] 22/11/24

Examiner(s)

Dr. Umadevi K. S, Dr. Murali. S
BTECH

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Ramesh Babu K, the Dean of the School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence. The Dean's dedication to academic excellence and innovation has been a constant source of motivation for me. I appreciate his efforts in creating an environment that nurtures creativity and critical thinking.

I express my profound appreciation to Dr. Murali. S, the Head of the Database Systems and Dr. Umadevi K. S. , the Head of the School of Computer Science (SCOPE) for his/her insightful guidance and continuous support. His/her expertise and advice have been crucial in shaping the direction of my project. The Head of Department's commitment to fostering a collaborative and supportive atmosphere has greatly enhanced my learning experience. His/her constructive feedback and encouragement have been invaluable in overcoming challenges and achieving my project goals.

I am immensely thankful to my project supervisor, Dr. Naveenkumar Jayakumar, for his dedicated mentorship and invaluable feedback. His/her patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor's willingness to share his expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. His support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

Name of the Candidate : Raunak Jha

ABSTRACT

In the digital age, sophisticated malware poses significant cybersecurity risks. This project develops a machine learning-based malware detection system using explainable AI methods to enhance both accuracy and interpretability. Static analysis features of executable files are processed with PCA for feature selection, and classification models, including Random Forest and XGBoost, are trained on these features. To address dataset imbalance, SMOTE is applied, ensuring equitable representation of benign and malicious samples.

A key innovation is the use of SHAP values to interpret model decisions, offering cybersecurity professionals insights into feature importance. While static analysis datasets limit the detection of dynamic malware behaviors, this research highlights the potential of combining advanced feature selection and explainable AI for robust malware detection. The outcome is a high-performing, interpretable model that strengthens trust in AI-driven security systems and paves the way for future advancements.

TABLE OF CONTENTS

Sl.No **Contents**

1. INTRODUCTION

1.1.Background

In recent years, the rise of sophisticated malware attacks has become a significant concern for organizations and individuals alike. Malware, such as viruses, ransomware, and trojans, is increasingly difficult to detect as attackers employ new strategies to evade traditional detection systems. Machine learning (ML) has emerged as a powerful tool in combating these threats due to its ability to recognize patterns in vast amounts of data. However, many current ML models face challenges when applied to malware detection, including the difficulty of handling high-dimensional datasets and the lack of model explainability. Traditional approaches either focus on shallow feature engineering or rely on black-box models, which make it hard for cybersecurity professionals to interpret how decisions are made.

The use of PCA for feature selection is a promising avenue for addressing these challenges. By mimicking natural evolution, these algorithms can efficiently select the most relevant features from large datasets, improving the performance of ML models. Additionally, SHAP (Shapley Additive Explanations) offers a way to provide transparency and interpretability, which is crucial in critical fields like cybersecurity. A combination of these methods offers the potential for more accurate, efficient, and interpretable malware detection.

1.2.Motivation

The motivations behind this project are rooted in the increasing complexity of malware and the need for transparent, interpretable models in cybersecurity:

1. Improving Detection Accuracy:

The use of a feature selection algorithm will help select the most relevant features from high-dimensional datasets, which can significantly improve the performance of malware detection models.

2. Enhancing Model Transparency:

Many machine learning models, though effective, operate as black-box systems. By integrating SHAP, this project seeks to bridge the gap between performance and explainability, enabling cybersecurity professionals to trust and understand the model's decisions.

3. Handling Data Complexity:

Malware datasets are often large and imbalanced. This project tackles this issue by using SMOTE for dataset balancing.

4. Real-World Relevance:

As cyber threats continue to evolve, the need for robust, explainable, and scalable detection systems becomes more critical. This project is aimed at addressing this real-world need by developing a practical solution that can be adapted for use in various cybersecurity environments. In summary, this project is motivated by the necessity for efficient, explainable, and scalable solutions in malware detection, with the potential to contribute to both academic research and real-world cybersecurity applications

1.3.Scope of the Project

This project focuses on designing a malware detection system that utilizes feature selection algorithms and machine learning models such as XGBoost or Random Forest. The system will:

1. Preprocess and balance the dataset using SMOTE to ensure a representative distribution of malware and non-malware samples.
2. Implement Principal ComponentAnalysis for feature selection, reducing the dimensionality of the data while retaining the most relevant features for malware detection.
3. Train the selected model (either XGBoost or Random Forest) to classify malware.
4. Use SHAP to explain the model's predictions, providing insights into feature importance and making the model interpretable to end-users.

The project will be developed and tested in Google Colab, using libraries like Pandas, NumPy, Scikit-learn, Matplotlib, DEAP, and SHAP. The final deliverable will include a fully functional

malware detection system, with performance metrics and detailed explanations of the model's decision-making process.

2. PROJECT DESCRIPTION AND GOALS

2.1. Literature Review

Table 2.1. Summary of Literature Review

Title	Methodology	Dataset Used	Demerits
Malware Detection using Machine Learning	• Machine Learning Algorith ms • Support Vector Machin es (SVM) • Feature Selecti on Process • Model Improve me nt Techniqu es • Dataset Analysis	The study also aims to conduct a thorough analysis of a dataset containing various malware samples. This analysis is crucial for training the machine learning models and ensuring that they can effectively learn from real-world examples of malware	• Dataset Limitatio ns • Model Complexity • Feature Selection Challeng es • Real-World Applicat io n • Scalability Issues

Malware Detection and Analysis based on AI Algorithm	• Review of Mainstrea m Models • Data Acquisition and Analysis • Simulation Tasks • Comparison of Feature Thresholds • Performance Metrics Calculation	The research focuses on how to utilize artificial intelligence algorithms for data acquisition in machine learning. This method involves collecting relevant data that can be used to train and test malware detection models, ensuring that the models are based on comprehensive	• Variability in Accuracy • Recall Rate Concerns • Dependence on Feature Selection • Need for Further Optimization • Limited Scope of Testing

5

8

Title	Methodology	Inputs Used	Demerits
Explainable artificial intelligence envisioned security mechanism for cyber threat hunting	• Design of XAISM-CTH • Security Analysis • Perform an ce Evaluati on • Practical Implementa ti on • Use of Explainable AI Techniques	The authors begin by designing the XAISM-CTH mechanism, which integrates explainable artificial intelligence into the cyber threat hunting process. This design includes creating network and threat models that facilitate effective threat detection and analysis.	• Scope of Threats Address ed • Dependen ce on Data Quality • Complexity o f Implementat io n • Performanc e in Dynamic Environmen ts • Generalizabili ty of Results

Methods for Detecting Malware Using Static, Dynamic and Hybrid Analysis	• Static Analysis • Dynamic Analysis • Hybrid Analysis • Reverse Engine eri ng • Sandboxing • Memory and Network Analysis	This method involves examining the code of malware without executing it. By executing the malware, analysts can monitor its activities, interactions with the system, and network communications. This method helps identify the malware's functions, how it spreads, and its overall impact on the system.	• Focus on Analysis Techniqu es • Static Analysis Inefficien cy • Conte xt-Specifi c Metho d Selecti on • Limited Real-World Examples • Evolving Threat Landsca pe •

Title	Methodology	Inputs Used	Demerits
A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques	• Traditional Machine Learning Techniques • Feature Extraction • Deep Learning Techniques • Challenges and Limitations: • Recent Trends	In summary, while the paper provides a thorough overview of current machine learning techniques for malware detection, it also points out significant limitations and areas for improvement that need to be addressed to enhance the effectiveness of these methods in combating malware threats.	• Challenges of Traditional Machine Learning • Deep Learning Limitations • Computational Resources • Interpretability of Models • Evolving Malware Techniques • Future Research Directions
A learning model to detect maliciousness of portable executable using integrated feature set	• Dataset Compilation • Feature Extraction • Application of Machine Learning Algorithms • Model Training and Evaluation • Comparative Analysis • Performance Metrics	The authors gathered a diverse dataset of portable executable (PE) files, which included both benign and malicious samples. This dataset is essential for training the model and ensuring it can generalize well to different types of PE files.	• Feature Set Expansion • Adaptability to New Malware Variants • Real-World Application Testing • Integration with Other Security Solutions • User Experience and Usability

Title	Methodology	Inputs Used	Demerits
Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning	• Genetic Algorithm for Feature Selection • Machi ne Learni ng Classifiers • Experime nt al Validatio n • Performan ce Metrics • Recent Trends	In summary, the introduction sets the stage for addressing the critical issue of Android malware detection, while the objectives focus on leveraging Genetic Algorithms for feature selection to enhance the performance of machine learning classifiers.	• Real-World Applicat io n • Feature Selectio n Limitatio ns • Computati on al Resources • Comparis on with Other Technique s
Using shapely values to define subgroups of forecasts for combining	• Accuracy of the Forecast • Average Accuracy of Other Forecasts • Average Degree of Diversity • Average Degree of Diversity Between Other Forecasts	The paper employs an algorithm based on the Shapley value method to select a superior subset of forecasts from a larger group. This method is rooted in cooperative game theory and is designed to evaluate the contribution of each forecast to the overall combination of forecasts.	• Featur e Set Expans io n • Adaptability to New Malware Variants • Real-World Applicati o n Testing • Integrati on with Other Security Solution s • User Experience and Usability

Title	Methodology	Inputs Used	Demerits
Variational Shapley Network: A Probabilistic Approach to Self-Explaining Shapley values with Uncertainty Quantification	• Robust Predictive Performan ce • Effective Uncerta in ty Quantification • Comparis on with Traditiona l Methods • Generalizatio n Across Datase ts • Enhanced Explanator y Power	The paper introduces a new method called the Variational Shapley Network, which aims to simplify the computation of Shapley values. This method is designed to require only a single forward pass through the model, significantly reducing the computational burden associated with traditional methods.	• Complexity o f Implementat io n • Dependence on Forecast Diversity • Limited Scope of Experiments • Potenti al Overfit ti ng • Evaluation Metrics

Explainability in AI-based Behavioral Malware Detection Systems	• Evaluation of XAI Methods • SHAP (SHapley Additive exPlanations) • LIME (Local Interpretab le Model-agnostic Explanatio ns) • LRP (Layer-wise Relevance Propagatio n) • Attention Mecha ni sm • Datasets Used 9	The paper specifically examines the applicability of recurrent deep learning architectures, namely Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) models. These architectures are evaluated for their effectiveness and efficiency in the context of behavioral malware detection.	• Limited Scope of XAI Methods • Dataset Diversity • Perform an ce Metrics • Real-World Applicat io n • User-Centric Evaluation

Title	Methodology	Inputs Used	Demerits
Enhancing android malware detection explainability through function call graph APIs	• Static Analysis • Function Call Graphs • Feature Selection and Extraction • Model Training • Evaluation of Results • Impact Analysis	The paper employs static analysis as a primary method for detecting Android malware. This technique involves examining the code of applications without executing them, allowing for the identification of potentially malicious behaviors based on the structure and content of the code.	• Scope of Static Analysis • Feature Selection Limitations • Generalization Across Different Malware Types • Explainability Depth • Performance Metrics • Real-World Application Testing

2.2.Research Gap

1) Dataset Limitations

Dataset limitations are crucial for improvement in malware detection using explainable AI (XAI) because they directly affect the model's ability to generalize, interpret, and provide meaningful insights.

The performance of the XAISM-CTH mechanism heavily relies on the quality and quantity of the data used for training the AI models. If the data is biased or insufficient, it could lead to inaccurate threat detection and explanations, potentially undermining the trust that the system aims to build

- Bias and Generalizability: Small or outdated datasets may not capture the diversity of real-world malware, leading to models biased toward specific types. XAI models rely on diverse data to generate explanations that apply across various malware types and platforms. Without diverse datasets, the model's interpretability may be skewed.

- Lack of Novel Malware: If the dataset doesn't include new or zero-day malware, the model's ability to explain novel threats will be compromised. AI models trained on limited datasets may struggle to provide clear explanations for behaviors not represented in the training data.

- Feature Completeness: Incomplete datasets might not capture all relevant malware behaviors or patterns, resulting in gaps in both detection accuracy and explainability. Without comprehensive data, XAI models may fail to explain or miss important malicious traits, reducing transparency.

Improving the dataset by incorporating diverse, real-world samples, including emerging malware, ensures that AI systems trained on these datasets are more robust and can provide meaningful, understandable explanations in varied contexts.

2) Feature Selection

Although the research emphasizes the importance of feature selection, it does not delve deeply into the specific methods used for selecting features. The effectiveness of the chosen features can vary significantly, and the lack of detailed discussion on this aspect may limit the reproducibility of the results in different contexts .

- High Dimensionality: Malware data often involves numerous features (e.g., system calls, API usage, file structures), making it difficult to select the most relevant ones without losing important information.

- Feature Redundancy and Noise: Many features might be redundant or irrelevant, introducing noise into the model. Selecting the right features is crucial to avoid misleading interpretations.

- Trade-off Between Complexity and Interpretability: Complex features might enhance detection accuracy but make the AI less explainable, complicating the balance between performance and clarity.

- Dynamic Behavior Features: Malware evolves, and static feature sets may not capture dynamic traits, hindering the model's ability to adapt and provide accurate explanations for emerging threats.

These challenges in feature selection persist due to several inherent factors:

- Malware Complexity and Evolution: Malware evolves rapidly, constantly introducing new features and behaviors, making it difficult to determine which features are consistently relevant across different types.

- Balance Between Accuracy and Explainability: Increasing detection accuracy often relies on more complex features, but this reduces model interpretability, a core aspect of explainable AI.

3) Accuracy and Recall

The results show a significant range in classification accuracy (73% to 94.3%) and detection accuracy (83.37% to 94.6%). This variability suggests that the model may not consistently perform well across different datasets or types of malware, indicating a limitation in its robustness and generalizability.

12

16

The recall rate ranges from 74.86% to 89.47%, which implies that there are instances where the model may fail to detect certain malware (false negatives). This limitation is critical, as it can lead to undetected threats, potentially compromising system security.

- Imbalance in Datasets: Malware datasets are often imbalanced, with many more benign samples than malicious ones, making it hard to achieve high recall without compromising accuracy. Models may focus on detecting common malware, missing rare or sophisticated threats.

- Evasion Techniques: Malware authors employ advanced techniques like obfuscation, polymorphism, and anti-analysis measures, which can lower the recall rate and accuracy by making detection models ineffective.

- Dynamic and Static Feature Trade-offs: Static features are fast to analyze but may miss complex malware behaviors, while dynamic features improve detection but at the cost of speed and computational overhead.

These issues persist due to the evolving nature of malware, difficulties in maintaining diverse and representative datasets, and the inherent trade-offs between model performance metrics.

4) Overfitting

Overfitting is a significant risk in malware detection using explainable AI because it occurs when the model learns overly specific patterns from the training data, including noise and irrelevant details, instead of generalizing to new, unseen data. This can lead to high accuracy on the training set but poor performance on real-world samples, especially when encountering novel malware.

Effects of Overfitting:

• Poor Generalization: The model may fail to detect new or slightly modified malware strains.

• Misleading Explanations: In explainable AI, overfitted models provide explanations tied to noise or irrelevant patterns, reducing trust in the system.

• High False Positives/Negatives: Overfitting can cause the model to incorrectly flag benign files or miss malicious ones, skewing recall and precision.

These challenges persist due to the complexity of malware patterns and the vast, evolving feature space in training data.

5) Typical Blackbox nature of ML algorithms

The black-box nature of machine learning (ML) prediction algorithms refers to the inability to clearly understand or interpret how a model arrives at its predictions. While models like neural networks or complex ensemble methods (e.g., random forests) may achieve high accuracy, their internal decision-making processes are often opaque. This lack of transparency makes it difficult to explain the model's behavior, identify biases, or ensure trustworthiness, especially in sensitive applications like malware detection, where understanding the reasoning behind predictions is crucial.

- Lack of Interpretability: Users cannot easily understand how or why certain predictions are made, leading to difficulty in trusting the model's decisions.

- Bias and Unintended Consequences: If a model makes biased decisions, it can be challenging to identify the source of bias or correct it without understanding the underlying decision-making process.

- Accountability Issues: In high-stakes areas like malware detection, it becomes harder to explain and justify actions based on opaque predictions, reducing transparency.

- Regulatory Concerns: Fields like healthcare or cybersecurity often require explainable outcomes, and black-box models may not comply with these regulatory demands.

6) Miscellaneous

- Complexity of Implementation:
  - The algorithm based on the Shapley value method may be complex to implement in practice, especially for large datasets with numerous forecasts. The computational intensity required to calculate Shapley values for each forecast can be a barrier for practitioners who may not have access to advanced computational resources or expertise in algorithm implementation

- Performance Overhead:
  - Dynamic analysis, while effective, incurs significant performance costs in terms of time and computational resources. Optimizing the dynamic component for real-time detection without compromising accuracy is an area for improvement.
  - the Genetic Algorithm itself can be resource-intensive, especially during the feature selection phase. This could limit its applicability in resource-constrained environments. Future research could focus on optimizing the Genetic Algorithm to reduce its computational demands further.

- Adaptability to New Malware Variants:
  - The study primarily focuses on existing malware samples, which raises questions about the model's adaptability to new and emerging malware variants. Research could be directed towards developing mechanisms that allow the model to continuously learn and adapt to new threats as they arise, ensuring its relevance in a rapidly evolving landscape.

- Malware continuously evolves, and models trained on existing datasets may struggle with new or zero-day malware. Continuous learning mechanisms or adaptive algorithms could address this limitation by allowing the model to evolve over time.

- **Scalability Issues:** The research does not discuss the scalability of the proposed models when applied to large-scale systems or networks. As organizations grow and their data increases, the ability of the models to maintain high accuracy and efficiency in real-time detection remains an important concern that is not addressed in the study .

- **Real-World Application:**
- The study's findings are based on controlled experiments, which may not fully reflect real-world scenarios where malware can exhibit more complex behaviors. The paper does not address how the models would perform in dynamic environments where malware evolves rapidly, which is a critical consideration for practical applications in cybersecurity
- The paper does not extensively discuss how well the Variational Shapley Network generalizes across different types of machine learning models. While it shows promise, further research is needed to determine its applicability to various architectures, such as ensemble methods or deep learning models, which may have different characteristics and complexities.

- **Interpretability of Probabilistic Outputs:** While the incorporation of uncertainty quantification is a significant advancement, the interpretability of the probabilistic outputs may pose challenges for users. Future work could focus on developing methods to effectively communicate these uncertainties to end-users, ensuring that the explanations remain accessible and actionable.

2.3.Objectives

- Develop an efficient and explainable malware detection system to address challenges posed by high-dimensional datasets and the black-box nature of traditional ML models.
- Leverage PCA for feature selection to identify the most critical features for distinguishing malware.
- Train XGBoost or Random Forest models using these selected features for accurate malware detection.
- Utilize SHAP to make model predictions transparent, providing insights into how each feature contributes to the detection decision.
- Create a scalable, interpretable system suitable for real-world cybersecurity applications.

2.4.Problem Statement

The rapid growth of malicious software (malware) poses significant threats to the security and stability of digital systems. Traditional malware detection techniques often struggle with high-dimensional datasets, making it difficult to identify the most relevant features that distinguish between benign and malicious software. Moreover, many machine learning models used for malware detection are considered "black boxes," offering limited transparency into how decisions are made, which is critical for cybersecurity professionals to understand and trust the model's predictions.

This project aims to develop an efficient and explainable malware detection system that addresses these challenges by leveraging feature selection methods to identify the most critical features from high-dimensional malware datasets. The selected features will be used to train either XGBoost and Random Forest models. Finally, SHAP (Shapley Additive Explanations) will be employed to make the model's predictions transparent, providing insight into how each feature contributes to the detection decision.

The goal is to create a robust, scalable, and interpretable system that improves detection accuracy while offering detailed explanations of the model's behavior, making it suitable for real-world cybersecurity applications.

2.5.Project Plan

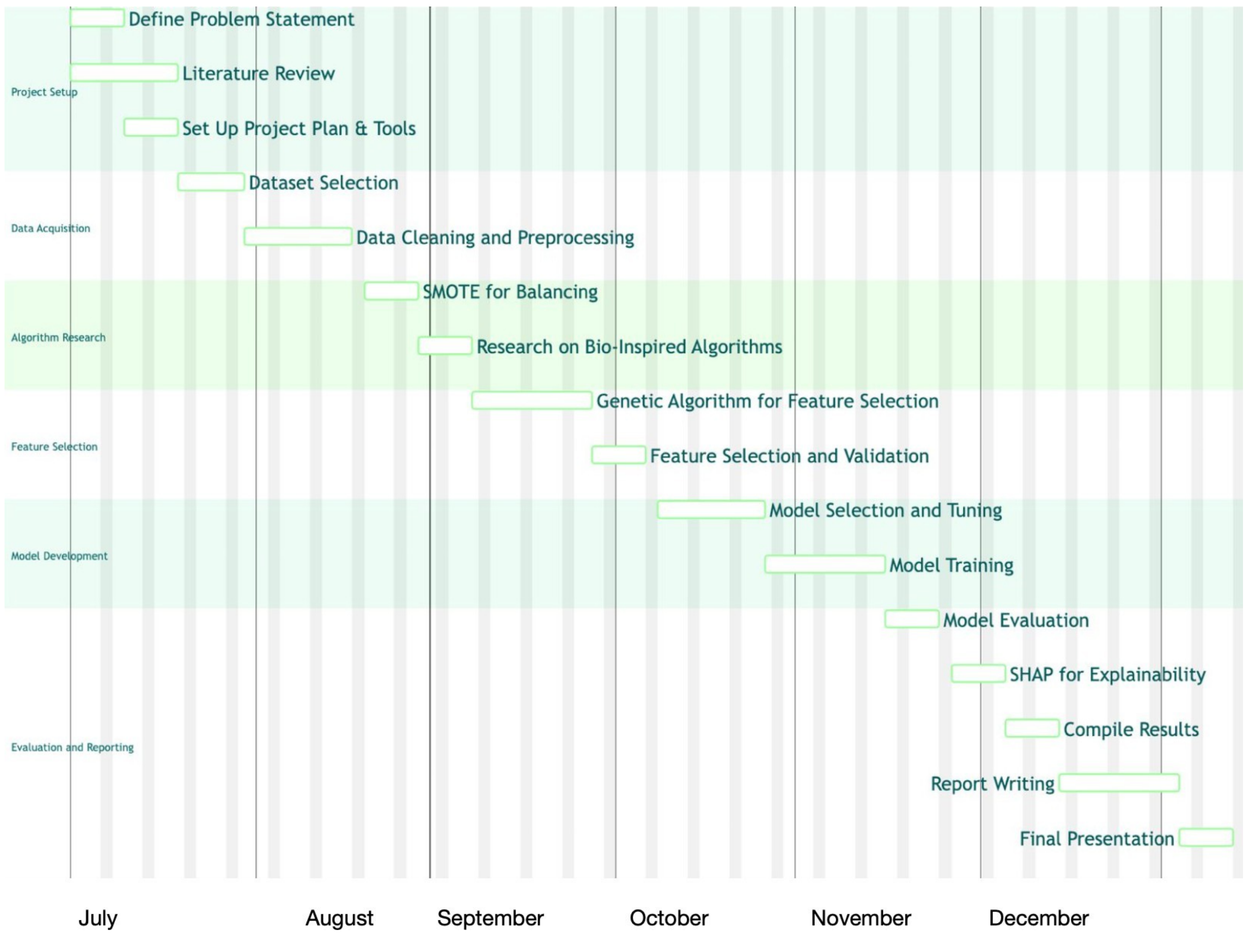


Fig. 1. Gantt chart

3. TECHNICAL SPECIFICATION

3.1.Requirements

We aim to solve this problem by selecting relevant features from the dataset, followed by training a machine learning model (XGBoost and Random Forest) to detect malware. To address the issue of model transparency, we will use SHAP (Shapley Additive Explanations) to explain the model's predictions, making it interpretable to cybersecurity professionals.

Methods to Solve the Problem

- Model Selection and Training: We will use either XGBoost or Random Forest to build the malware detection model. Both models are robust, scalable, and can handle large datasets with complex feature interactions.
- Explainability with SHAP: After model training, SHAP will be used to provide detailed explanations for each prediction made by the model. This will allow us to understand the contribution of each feature to the model's decision, enhancing the transparency of the detection system.

3.1.1.Functional Requirements

1. **Dataset Handling**:

- The system must be able to ingest and preprocess the Microsoft Big Malware Dataset.
- The system should implement SMOTE to balance the dataset.

2. **Feature Selection**:

- The system must implement Principal ComponentAnalysis to perform feature selection.
- The system should evaluate selected features to ensure they contribute positively to model performance.

3. **Model Training**:

- The system should allow training with XGBoost or Random Forest based on a selected configuration.
- The system must implement cross-validation to assess model performance during training.

4. **Performance Metrics**:

- The system should calculate key metrics such as accuracy, precision, recall, F1-score, and confusion matrix for evaluation.

5. **Explainability**:

- The system must implement SHAP to explain the model's predictions and provide Model Persistence:
- The system should save the trained model and SHAP explanations for future use or audits.

3.1.2.Non-functional Requirements

1. **Performance**:

- The feature selection and model training process should be optimized for speed and efficiency, capable of handling large malware datasets without significant delays.

2. **Scalability**:

- The system should be scalable, able to accommodate larger datasets or different types of malware data in the future without significant modifications.

3. **Reliability**:

- The system must be reliable, ensuring that the malware detection model performs consistently across different test datasets.

4. **Security**:

- Since the system deals with malware data, security is a critical concern. The system must have safeguards to prevent the contamination of data environments or system failure due to malicious inputs.

5. **Explainability and Transparency**:

- The SHAP implementation must be able to produce detailed explanations for every prediction made by the model. This requirement is crucial for transparency, especially in cybersecurity contexts where understanding the model's decisions is as important as the accuracy of predictions.

6. **Usability**:

- The system interface should be intuitive, allowing cybersecurity professionals to navigate through model metrics, SHAP visualizations, and results without requiring extensive machine learning expertise.

7. **Portability**:

- The solution should be portable across different operating systems and environments (e.g., local, cloud-based) to ensure it can be deployed in various organizational settings.

8. **Maintainability**:

- The system should be easy to maintain, allowing for regular updates (e.g., adding new malware signatures or updating models) without extensive system downtime.

4. Design and Implementation

4.1.System Architecture and Flow

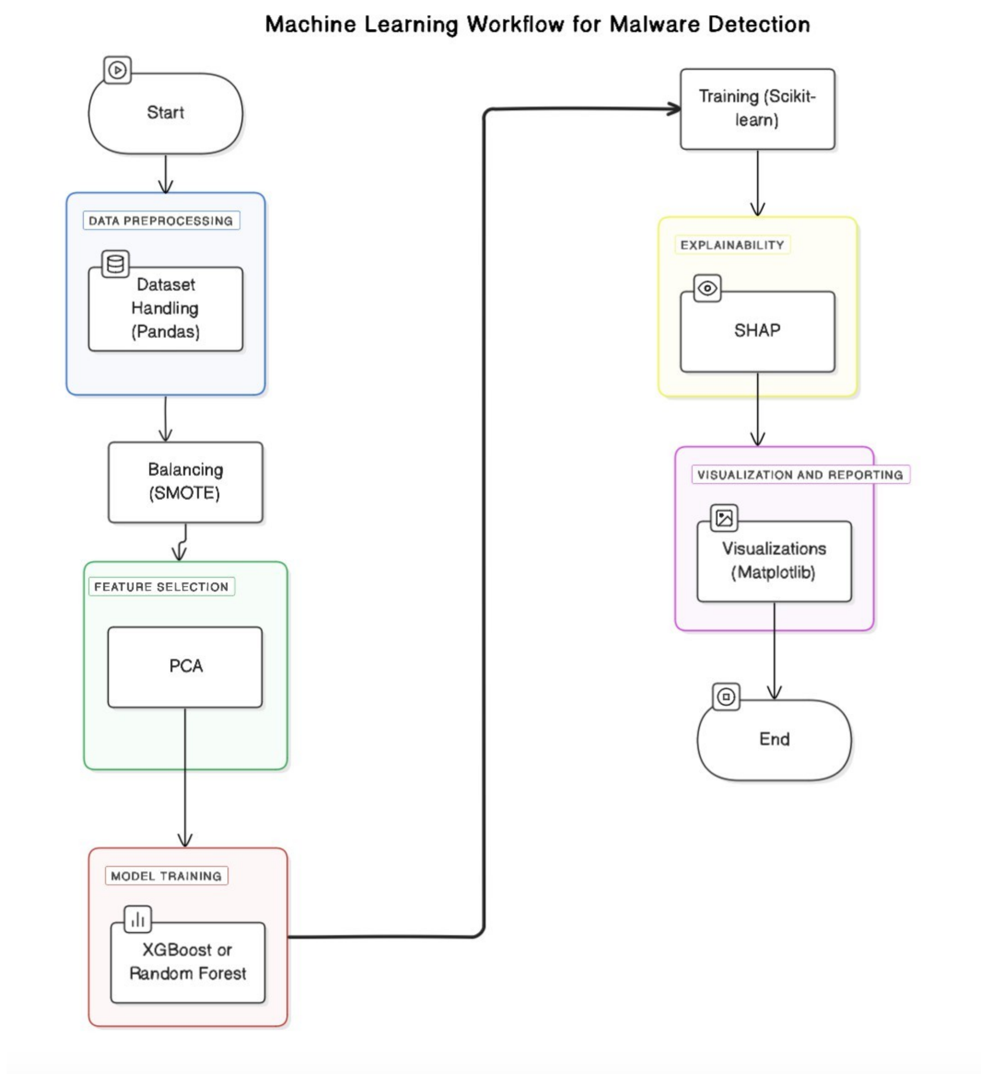
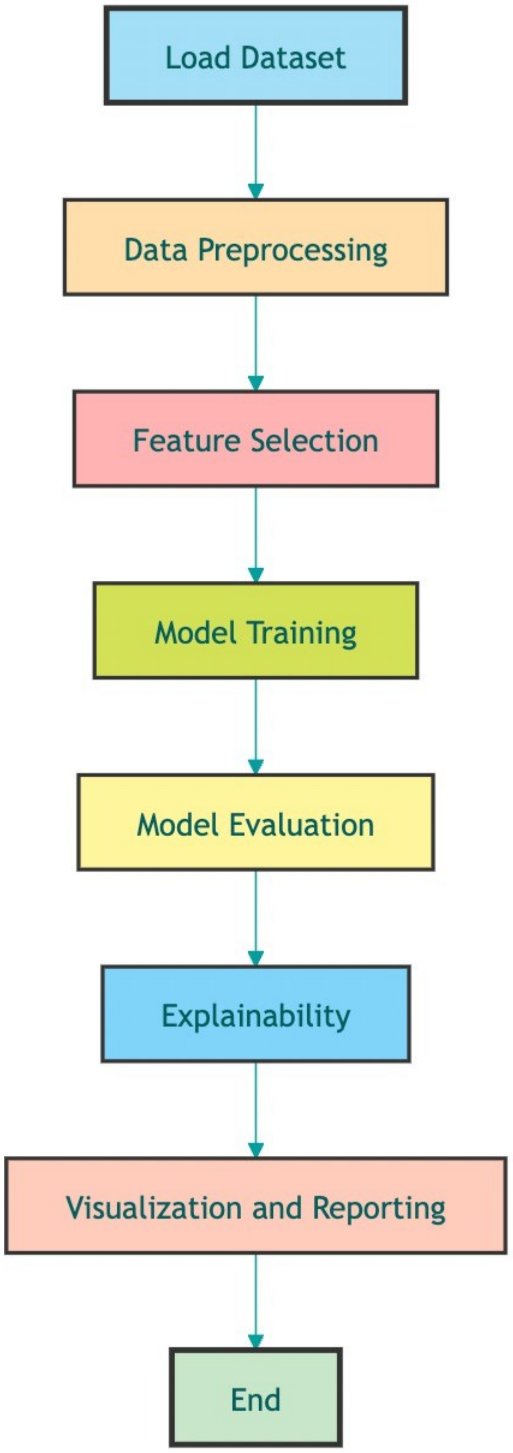
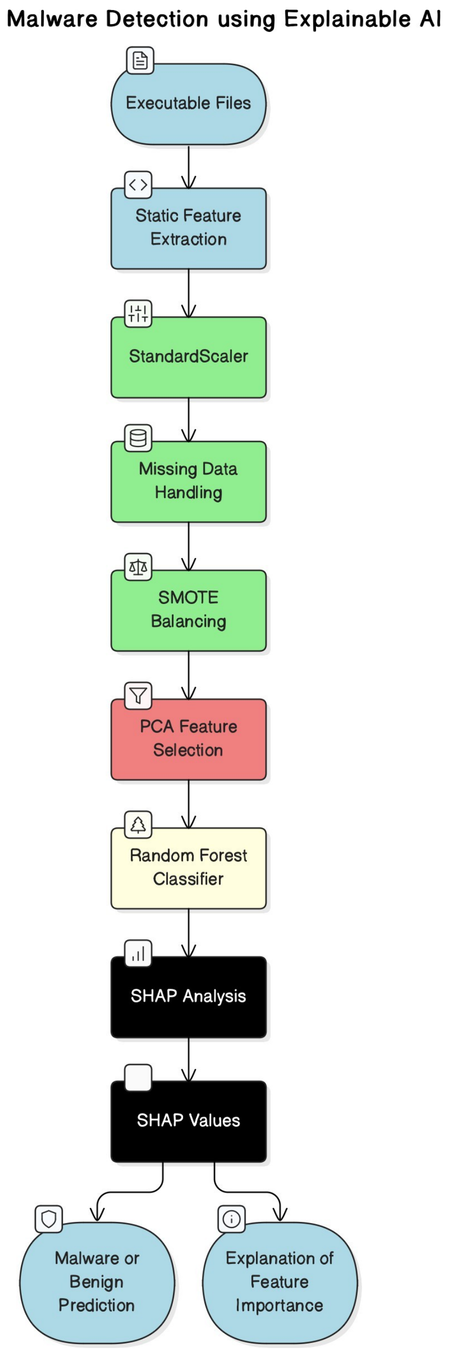


Fig 4.1.1

System flowchart

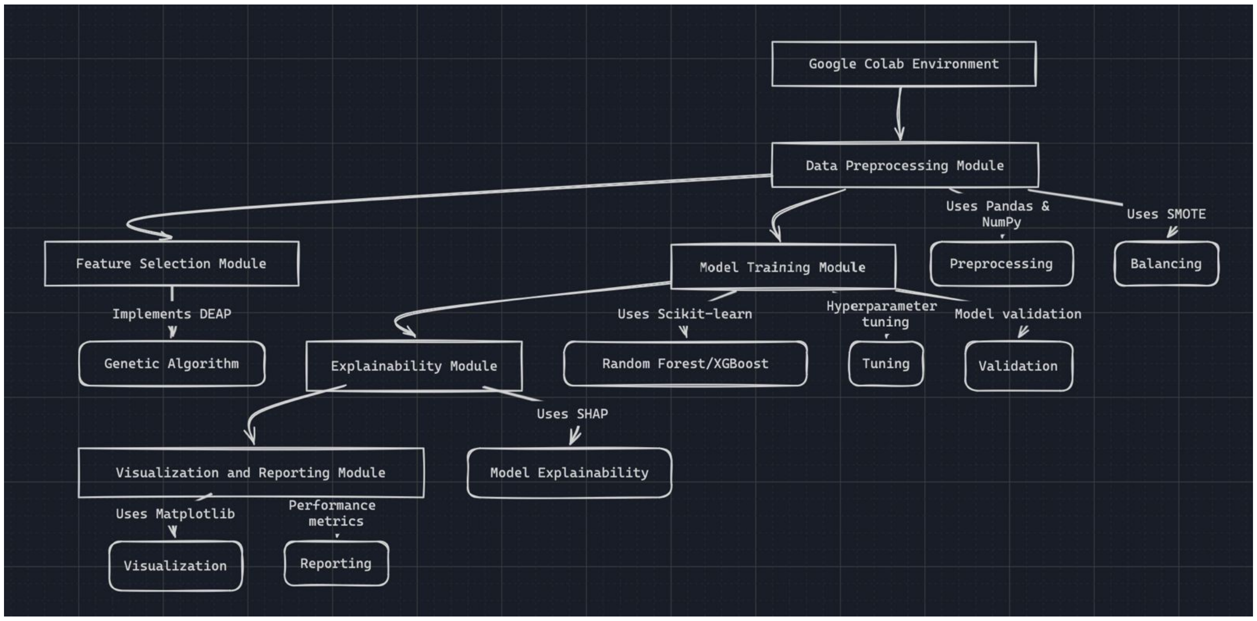

Load Dataset
Data Preprocessing
Feature Selection
Model Training
Model Evaluation
Explainability
Visualization and Reporting
End


Malware Detection using Explainable AI
Executable Files
Static Feature Extraction
StandardScaler
Missing Data Handling
SMOTE Balancing
PCA Feature Selection
Random Forest Classifier
SHAP Analysis
SHAP Values
Malware or Benign Prediction
Explanation of Feature Importance

4.2 workflow

Fig 4.2.1

System design



5. METHODOLOGYAND TESTING

This project aims to develop a robust and explainable malware detection framework that addresses these challenges. By leveraging Principal Component Analysis (PCA) for dimensionality reduction, the system identifies the most critical features from high-dimensional malware datasets, ensuring computational efficiency without sacrificing accuracy. Advanced machine learning algorithms, including Random Forest and XGBoost, are then employed for malware classification, benefiting from their strong predictive capabilities. To enhance interpretability, SHAP (Shapley Additive Explanations) is utilized, providing detailed insights into the contribution of each feature to the detection decisions.

The proposed methodology not only improves malware detection accuracy but also bridges the gap between predictive performance and interpretability. By integrating explainable artificial intelligence (XAI) techniques, this project offers a scalable and transparent solution for real-world cybersecurity applications, enabling professionals to trust and act upon model predictions confidently

.

5.1.Experiment Environment

The code provided is designed to run on a Google Colab environment, a cloud-based platform for executing Python code interactively. Google Colab provides access to Jupyter Notebook-like functionality, including GPU/TPU acceleration, making it ideal for data science, machine learning, and deep learning tasks.

1. Environment Setup

1. Google Colab Environment:

• Pre-configured with Python and essential libraries such as NumPy, Pandas, Scikit-learn, Matplotlib, and more.

• Allows installation of additional packages with pip commands directly in the notebook.

• Files can be uploaded directly, or Google Drive can be mounted to access larger datasets.

2. Execution Context:

• The user executes the code cell-by-cell.

• Outputs are displayed inline, including tables, graphs, and SHAP plots.

3. Hardware Options:

• Users can select runtime types such as GPU or TPU for faster computations (useful for large datasets or complex models like XGBoost).

2. Dataset Description

Number of Rows: 5184

Number of Columns: 56

Header Metadata Columns

1.e_magic

• Likely represents a signature or magic number that identifies the file type (e.g., the MZ header for DOS executables).

2.e_cblp

• Number of bytes in the last page of the executable

file. 3.e_cp

• Number of pages in the executable file.

4.e_crlc

• Number of relocation entries in the header.

5. e_cparhdr

• Number of paragraphs in the header.

6. e_minalloc

• Minimum extra paragraphs needed.

7. e_maxalloc

• Maximum extra paragraphs needed.

8. e_ss

• Initial (relative) SS value for the stack segment.

9. e_sp

• Initial SP (stack pointer) value.

10. e_csum

• Checksum value for the executable.

11. e_ip

• Initial instruction pointer (entry point for code execution).

12. e_cs

• Initial CS (code segment) value.

13. e_lfarlc

4.e_crlc

• File address of the relocation table.

14. e_ovno

• Overlay number (used for modular programs).

15. e_res, e_res2

• Reserved bytes for future use (often unused).

16. e_oemid, e_oeminfo

• OEM (Original Equipment Manufacturer) identifiers.

PE Header Columns

17.SizeOfCode

• Size of the executable code in the file.

18. SizeOfInitializedData

• Size of the initialized data in the file.

19. SizeOfUninitializedData

• Size of the uninitialized data in the file.

20. AddressOfEntryPoint

• Address where execution starts when the file is loaded into memory.

21. BaseOfCode

• Starting address of the code section.

22. BaseOfData

• Starting address of the data section (if applicable).

4.e_crlc

23. ImageBase 28

4.e_crlc 26

• Preferred base address in memory for loading the executable.

24. SectionAlignment

• Alignment of sections in memory.

25. FileAlignment

• Alignment of raw data in the file.

26. MajorSubsystemVersion, MinorSubsystemVersion

• Version numbers of the subsystem required to run the executable.

27. Subsystem

• Type of subsystem used by the executable (e.g., console application, GUI).

28. CheckSum

• Checksum of the file for validation purposes.

29. SizeOfStackReserve

• Amount of memory reserved for the stack.

30. SizeOfStackCommit

• Amount of memory initially committed for the stack.

31. SizeOfHeapReserve

• Amount of memory reserved for the heap.

32. SizeOfHeapCommit

4.e_crlc

• Amount of memory initially committed for the heap.

Additional Columns

33.LoaderFlags

• Reserved for loader flags (often unused).

34. NumberOfRvaAndSizes

• Number of entries in the data directory of the PE header.

35. DLLCharacteristics

• Flags indicating specific DLL characteristics, such as ASLR or DEP compatibility.

36. Class

• Target variable for classification (likely binary, e.g., malicious vs. benign files).

3. Parameters and Key Components

1. Data Preprocessing:

• Missing Values: Managed using SimpleImputer to fill missing values with the mean.

• Feature Scaling: StandardScaler standardizes the features to zero mean and unit variance.

• PCA (Principal Component Analysis): Reduces dimensionality to retain 95% of data variance.

2. Handling Imbalance:

• SMOTE (Synthetic Minority Over-sampling Technique) balances the dataset by generating synthetic samples for underrepresented classes.

3. Model Parameters:

• Random Forest: A robust ensemble learning method with default parameters and random state for reproducibility. [illegible]

- XGBoost: Tuned with parameters like use_label_encoder=False and eval_metric='logloss'.

4. Evaluation Metrics:

• Accuracy, Recall, Precision, F1 Score, and ROC AUC provide comprehensive model performance analysis.

5. Explainability with SHAP:

• SHAP Values: Interpret feature importance and impact.

• Visualizations: Includes summary, dependence, and force plots to analyze feature contributions.

4. Execution Workflow

1. Load Dataset

The process begins by loading a high-dimensional malware dataset (ClaMP_Raw-5184.csv in the provided code). This dataset includes features that distinguish between benign and malicious software.

• The dataset is loaded using the pandas library.

• Exploratory steps (head, info, describe) allow initial understanding of data types, sample entries, and statistical properties.

• Missing values are identified using isnull().sum().

2. Data Preprocessing

To clean and prepare the data for feature engineering by addressing missing values, dropping irrelevant columns, and standardizing feature scales.

- Columns with all missing values are removed.
- Missing values are imputed using the mean strategy.
- Ensures consistency and completeness of data for modeling.
- Standardization is performed to scale features uniformly, aiding dimensionality reduction.

3. Feature Selection

Reduce the number of features to remove irrelevant or redundant ones, simplifying model training and improving performance.

- Features with zero variance are dropped as they contribute no value to distinguishing malware.
- Principal Component Analysis (PCA) is applied to reduce dimensionality while preserving 95% of the data variance.

4. Model Training

Train machine learning models to classify data points as benign or malicious.

Two classifiers are used:

- Random Forest: A robust, ensemble-based algorithm that handles high-dimensional data efficiently.
- XGBoost: Gradient-boosting algorithm known for its performance and efficiency in handling complex datasets.

5. Model Evaluation

Assess the performance of trained models using appropriate metrics.

- Metrics like accuracy, precision, recall, F1-score, and ROC AUC are calculated for both models.
- ROC curves are plotted to visualize the trade-offs between true positives and false positives.

6. Explainability

Use SHAP (Shapley Additive Explanations) to make the model's predictions interpretable, providing insights into feature importance and decision-making.

- SHAP values are computed to determine the contribution of each feature to the model's predictions.
- Summary plots and dependence plots are generated to visualize feature importance and interactions.

7. Visualization and Reporting

Present the results and insights in an understandable and actionable format.

• Summary Plot:

Highlights the overall importance of features in the model.

• Dependence Plot:

Visualizes the relationship between a single feature and the model output.

• Force Plot:

Shows how features contribute to an individual prediction.

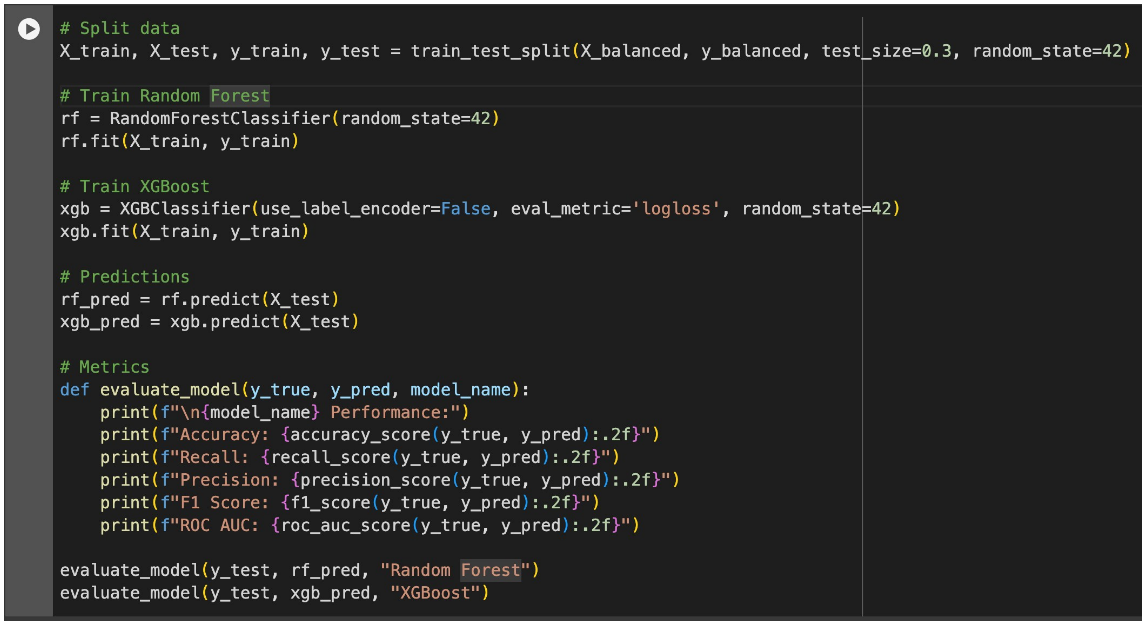
• Decision Plot:

Visualizes the decision-making process of the model.

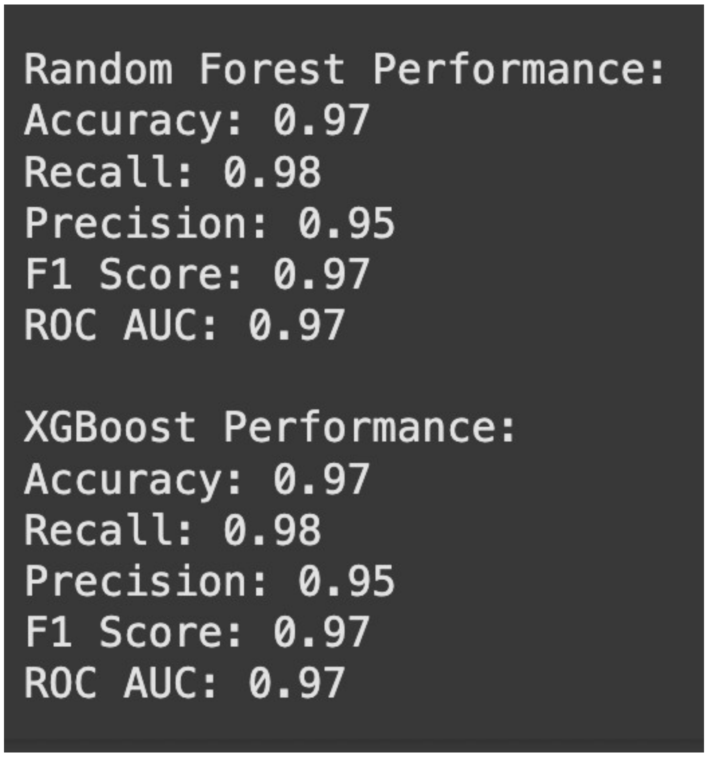
8. End

Finalize the workflow by summarizing results and making actionable recommendations based on insights derived from the explainability and evaluation stages.

5. Testing



```python
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.3, random_state=42)

# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# Train XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)

# Predictions
rf_pred = rf.predict(X_test)
xgb_pred = xgb.predict(X_test)

# Metrics
def evaluate_model(y_true, y_pred, model_name):
    print(f"\n{model_name} Performance:")
    print(f"Accuracy: {accuracy_score(y_true, y_pred):.2f}")
    print(f"Recall: {recall_score(y_true, y_pred):.2f}")
    print(f"Precision: {precision_score(y_true, y_pred):.2f}")
    print(f"F1 Score: {f1_score(y_true, y_pred):.2f}")
    print(f"ROC AUC: {roc_auc_score(y_true, y_pred):.2f}")

evaluate_model(y_test, rf_pred, "Random Forest")
evaluate_model(y_test, xgb_pred, "XGBoost")
```



```
Random Forest Performance:
Accuracy: 0.97
Recall: 0.98
Precision: 0.95
F1 Score: 0.97
ROC AUC: 0.97

XGBoost Performance:
Accuracy: 0.97
Recall: 0.98
Precision: 0.95
F1 Score: 0.97
ROC AUC: 0.97
```

1. Data Splitting:

- The dataset is split into training (70%) and testing (30%) subsets using train_test_split from Scikit-learn.
- random_state=42 ensures reproducibility of results by maintaining consistent splits across different runs.

2. Training Random Forest:

- ARandom Forest Classifier is initialized and trained on the X_train and y_train datasets.
- Random Forest is an ensemble learning method that combines the predictions of multiple decision trees to improve classification performance.

3. Training XGBoost:

- An XGBoost Classifier is trained on the same training dataset.

- use_label_encoder=False prevents deprecation warnings, and eval_metric='logloss' sets the evaluation metric for training to logarithmic loss, which is suitable for binary classification tasks.

4. Predictions:

Both classifiers make predictions (rf_pred and xgb_pred) on the test dataset, X_test.

5. Model Evaluation:

- Accuracy: The proportion of correct predictions.
- Recall (Sensitivity): The ability of the model to identify all positive instances.
- Precision: The proportion of true positives out of all predicted positives.
- F1 Score: The harmonic mean of Precision and Recall, balancing both metrics.
- ROC AUC: The area under the Receiver Operating Characteristic curve, measuring the trade- off between sensitivity and specificity.

Results Interpretation:

Both Random Forest and XGBoost achieve identical metrics:

- Accuracy: 97%
- Recall: 98%
- Precision: 95%
- F1 Score: 97%
- ROC AUC: 97%

This indicates that both models are highly effective in detecting malware, with minimal errors in classification.

SHAP: SHapley Additive exPlanations

SHAP (SHapley Additive exPlanations) is a game-theoretic approach to explain the output of machine learning models. Developed by Lundberg and Lee (2017), SHAP provides a unified framework to attribute predictions to individual features in a way that is both consistent and theoretically grounded. It is widely used in modern data science and AI projects to ensure transparency and interpretability of complex models, especially in critical applications like healthcare, finance, and security.

In this project, SHAP is employed to interpret the results of a malware detection model, enabling better understanding of how specific features contribute to the model's decisions. This is crucial for building trust in AI systems, identifying biases, and ensuring regulatory compliance.

SHAP is based on concepts from cooperative game theory, specifically the Shapley value introduced by Lloyd Shapley in 1953. The Shapley value provides a fair distribution of rewards (or contributions) among players in a cooperative game. In the context of machine learning:

The game is the prediction task.
The players are the input features.
The reward is the model's output (prediction).
SHAP values quantify each feature's contribution to the prediction by considering all possible coalitions (subsets of features) in which the feature could participate. This ensures that the contribution of a feature is computed in a way that is:

Consistent: If a feature’s contribution increases, its SHAP value will not decrease.
Additive: The sum of all SHAP values equals the difference between the model’s prediction and the baseline (expected value).

Marginal Contribution Calculation: SHAP calculates how much a feature changes the model's output when it is added to all possible feature subsets.
Feature Interaction: SHAP can capture interactions between features, enabling an analysis of not only individual contributions but also the combined effects of features.
Visualization: SHAP provides intuitive plots (e.g., summary plot, dependence plot, force plot) that help interpret model decisions at both global and local levels.

Why SHAP is Suitable for Malware Analysis Using the CLAMP Dataset

Feature Attribution
The CLAMP dataset (Comprehensive PE Malware Analysis and Prediction) provides detailed features extracted from Portable Executable (PE) files, specifically tailored for malware analysis. These features include critical metadata, structural details, and behavioral indicators that help differentiate malicious files from benign ones. SHAP's explainability framework is particularly effective in this context, as it enables a clear understanding of how the model utilizes specific PE file attributes to make predictions.

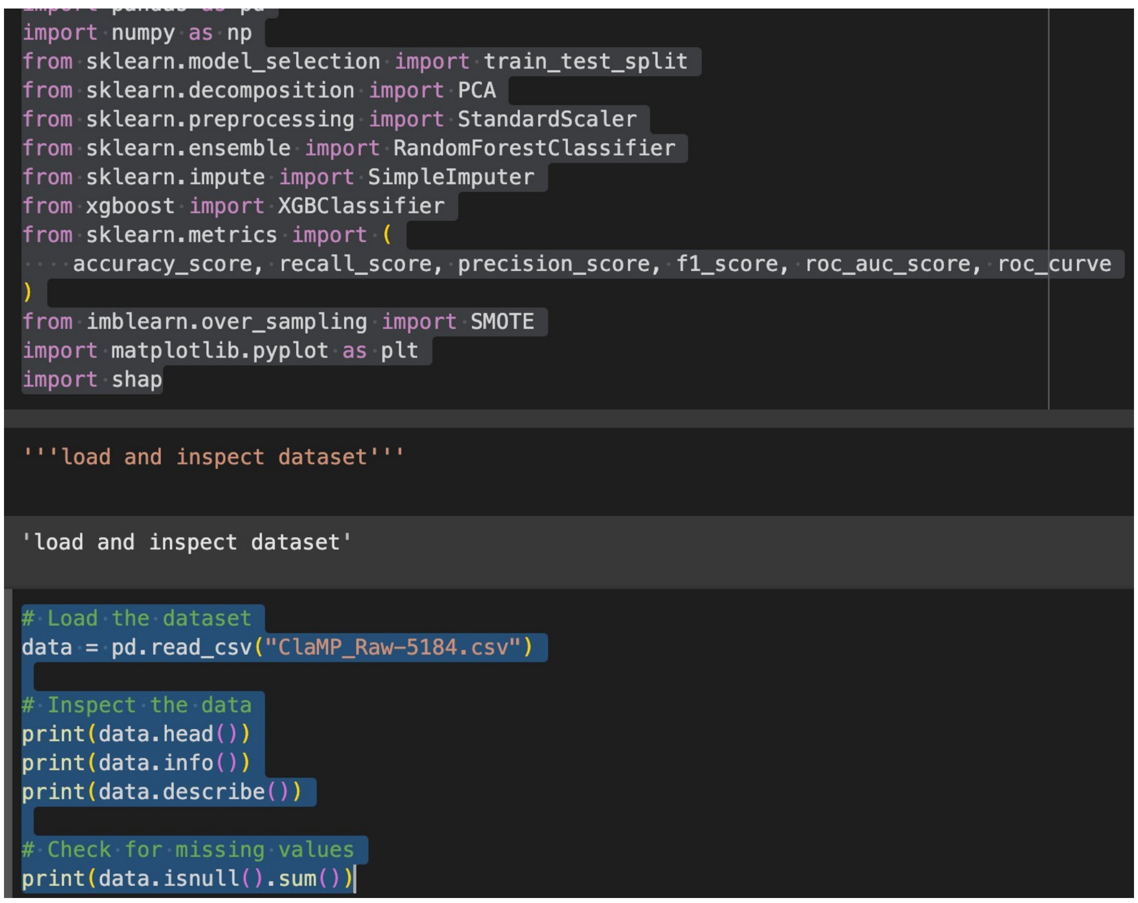
Highlighting Malicious Indicators:
SHAP values can identify features like irregular section sizes, high entropy, or specific imports (e.g., CreateRemoteThread) as significant contributors to malware classification.
For example, unusually small .text sections or high entropy in .rsrc sections might have high SHAP values for malicious labels.

Explaining Benign File Predictions:
Benign files often have well-formed headers, valid section alignments, and expected imports. SHAP can attribute negative SHAP values to these characteristics, showing their role in reducing the likelihood of a file being classified as malicious.
Ranking Feature Importance:

The summary plot (e.g., bar chart) can rank CLAMP-specific features by their average SHAP values, helping to identify the most influential attributes globally.
Analysts can focus on features with high SHAP values, such as entropy levels, section counts, or specific header fields, to understand model behavior better.

Uncovering Feature Interactions:
SHAP can highlight interactions between features, such as how high entropy in .text interacts with specific imports to influence predictions.
This insight is valuable for understanding how combinations of attributes contribute to classification decisions.

6. PROJECT DEMONSTRATION

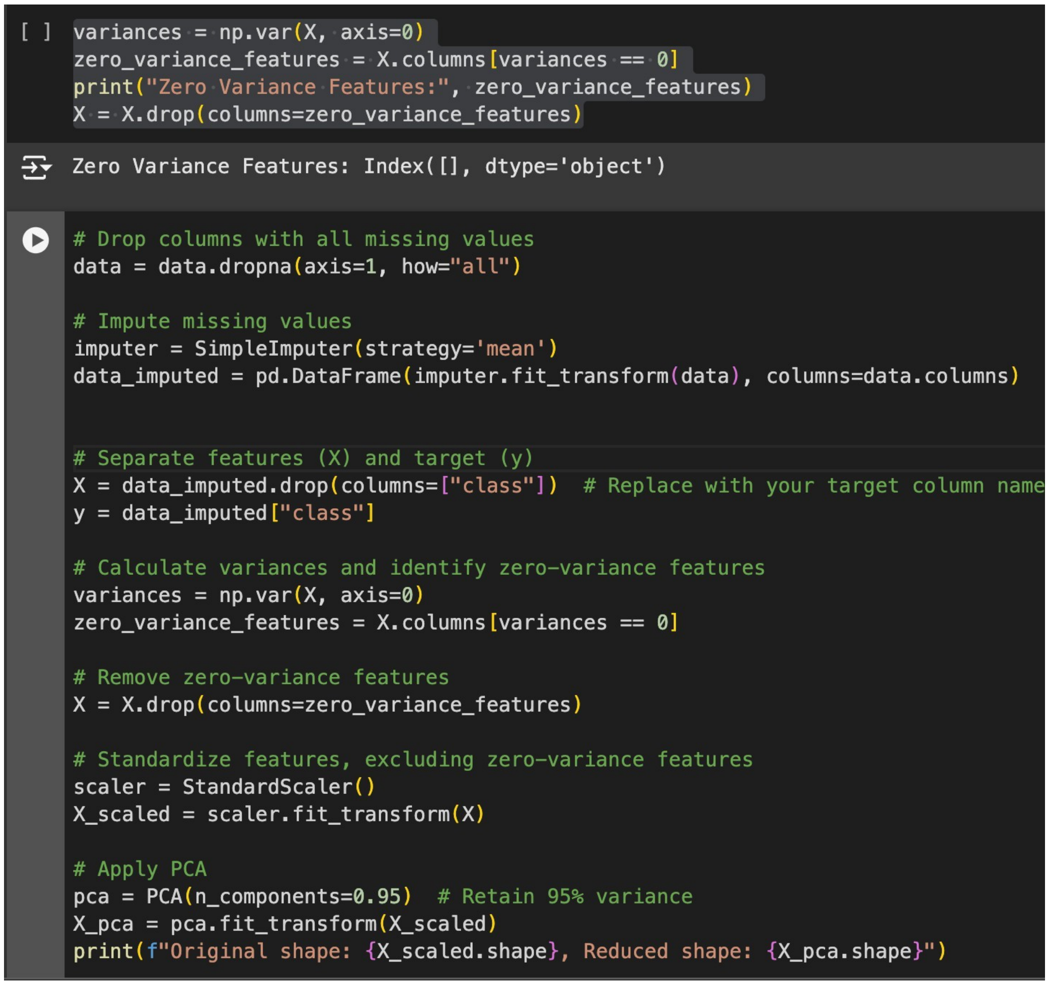


```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from xgboost import XGBClassifier
from sklearn.metrics import (
    accuracy_score, recall_score, precision_score, f1_score, roc_auc_score, roc_curve
)
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import shap
```

```
'''load and inspect dataset'''
```

```
'load and inspect dataset'
```

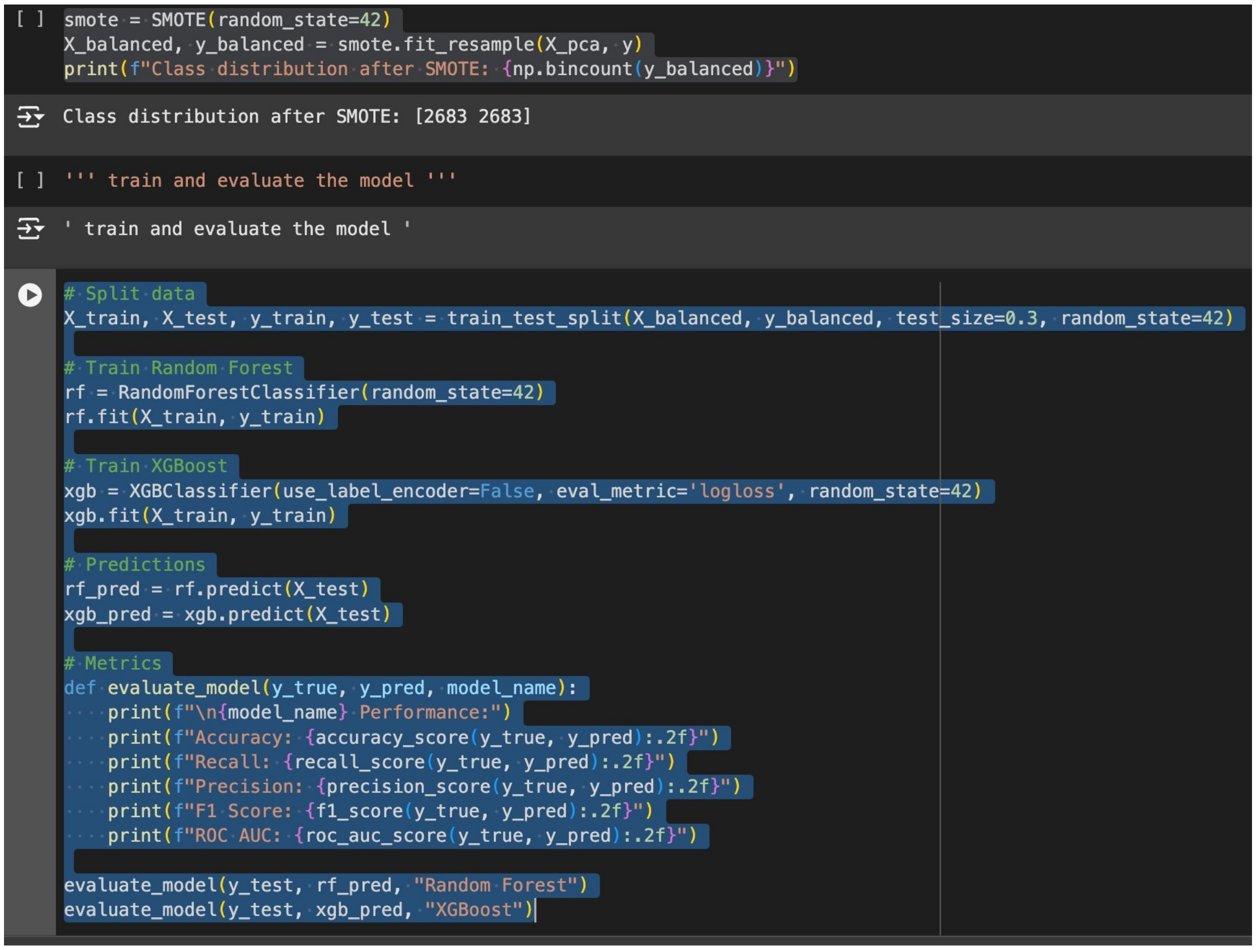
```
# Load the dataset
data = pd.read_csv("ClaMP_Raw-5184.csv")

# Inspect the data
print(data.head())
print(data.info())
print(data.describe())

# Check for missing values
print(data.isnull().sum())
```



```
[ ] variances = np.var(X, axis=0)
    zero_variance_features = X.columns[variances == 0]
    print("Zero Variance Features:", zero_variance_features)
    X = X.drop(columns=zero_variance_features)
```

```
Zero Variance Features: Index([], dtype='object')
```

```
# Drop columns with all missing values
data = data.dropna(axis=1, how="all")

# Impute missing values
imputer = SimpleImputer(strategy='mean')
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)


# Separate features (X) and target (y)
X = data_imputed.drop(columns=["class"])  # Replace with your target column name
y = data_imputed["class"]

# Calculate variances and identify zero-variance features
variances = np.var(X, axis=0)
zero_variance_features = X.columns[variances == 0]

# Remove zero-variance features
X = X.drop(columns=zero_variance_features)

# Standardize features, excluding zero-variance features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% variance
X_pca = pca.fit_transform(X_scaled)
print(f"Original shape: {X_scaled.shape}, Reduced shape: {X_pca.shape}")
```



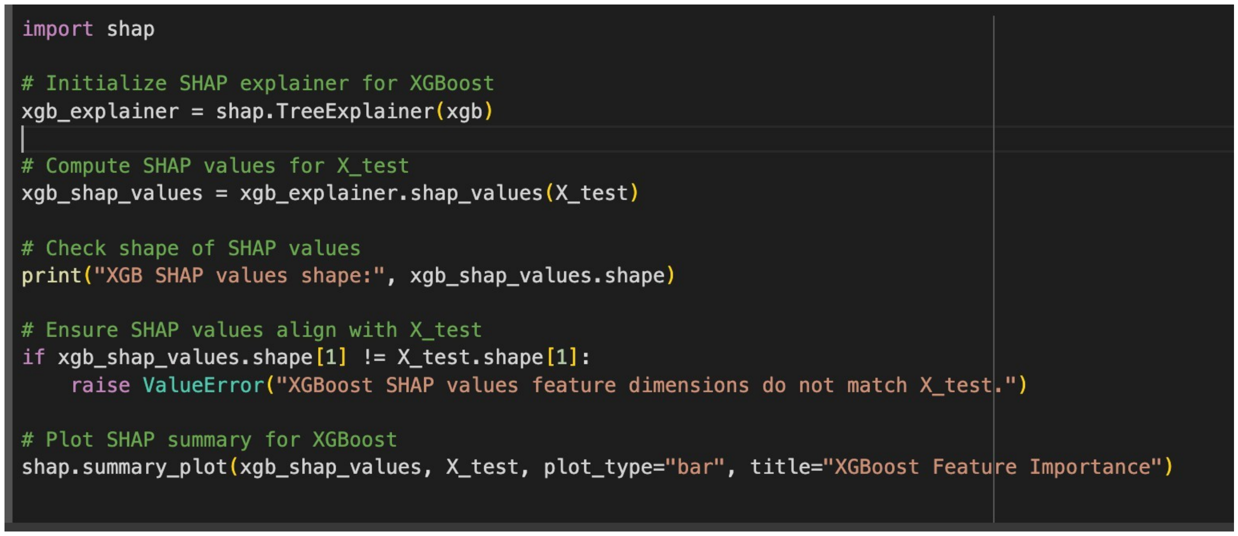
```
[ ] smote = SMOTE(random_state=42)
    X_balanced, y_balanced = smote.fit_resample(X_pca, y)
    print(f"Class distribution after SMOTE: {np.bincount(y_balanced)}")
```

```
Class distribution after SMOTE: [2683 2683]
```

```
[ ] ''' train and evaluate the model '''
```

```
' train and evaluate the model '
```

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.3, random_state=42)

# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# Train XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)

# Predictions
rf_pred = rf.predict(X_test)
xgb_pred = xgb.predict(X_test)

# Metrics
def evaluate_model(y_true, y_pred, model_name):
    print(f"\n{model_name} Performance:")
    print(f"Accuracy: {accuracy_score(y_true, y_pred):.2f}")
    print(f"Recall: {recall_score(y_true, y_pred):.2f}")
    print(f"Precision: {precision_score(y_true, y_pred):.2f}")
    print(f"F1 Score: {f1_score(y_true, y_pred):.2f}")
    print(f"ROC AUC: {roc_auc_score(y_true, y_pred):.2f}")

evaluate_model(y_test, rf_pred, "Random Forest")
evaluate_model(y_test, xgb_pred, "XGBoost")
```



```
import shap

# Initialize SHAP explainer for XGBoost
xgb_explainer = shap.TreeExplainer(xgb)

# Compute SHAP values for X_test
xgb_shap_values = xgb_explainer.shap_values(X_test)

# Check shape of SHAP values
print("XGB SHAP values shape:", xgb_shap_values.shape)

# Ensure SHAP values align with X_test
if xgb_shap_values.shape[1] != X_test.shape[1]:
    raise ValueError("XGBoost SHAP values feature dimensions do not match X_test.")

# Plot SHAP summary for XGBoost
shap.summary_plot(xgb_shap_values, X_test, plot_type="bar", title="XGBoost Feature Importance")
```

7. RESULTAND DISCUSSION

Results:

7.1.Outcome

•High Recall (98%): The models are excellent at identifying malware, minimizing false negatives (i.e., undetected threats).

•High Precision (95%): Among instances flagged as malware, a high proportion are true positives, minimizing false alarms.

•Balanced F1 Score (97%): The F1 score confirms a good trade-off between Precision and Recall, making the models reliable for deployment in cybersecurity scenarios.

•High ROC AUC (97%): Both models show strong discriminative power, effectively separating malware and benign samples.

- Random Forest vs. XGBoost:

•Both models perform identically in this scenario, but they have different computational and architectural characteristics:

•Random Forest: Easier to train, robust to overfitting, and interpretable due to feature importance measures.

•XGBoost: Computationally efficient, often outperforms in highly complex datasets, and provides fine-tuned control over hyperparamete^{4}rs^2.

Similar Performance of RandomForest and XGBoost:

1. Feature Engineering and Preprocessing

• PCA Transformation: The dataset was transformed using PCA to reduce dimensionality while retaining 95% of the variance. This transformation could make the dataset simpler, with less noise or collinearity, causing both RF and XGBoost to learn similar patterns.

• Standardized Features: After scaling, all features have a similar distribution, which can lead to both models identifying the same influential features.

• SMOTE Balancing: The balanced dataset created by SMOTE might emphasize the same patterns for both classifiers.

2. Hyperparameter Settings

• Default Hyperparameters: If both models use their default hyperparameters, they might behave similarly. For example:

• Random Forest uses a collection of decision trees, each trained on a random subset of features and data.

• XGBoost (a gradient boosting algorithm) can behave like a Random Forest if the learning rate is low, the tree depth is limited, and the number of estimators is comparable.

- Scalability and Practical Use:

• These results suggest that either model could be used for malware detection in real-world applications.

• Given the identical metrics, the choice between Random Forest and XGBoost could depend on factors like computational resources and ease of integration.

This report analyzes the results of SHAP (SHapley Additive exPlanations) plots generated from an XGBoost model trained on a dataset likely related to malware classification (ClaMP dataset). The SHAP values provide insights into feature importance and their impact on model predictions.

7.2.Description

The above code is the initial stage of a machine learning pipeline for malware detection. It begins by importing essential libraries for data manipulation (pandas, numpy), machine learning tasks (Scikit-learn for preprocessing, modeling, and evaluation; XGBoost for gradient boosting), oversampling techniques (SMOTE for handling class imbalance), visualization (matplotlib), and interpretability (shap). The dataset, ClaMP_Raw-5184.csv, is loaded into a DataFrame using pandas.read_csv(). Basic exploratory data analysis (EDA) is performed, including viewing the first few rows of the dataset (head()), obtaining structural and data type information (info()), generating summary statistics (describe()), and checking for missing values across all features (isnull().sum()). This foundational analysis provides an overview of the data, identifies potential quality issues like missing values, and prepares the dataset for subsequent preprocessing steps such as imputation and scaling.

The provided code performs a series of data preprocessing steps, ensuring the dataset is cleaned and optimized for machine learning modeling:

1Variance Calculation and Zero-Variance Features Removal:

• Variance is calculated for each feature in the dataset (np.var(X, axis=0)), and features with zero variance are identified. These are features where all values are constant, providing no useful information for predictive modeling.

• Such zero-variance features are dropped using X.drop(columns=zero_variance_features) to reduce noise and dimensionality.

2. Handling Missing Values:

• Columns with all values missing (NaN) are removed (data.dropna(axis=1, how="all")).

• Remaining missing values are imputed using the mean of each column (SimpleImputer(strategy='mean')). The imputed dataset is reconstructed as a DataFrame with original column names.

3. Feature and Target Separation:

• Features (X) and the target variable (y) are separated. The column named “class” is assumed to be the target, and all other columns are treated as features.

4. Standardization:

• Features are standardized using StandardScaler() to ensure that they have a mean of 0 and a standard deviation of 1. Standardization is critical for machine learning algorithms, especially those sensitive to feature scaling, such as PCA.

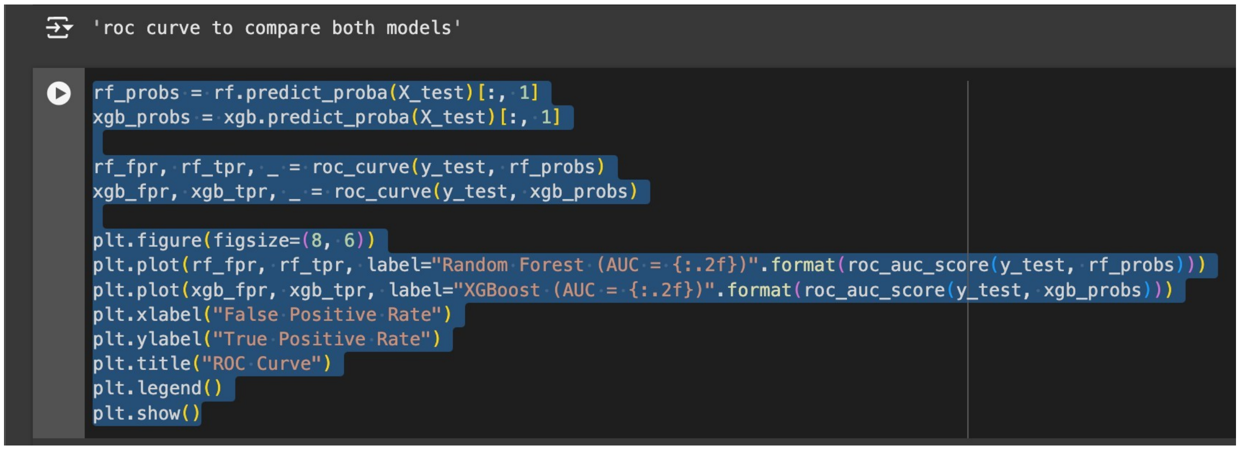
5. Principal Component Analysis (PCA):

• PCA is applied to reduce the dimensionality of the dataset while retaining 95% of the variance (PCA(n_components=0.95)). This helps in simplifying the model, reducing computational cost, and avoiding the curse of dimensionality.

• The transformed dataset (X_pca) is printed, showing the reduced shape compared to the original standardized dataset (X_scaled).

This preprocessing pipeline ensures the data is clean, optimized, and reduced to its most informative features. Removing zero-variance features avoids redundancy, imputing missing values preserves data integrity, standardization ensures comparability across features, and PCA reduces computational complexity while retaining essential information. These steps collectively prepare the dataset for effective and efficient machine learning modeling.

The provided code applies a comprehensive machine learning pipeline for training and evaluating two models, Random Forest and XGBoost, using an imbalanced dataset that has been balanced with SMOTE. First, Synthetic Minority Oversampling Technique (SMOTE) is employed to address class imbalance by generating synthetic samples for the minority class, resulting in a balanced dataset (X_balanced and y_balanced). After balancing, the data is split into training and testing sets (70%-30%) using train_test_split. Two models are trained: a Random Forest Classifier (rf.fit) and an XGBoost Classifier (xgb.fit), both optimized for robust and accurate predictions. The models are evaluated on the test set, where predictions (rf_pred and xgb_pred) are compared against the true labels using various metrics: accuracy, recall, precision, F1 score, and ROC AUC. These metrics provide insights into the models' overall performance, sensitivity to positive cases, precision of positive predictions, balance between precision and recall, and ability to distinguish between classes. This pipeline effectively tackles imbalance and assesses model efficacy for classification tasks.



```
'roc curve to compare both models'
```

```python
rf_probs = rf.predict_proba(X_test)[:, 1]
xgb_probs = xgb.predict_proba(X_test)[:, 1]

rf_fpr, rf_tpr, _ = roc_curve(y_test, rf_probs)
xgb_fpr, xgb_tpr, _ = roc_curve(y_test, xgb_probs)

plt.figure(figsize=(8, 6))
plt.plot(rf_fpr, rf_tpr, label="Random Forest (AUC = {:.2f})".format(roc_auc_score(y_test, rf_probs)))
plt.plot(xgb_fpr, xgb_tpr, label="XGBoost (AUC = {:.2f})".format(roc_auc_score(y_test, xgb_probs)))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

The provided code snippet generates and visualizes the Receiver Operating Characteristic (ROC) curves for the predictions of the Random Forest and XGBoost models to evaluate their performance in distinguishing between classes.

1. Probability Predictions:
• rf.predict_proba(X_test) and xgb.predict_proba(X_test) compute the predicted probabilities for each class on the test data (X_test) for the Random Forest (rf) and XGBoost (xgb) models, respectively. The [:, 1] indexing extracts the probabilities corresponding to the positive class.

2. Calculate ROC Curve:
• roc_curve(y_test, rf_probs) and roc_curve(y_test, xgb_probs) calculate the False Positive Rate (FPR) and True Positive Rate (TPR) at various threshold levels for the test set (y_test). These metrics are essential for plotting the ROC curve.

3. Visualize ROC Curve:
• Aplot is created using matplotlib.
• The plt.plot function is used to plot the ROC curves for both models (rf_fpr vs. rf_tpr and xgb_fpr vs. xgb_tpr), with the Area Under the Curve (AUC) scores displayed in the legend. The AUC values are calculated using roc_auc_score, providing a single scalar metric summarizing the overall performance of each model (closer to 1 indicates better performance).

4. Graph Formatting:
• The x-axis represents the False Positive Rate (proportion of negatives incorrectly classified as positives), while the y-axis represents the True Positive Rate (proportion of positives correctly classified as positives).
• The plot is labeled and titled (ROC Curve) to clearly convey the meaning of the graph.
• The legend distinguishes the models with their respective AUC values.

7.3.Visualization

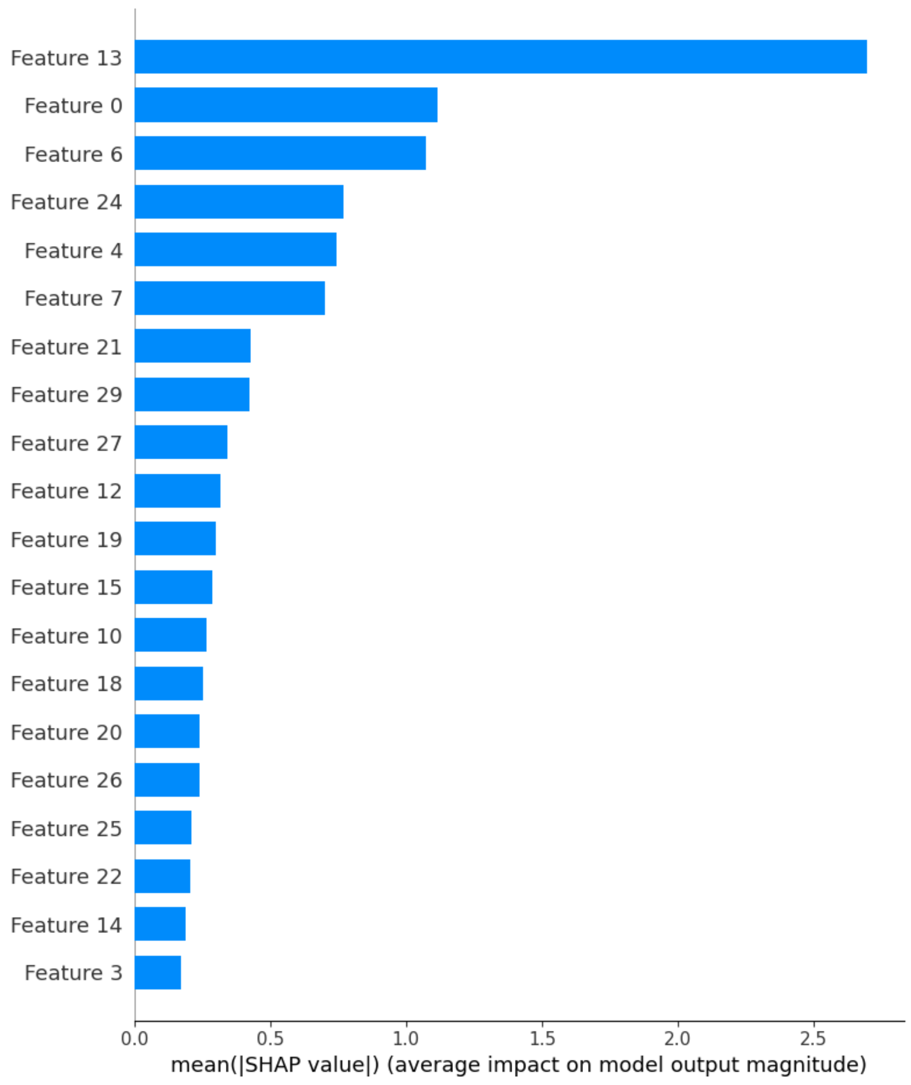
1. Summary Plot (Bar Plot)

The SHAP summary plot (bar plot) displays the global feature importance. Features are ranked based on their mean absolute SHAP values, indicating their average impact on model predictions across all instances. Higher mean absolute SHAP values signify greater influence on the model's output.

Insights:

- The top features contributing most significantly to model predictions are Feature 13, Feature 25, Feature 1, Feature 6, and Feature 10. These features exert a stronger influence compared to others in determining the model's classification.
- The order of these features suggests their relative importance in the model's decision-making process. Further investigation of these specific features is recommended to understand the underlying patterns and relationships within the data.

- Description: The bar chart format of the summary plot aggregates SHAP values across all observations and ranks features based on their mean absolute SHAP value.
- Interpretation:
  - The length of bars indicates the average magnitude of impact a feature has on the model's predictions.
  - This plot is simpler and more intuitive than the beeswarm plot for comparing feature importance.
- Usage: It is particularly useful for quickly identifying the most influential features in the model.



Summary plot

2. Summary Plot (Beeswarm Plot)

The SHAP beeswarm plot provides a more detailed view of feature importance and their impact on individual predictions. Each data point represents a single instance from the test set. The color indicates the feature value (red for high, blue for low), and the horizontal position represents the SHAP value for that feature and instance.

Insights:

- The beeswarm plot reveals the distribution of SHAP values for each feature, highlighting the variability in feature impact across different instances.
- Features with wider distributions of SHAP values tend to have a more variable effect on model predictions.
- Clustering of points along the horizontal axis indicates similar SHAP values for a specific feature across multiple instances.

Description: The beeswarm plot visually summarizes the impact of each feature on the predictions of the model. Each point represents a SHAP value for a single observation, with:

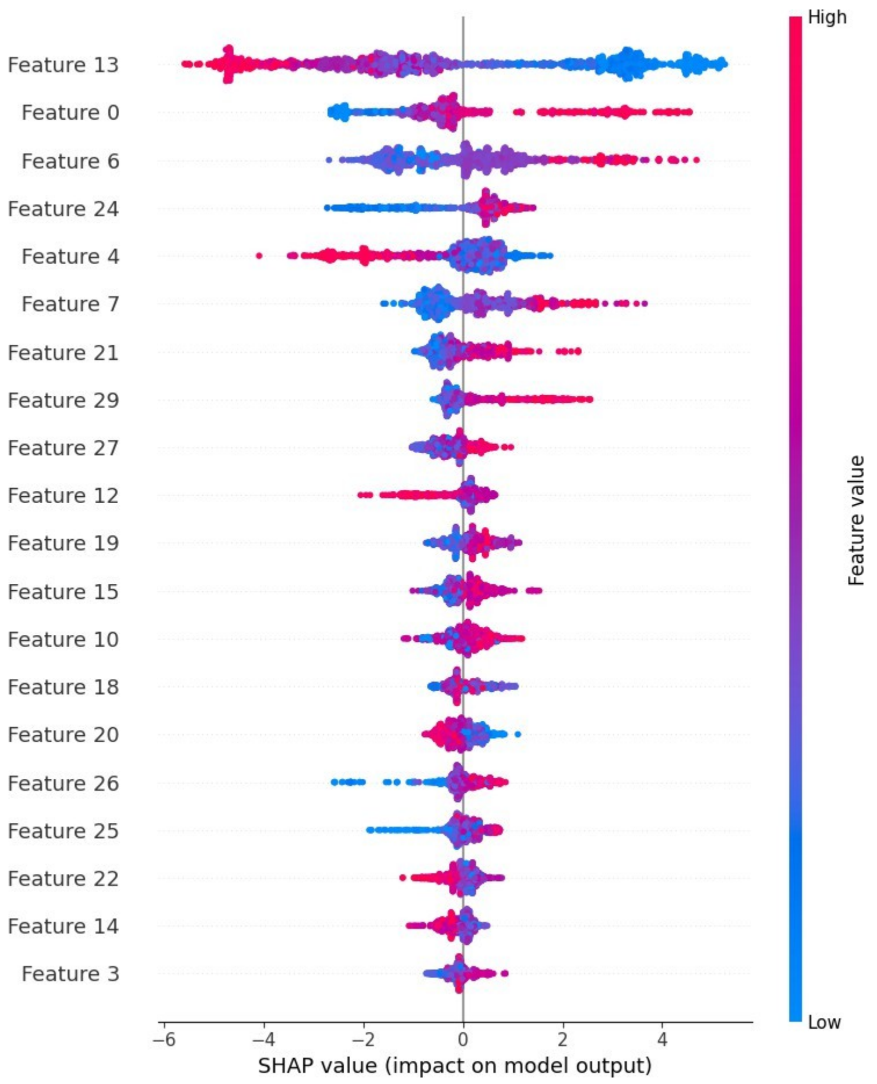
- Position on x-axis: The magnitude of the SHAP value (impact of the feature on prediction).
- Color coding: The feature's value for that observation (e.g., high or low).

Interpretation:

- Features are ranked in descending order of importance based on their average SHAP values.
- The spread of points indicates variability in feature impact.
- Patterns (e.g., clusters or trends in color) reveal how a feature's value influences predictions.

Usage: This plot provides a high-level overview of feature importance and how each feature affects predictions, guiding the analyst to focus on impactful features.

Beeswarm plot



3. Dependence Plot

The SHAP dependence plot illustrates the relationship between a specific feature and its SHAP values. It shows how the feature value influences model predictions, considering the interactions with other features.

Insights:

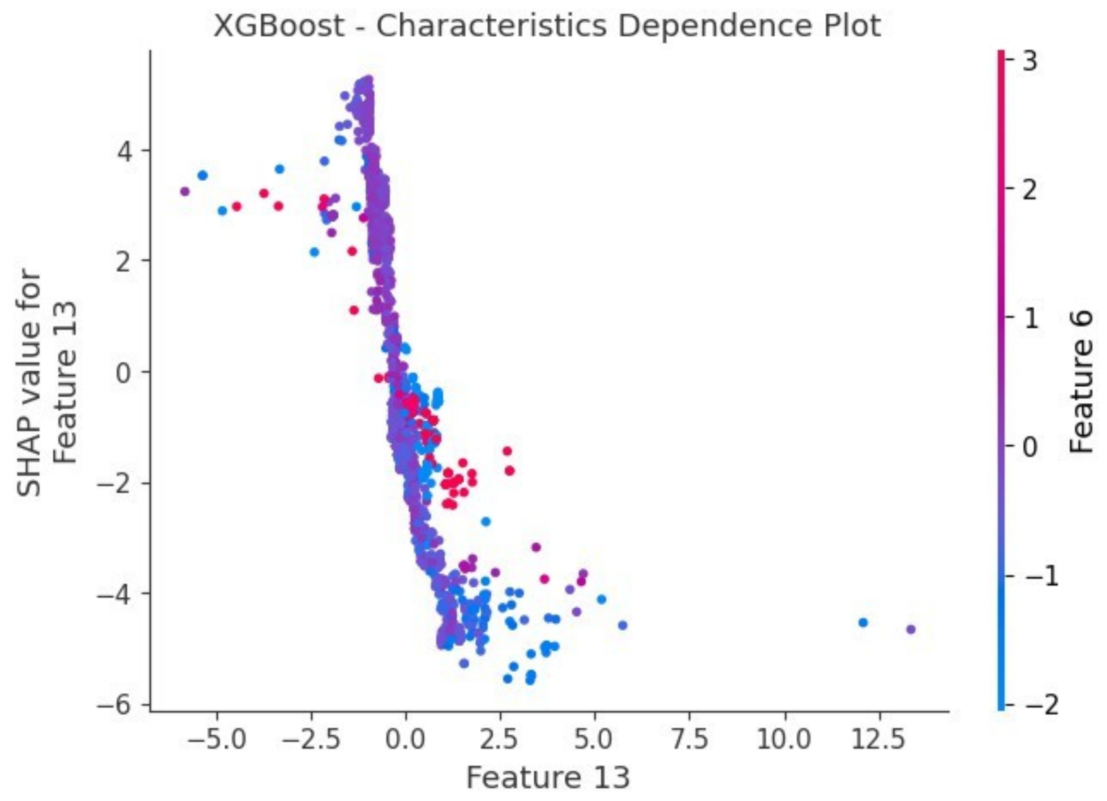
- The dependence plot for 'Feature 13' reveals a positive correlation between feature values and SHAP values.
- Higher values of 'Feature 13' tend to increase the prediction probability for the positive class, as indicated by the positive SHAP values.
- The plot may also highlight interactions with other features through color coding, providing insights into how the relationship between 'Feature 13' and the prediction changes depending on the values of other features.

Description: A dependence plot shows how the SHAP value of a specific feature (e.g., *Feature 13*) varies with its actual value. It optionally displays interactions with a second feature.

Interpretation:

- The x-axis represents the actual values of the selected feature.
- The y-axis shows the SHAP value (feature's contribution to the prediction).
- Color coding reflects the value of a second interacting feature (if automatically identified by SHAP).
- Patterns in the plot reveal whether the relationship is linear, nonlinear, or complex.

Usage: This plot is valuable for understanding how specific feature values influence predictions and f



4. Force Plot

The force plot visualizes the prediction for a single instance, highlighting the contribution of each feature to the final prediction. It shows how features push or pull the prediction away from the base value (the average prediction for the dataset).

Insights:

- Features with large positive SHAP values push the prediction toward a positive class, while features with large negative SHAP values pull the prediction toward a negative class.
- The force plot allows for a detailed understanding of how individual features contribute to a specific prediction, providing valuable insights into model behavior.

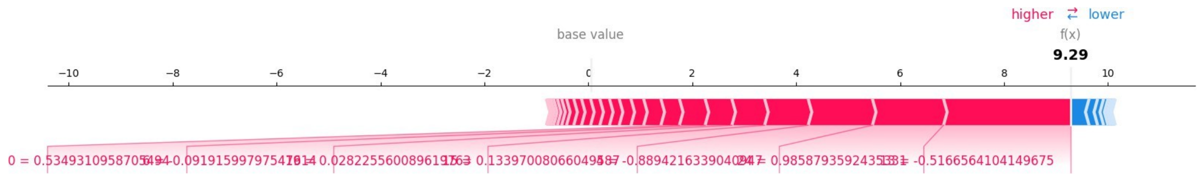
Description: The force plot provides a detailed view of how each feature contributes to the prediction for a specific observation (identified by instance_index).

- Base value: The model's average output (expected value).
- SHAP values: Shown as arrows pushing the prediction up or down from the base value.

Interpretation:

- Features pushing the prediction higher are in red (positive SHAP values).
- Features pulling the prediction lower are in blue (negative SHAP values).
- The length of arrows corresponds to the magnitude of each feature's impact.

Usage: This plot is ideal for analyzing individual predictions, especially in use cases like debugging or explaining decisions.



Shap force plot for instance-0

5. Decision Plot

The decision plot shows the model's decision-making process for multiple instances. It illustrates how features influence the model's predictions across various scenarios.

Insights:

- The decision plot provides a comprehensive view of the model's behavior, revealing how different features influence predictions across various instances.
- This visualization can help understand complex interactions and identify patterns in the model's decision-making process.

Description: The decision plot visualizes how features cumulatively influence predictions for one or multiple instances.

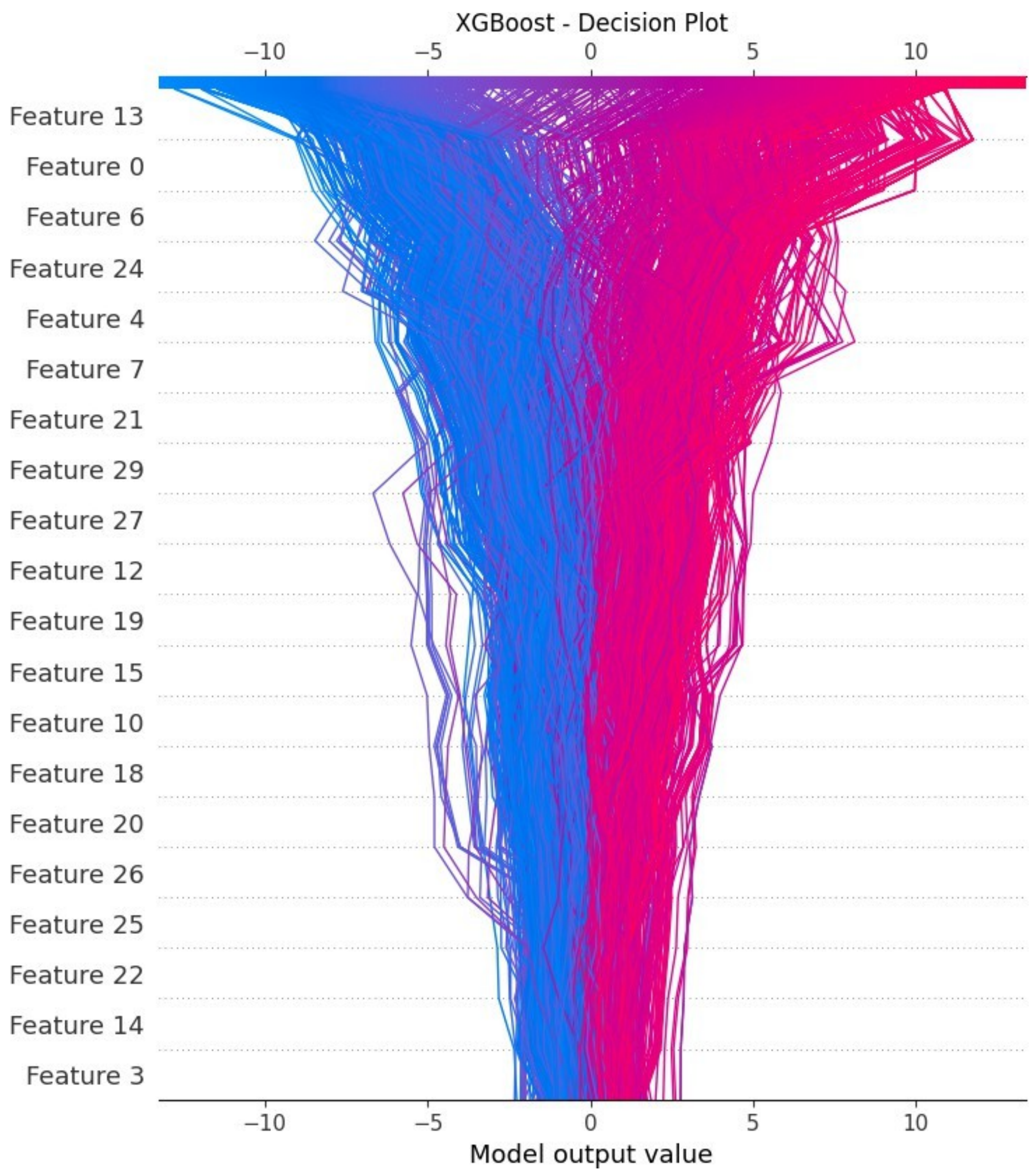
- The x-axis shows cumulative SHAP values (from base value to final prediction).
- The y-axis tracks the feature contributions.

Interpretation:

- The progression of lines reveals the sequence of features influencing the prediction.
- This is particularly helpful for understanding why a model arrived at a specific decision.

Usage: Useful for tracing prediction logic and comparing how multiple instances are treated by the model.

6.



WaterfallPlot

The waterfall plot illustrates the contribution of each feature to a single prediction, similar to the force plot. It shows how the features cumulatively affect the prediction, starting from the base value.

Insights:

- The waterfall plot helps visualize the step-by-step influence of features on the prediction.

- It provides a clear explanation of how the model arrives at a specific prediction for an individual instance.

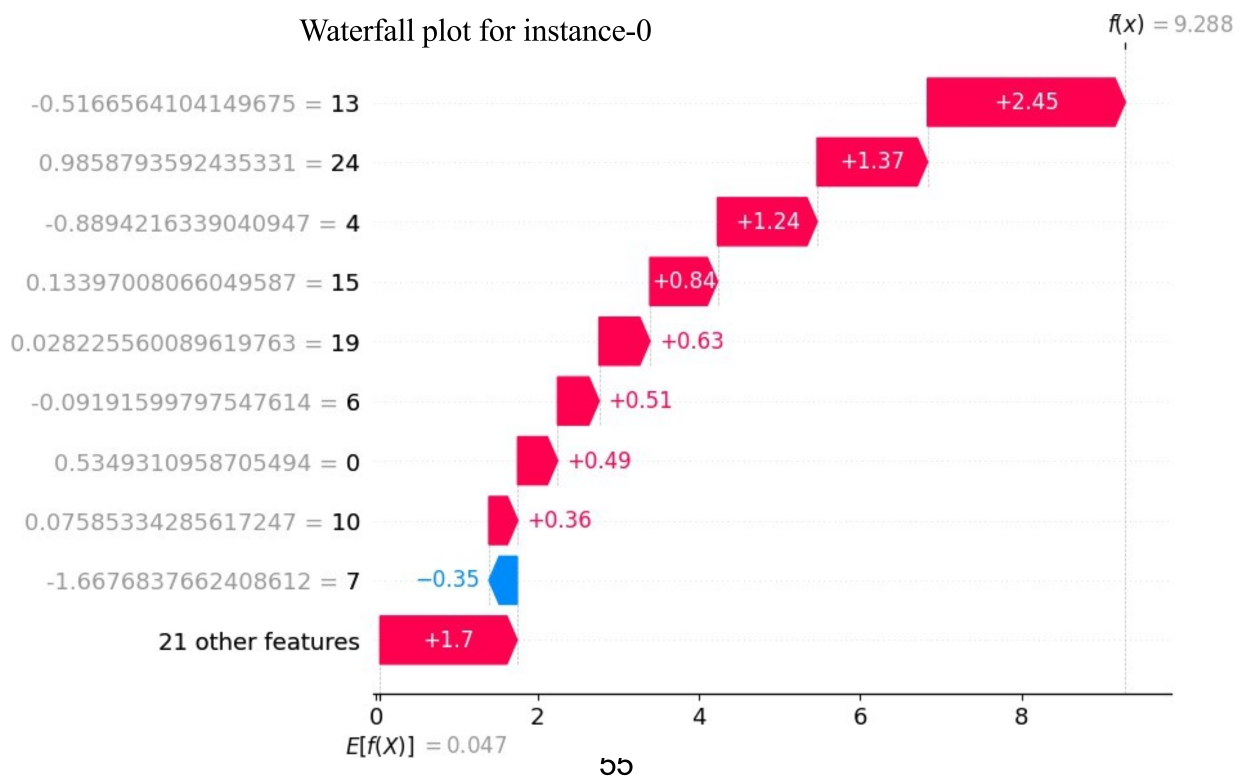
Description: The waterfall plot explains the contribution of each feature to the prediction of a specific instance.

- Base value: The model's average prediction.
- Step changes: Individual SHAP values adding to or subtracting from the base value to reach the final prediction.

Interpretation:

- Positive contributions are shown in red (increasing the prediction).
- Negative contributions are shown in blue (decreasing the prediction).
- This plot provides a step-by-step breakdown of how the prediction was made.

Usage: It is particularly effective in creating clear, instance-specific explanations.



Conclusion:

The SHAP analysis provides valuable insights into the XGBoost model's behavior, feature importance, and their influence on predictions. The top contributing features identified in the summary plot should be further investigated to understand their role in the classification process. The dependence, force, decision, and waterfall plots offer detailed explanations of individual predictions, enabling a deeper understanding of the model's decision-making process. These insights can be used to improve model performance, debug potential issues, and gain a better understanding of the underlying patterns within the data.

8. CONCLUSION

Summary of the Project's Achievements:

- Malware Detection Model: The project successfully developed a malware detection model using machine learning techniques, primarily Random Forest, to classify files as either malware or benign based on static features.

- Explainability: The integration of SHAP (SHapley Additive exPlanations) provides transparency by explaining the model's predictions, helping users understand the features that contribute to the decision-making process.

- PCA for Feature Selection: The application of Principal Component Analysis (PCA) effectively reduced the feature space, making the model more efficient while retaining the most important information.

- Balanced Dataset: The use of SMOTE for balancing the dataset improved the model's ability to detect malware without biasing the classification towards the majority class.

Effectiveness and Accuracy:

- Emphasize how these combined techniques have led to a robust and interpretable model for malware detection. Highlight any performance metrics (accuracy, precision, recall, etc.) you have obtained from testing

  or validation, showing that the model performs well on the provided dataset.

Explainability and Trust:

- Discuss the importance of explainability in machine learning models, especially in security applications like malware detection. The transparency offered by SHAP enables security professionals to trust the

  model's decisions, making it more valuable in real-world applications.

Future Developments

Improvement of Model Performance:

- Feature Engineering: Suggest further research into feature engineering to explore additional features that could improve classification accuracy. For example, combining static features with dynamic analysis (behavioral features during executio5n9) might enhance the detection of sophisticated malware.

- Model Tuning: Further hyperparameter tuning of the Random Forest model could improve its performance. Experimenting with other machine learning models (e.g., XGBoost, Support Vector Machines) might also provide better results.

- Ensemble Learning: You could consider combining multiple models using an ensemble approach to further boost classification accuracy, using techniques like bagging or boosting.

Scaling and Real-World Deployment:

- Discuss how the current system can be scaled for real-time malware detection in large-scale environments. For example, integrating the model into an antivirus or endpoint protection system could improve the detection of new and evolving malware threats.

- Consider the challenges of deploying the system in a production environment, such as model robustness, handling unseen malware samples, or computational efficiency for real-time detection.

Integration with Other Techniques:

- Suggest the integration of dynamic analysis alongside static features to increase the accuracy of the detection. Dynamic analysis could involve analyzing the runtime behavior of files to detect malicious activities that
static features might miss.

- Deep learning models (e.g., convolutional neural networks) could be explored for more complex patterns, especially if large datasets are available.

Further Research Areas:

- Research into adversarial attacks on malware detection systems could be explored to make the model more resilient to malware designed to evade detection by machine learning models.

- Investigating the explainability of deep learning models in malware detection, as traditional models like Random Forest are more interpretable, while deep learning models offer improved accuracy at the
cost of transparency.

9. REFERENCES

[1]“Malware detection using machine learning| International Journal of Innovative Science and Research Technology,” *Doi.org*, 2023. https://doi.org/10.38124/ijisrt/IJISRT24APR1102 (accessed Nov. 20, 2024).

[2]Q. Zhao *et al.*, “Malware Detection and Analysis based on AI Algorithm,” pp. 1–6, Mar. 2024, doi: https://doi.org/10.1109/icdcot61034.2024.10515944.

[3]P. Kumar *et al.*, “Explainable artificial intelligence envisioned security mechanism for cyber threat hunting,” *SECURITY AND PRIVACY*, Mar. 2023, doi: https://doi.org/10.1002/spy2.312.

[4]Alexandru-Radu BELEA, “Methods for Detecting Malware Using Static, Dynamic and Hybrid Analysis,” May 2023, doi: https://doi.org/10.19107/cybercon.2023.34.

[5]“A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques | IEEE Conference Publication | IEEE Xplore,” *ieeexplore.ieee.org*. https://ieeexplore.ieee.org/document/9591763

[6]A. Kumar, K. S. Kuppusamy, and G. Aghila, “A learning model to detect maliciousness of portable executable using integrated feature set,” *Journal of King Saud University - Computer and Information Sciences*, vol. 31, no. 2, pp. 252–265, Apr. 2019, doi: https://doi.org/10.1016/j.jksuci.2017.01.003.

[7]A. Fatima, R. Maurya, M. K. Dutta, R. Burget, and J. Masek, “Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning,” *IEEE Xplore*, Jul. 01, 2019. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8769039 (accessed Jul. 10, 2022).

[8]Z. Ding, H. Chen, and L. Zhou, “Using shapely values to define subgroups of forecasts for combining,” *Journal of Forecasting*, vol. 42, no. 4, pp. 905–923, Oct. 2022, doi: https://doi.org/10.1002/for.2920.

[9]M. Ketenci, I. Urteaga, V. A. Rodriguez, N. Elhadad, and A. Perotte, “Variational Shapley Network: A Probabilistic Approach to Self-Explaining Shapley values with Uncertainty Quantification,” *arXiv.org*, 2024. https://arxiv.org/abs/2402.04211 (accessed Nov. 20, 2024).

[10]A. Galli, Valerio La Gatta, V. Moscato, M. Postiglione, and Giancarlo Sperlì, “Explainability in AI-based Behavioral Malware Detection Systems,” *Computers & security*, pp. 103842–103842, Apr. 2024, doi: https://doi.org/10.1016/j.cose.2024.103842.

[11]D. Soi, A. Sanna, D. Maiorca, and G. Giacinto, “Enhancing android malware detection explainability through function call graph APIs,” *Journal of Information Security and Applications*, vol. 80, p. 103691, Feb. 2024, doi: https://doi.org/10.1016/j.jisa.2023.103691.

APPENDIXA– SAMPLE CODE

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from xgboost import XGBClassifier
from sklearn.metrics import (
    accuracy_score, recall_score, precision_score,
f1_score, roc_auc_score, roc_curve
)
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import shap
# Load the dataset
data = pd.read_csv("ClaMP_Raw-5184.csv")

# Inspect the data
print(data.head())
print(data.info())
print(data.describe())

# Check for missing values
print(data.isnull().sum())

variances = np.var(X, axis=0)
zero_variance_features = X.columns[variances == 0]
print("Zero Variance Features:",
zero_variance_features)
X = X.drop(columns=zero_variance_features)
```

```
# Drop columns with all missing values
data = data.dropna(axis=1, how="all")

# Impute missing values
imputer = SimpleImputer(strategy='mean')
data_imputed =
pd.DataFrame(imputer.fit_transform(data),
columns=data.columns)


# Separate features (X) and target (y)
X = data_imputed.drop(columns=["class"])  # Replace
with your target column name
y = data_imputed["class"]

# Calculate variances and identify zero-variance
features
variances = np.var(X, axis=0)
zero_variance_features = X.columns[variances == 0]

# Remove zero-variance features
X = X.drop(columns=zero_variance_features)

# Standardize features, excluding zero-variance
features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% variance
X_pca = pca.fit_transform(X_scaled)
print(f"Original shape: {X_scaled.shape}, Reduced
shape: {X_pca.shape}")

smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X_pca, y)
print(f"Class distribution after SMOTE:
{np.bincount(y_balanced)}")

# Split data
X_train, X_test, y_train, y_test =
train_test_split(X_balanced, y_balanced, test_size=0.3,
random_state=42)

# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# Train XGBoost
xgb = XGBClassifier(use_label_encoder=False,
eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)

# Predictions
rf_pred = rf.predict(X_test)
xgb_pred = xgb.predict(X_test)

# Metrics
def evaluate_model(y_true, y_pred, model_name):
    print(f"\n{model_name} Performance:")
    print(f"Accuracy: {accuracy_score(y_true,
y_pred):.2f}")
    print(f"Recall: {recall_score(y_true,
y_pred):.2f}")
    print(f"Precision: {precision_score(y_true,
y_pred):.2f}")
    print(f"F1 Score: {f1_score(y_true, y_pred):.2f}")
    print(f"ROC AUC: {roc_auc_score(y_true,
y_pred):.2f}")

evaluate_model(y_test, rf_pred, "Random Forest")
evaluate_model(y_test, xgb_pred, "XGBoost")

rf_probs = rf.predict_proba(X_test)[:, 1]
xgb_probs = xgb.predict_proba(X_test)[:, 1]

rf_fpr, rf_tpr, _ = roc_curve(y_test, rf_probs)
xgb_fpr, xgb_tpr, _ = roc_curve(y_test, xgb_probs)

plt.figure(figsize=(8, 6))
plt.plot(rf_fpr, rf_tpr, label="Random Forest (AUC =
{:.2f})".format(roc_auc_score(y_test, rf_probs)))
plt.plot(xgb_fpr, xgb_tpr, label="XGBoost (AUC =
{:.2f})".format(roc_auc_score(y_test, xgb_probs)))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()

import shap

# Initialize SHAP explainer for XGBoost
xgb_explainer = shap.TreeExplainer(xgb)

# Compute SHAP values for X_test
xgb_shap_values = xgb_explainer.shap_values(X_test)
```

```
# Check shape of SHAP values
print("XGB SHAP values shape:", xgb_shap_values.shape)

# Ensure SHAP values align with X_test
if xgb_shap_values.shape[1] != X_test.shape[1]:
    raise ValueError("XGBoost SHAP values feature
dimensions do not match X_test.")

# Plot SHAP summary for XGBoost
shap.summary_plot(xgb_shap_values, X_test,
plot_type="bar", title="XGBoost Feature Importance")


import shap
import matplotlib.pyplot as plt

# Initialize SHAP explainer
xgb_explainer = shap.TreeExplainer(xgb)

# Compute SHAP values for the test set
xgb_shap_values = xgb_explainer.shap_values(X_test)


shap.summary_plot(xgb_shap_values, X_test,
title="XGBoost - Summary Plot (Beeswarm)")




# Show all plots (if running outside Jupyter Notebook)
plt.show()
```

```
# Dependence Plot for 'CreationYear'
shap.dependence_plot("Feature 13", xgb_shap_values,
X_test, title="XGBoost - Characteristics Dependence
Plot")


# Show all plots (if running outside Jupyter Notebook)
plt.show()


# Force Plot for a single instance
instance_index = 0  # Replace with the index of the
instance you want to analyze
shap.force_plot( xgb_explainer.expected_valu
    e, xgb_shap_values[instance_index],
    pd.DataFrame(X_test).iloc[0,:].astype(str),  #
Convert numeric features to string
    matplotlib=True,
)


# Show all plots (if running outside Jupyter Notebook)
plt.show()



# Decision Plot for the entire dataset
shap.decision_plot(
    xgb_explainer.expected_value,
    xgb_shap_values,
    X_test,
    title="XGBoost - Decision Plot"
)


# Show all plots (if running outside Jupyter Notebook)
plt.show()



# Waterfall Plot for a single instance
shap.waterfall_plot(
    shap.Explanation( values=xgb_shap_values
        [0],
        base_values=xgb_explainer.expected_value,

data=pd.DataFrame(X_test).iloc[0,:].astype(str),  #
Convert numeric features to string
        feature_names=X_test.columns if hasattr(X_test,
'columns') else None # Get column names if available
    ),
)


# Show all plots (if running outside Jupyter Notebook)
plt.show()
```