Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Machine Learning Model Performance Analysis and Optimization for Android Malware Detection Using the CHIMERA Dataset

KIRUBAVATHI.G[1],JASIN DAVID JAYA SINGH[1], JIVESH K[1],LOGANAND S[1], XXX XXX[2], XXX XXX[3], HABIB HAMAM [4,5,6,7]

[1]Department of Mathematics, Amrita School of Physical Sciences, Amrita Vishwa Vidyapeetham, Tamil Nadu,India
[2]xxx, xxx
[3]xxx, xxx
[4]Faculty of Engineering, Uni de Moncton, Moncton, NB E1A3E9, Canada, Habib.Hamam@umoncton.ca
[5]School of Electrical Engineering, University of Johannesburg, Johannesburg 2006, South Africa
[6]International Institute of Technology and Management (IITG), Av. Grandes Ecoles, Libreville BP 1989, Gabon
[7]Bridges for Academic Excellence - Spectrum, Tunis, Center-ville, Tunisia

Corresponding author:Kirubavathi.G (e-mail: g_kirubavathi@cb.amrita.edu).

ABSTRACT Cybersecurity threats have escalated with the increasing complexity of malicious software, necessitating robust machine learning-driven malware detection systems. This study leverages the CHIMERA dataset from IEEE, a comprehensive collection of malware and benign software attributes, to evaluate multiple classification models for effective malware detection. The dataset comprises 21,992 samples with 16,985 features, including API calls, permissions, and binary attributes. We employed Logistic Regression, Random Forest, Gradient Boosting, XGBoost, and Neural Networks to classify malware, following preprocessing techniques such as feature selection, label encoding, and standardization. Feature importance analysis identified key attributes influencing malware classification, including permissions and API calls. After hyperparameter tuning, XGBoost demonstrated the highest classification performance with an accuracy of 0.769, outperforming other models. Additionally, interpretability methods like SHAP and LIME provided insights into feature contributions, aiding in understanding malware behavior. Our analysis revealed that permissions related to phone state, SMS access, and system monitoring are particularly indicative of malicious behavior. These findings highlight the potential of ensemble learning and explainable AI in enhancing malware detection systems. Future work will focus on integrating deep learning models, addressing adversarial attacks, and refining feature engineering to further improve classification accuracy.

INDEX TERMS Android Malware Detection, Machine Learning, XGBoost, CHIMERA Dataset, SHAP, LIME, Cybersecurity, Feature Selection

I. INTRODUCTION

A. CONTEXT

The rapid evolution of cyber threats, particularly malware, poses significant challenges to cybersecurity. Malware, including viruses, trojans, and ransomware, is designed to exploit system vulnerabilities, leading to data breaches, financial losses, and operational disruptions. Traditional signature-based detection methods struggle to keep pace with emerging threats, necessitating the development of machine learning-based approaches for more adaptive and robust malware detection [**?**].

The CHIMERA dataset from IEEE [1] serves as a valuable resource for researching malware classification. It comprises 21,992 samples with 16,985 extracted features, including API calls, permissions, and system interactions. These attributes provide critical insights into the behavioral patterns of malicious and benign software. Machine learning models, trained on such datasets, can identify subtle patterns indicative of malicious activity, thereby enhancing detection accuracy and reducing false positives [2].

The increasing sophistication of malware has rendered traditional detection methods inadequate. Static analysis techniques, which examine code without execution, often fail to detect obfuscated malware. Similarly, dynamic analysis, which observes behavior during execution, faces challenges in creating comprehensive execution environments. Machine learning offers a promising alternative by learning patterns from large datasets of known malware samples, enabling the detection of previously unseen threats based on similar behavioral or structural patterns.

This study evaluates multiple machine learning models—Logistic Regression, Random Forest, Gradient Boosting, XGBoost [3], and Neural Networks—to classify malware effectively. Feature selection techniques were employed to extract the most relevant attributes, improving model interpretability and efficiency [4]. Additionally, explainability techniques such as SHAP (SHapley Additive Explanations) [5] and LIME (Local Interpretable Model-agnostic Explanations) [6] were used to analyze feature contributions and improve transparency in decision-making.

B. RESEARCH OBJECTIVES

The primary objectives of this research include evaluating the effectiveness of various machine learning models in detecting malware using the CHIMERA dataset, considering both traditional classifiers and deep learning approaches. This study aims to identify the most significant features that contribute to accurate malware classification, including static, dynamic, and behavioral indicators. Additionally, it analyzes the performance trade-offs between different models in terms of accuracy, precision, recall, F1-score, computational efficiency, and scalability across large datasets.

Another key objective is to develop interpretable models that not only detect malware but also provide insights into the detection rationale through explainable AI techniques such as SHAP and LIME. Furthermore, this research seeks to establish a robust methodology for malware detection that can adapt to evolving threats by incorporating continual learning and adversarial defense mechanisms. It also explores the comparative effectiveness of supervised, unsupervised, and hybrid learning approaches to enhance detection capabilities against novel malware variants. Finally, the study investigates the integration of malware detection techniques into real-time cybersecurity frameworks for proactive threat mitigation.

C. SIGNIFICANCE OF THE STUDY

This research makes several significant contributions to the field of cybersecurity by providing a comprehensive comparative analysis of various machine learning models on the CHIMERA dataset for malware classification. It identifies the most effective classification techniques and their applicability in real-world cybersecurity scenarios. Additionally, the study conducts feature selection and importance analysis to determine key malware indicators that can guide security practitioners in designing efficient and targeted detection mechanisms. By implementing interpretability techniques such as SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-Agnostic Explanations), this research enhances trust in automated detection systems by providing transparency into model predictions. Moreover, it offers guidelines for developing robust malware detection systems that balance accuracy with computational efficiency, ensuring feasibility for deployment in resource-constrained environments.

The study also provides valuable insights into the behavioral patterns of modern malware, particularly in Android environments, addressing the evolving nature of cyber threats. Furthermore, it explores the integration of adversarial defense mechanisms to enhance model resilience against evasion tactics employed by sophisticated malware. By considering real-time detection scenarios, this research contributes to the development of adaptive cybersecurity frameworks capable of mitigating emerging threats proactively.

D. STRUCTURE OF THE ARTICLE

The rest of the paper is structured as follows: Section 2 discusses related work in machine learning-based malware detection. Section 3 details the dataset and preprocessing techniques. Section 4 describes the methodology, including model selection and evaluation. Section 5 presents results and discussions, while Section 6 concludes with future research directions.

II. RELATED WORK

A. CURRENT RESEARCH

Machine learning-based malware detection has gained significant attention in recent years, with researchers exploring various approaches to improve detection accuracy and adaptability. This section reviews key studies in the field, highlighting their methodologies, findings, and limitations.

Early malware detection systems relied primarily on signature-based methods, which involve matching file signatures with known malware patterns [11]. While effective for known threats, these methods struggle with detecting novel or obfuscated malware. Heuristic-based approaches emerged as an alternative, using rule-based systems to identify suspicious behaviors [12]. However, these approaches often generate high false positive rates and require constant updating to remain effective.

The application of machine learning to malware detection has evolved significantly over the past decade. Ucci et al. [2] provided a comprehensive survey of machine learning techniques for malware analysis, categorizing them based on feature extraction methods, learning algorithms, and evaluation metrics. Their study highlighted the growing trend toward ensemble methods and deep learning approaches. Recent studies have also explored AI-driven intrusion detection systems for critical infrastructure cybersecurity, showcasing the effectiveness of big data analytics and machine learning in identifying complex attack patterns [13].

Pascanu et al. [8] demonstrated the effectiveness of recurrent neural networks (RNNs) in malware classification,

particularly for analyzing sequential data such as API call sequences. Their model achieved high accuracy by capturing temporal dependencies in malware behavior.

Ensemble learning methods have shown promising results in malware detection. Chen and Guestrin [3] introduced XGBoost, a gradient boosting framework that has demonstrated superior performance in various machine learning competitions. In the context of malware detection, XGBoost has been particularly effective due to its ability to handle high-dimensional data and capture complex feature interactions. Furthermore, machine learning combined with blockchain technologies has been proposed as a resilient solution to cybersecurity threats in smart grid and IoT environments [14].

Feature selection plays a crucial role in malware detection, as highlighted by Chandrasekaran et al. [4]. Their study evaluated various feature selection techniques, including filter, wrapper, and embedded methods, for malware detection. They found that appropriate feature selection not only improves classification accuracy but also enhances model efficiency. Additionally, IoT-based security attack detection has gained increased importance, with emerging machine learning techniques being deployed to improve defense mechanisms against complex, evolving threats [15].

As machine learning models become increasingly complex, the need for explainability has grown. Lundberg and Lee [5] introduced SHAP (SHapley Additive exPlanations), a unified framework for interpreting model predictions. Similarly, Ribeiro et al. [6] proposed LIME (Local Interpretable Model-agnostic Explanations) to explain individual predictions. These techniques have been increasingly applied in cybersecurity to provide transparency in malware detection decisions.

Recent studies have explored graph-based approaches for malware analysis. Zhang et al. [9] proposed a Graph Neural Network (GNN) framework for Android malware detection based on API call graphs. Their approach effectively captures the structural information of malware behavior, outperforming traditional machine learning methods.

Adversarial attacks pose a significant challenge to machine learning-based malware detection systems. Hu and Tan [10] demonstrated how Generative Adversarial Networks (GANs) could be used to generate adversarial malware examples that evade detection. This highlights the importance of developing robust models that can withstand such attacks.

Despite significant advances in machine learning-based malware detection, several challenges remain unaddressed. One major limitation is the limited research on the interpretability of complex models, making it difficult for cybersecurity practitioners to understand and trust automated detection decisions. Additionally, insufficient attention has been given to feature importance analysis across different model types, which is crucial for identifying the most relevant attributes that contribute to effective classification.

Another critical gap is the lack of comprehensive comparative studies utilizing large and diverse datasets like CHIMERA, which are essential for evaluating model generalizability and robustness. Furthermore, many existing studies focus on theoretical performance metrics without considering practical deployment constraints, such as computational efficiency, adaptability to evolving malware variants, and real-time detection capabilities. Addressing these gaps is vital for developing more transparent, scalable, and reliable malware detection systems that can be effectively deployed in real-world cybersecurity frameworks.

This study addresses these gaps by conducting a comprehensive evaluation of machine learning models on the CHIMERA dataset, with a focus on model interpretability and feature importance analysis.

B. RESEARCH GAP

Despite notable advancements in machine learning-based malware detection, several critical gaps remain unaddressed. First, most existing studies focus predominantly on classification accuracy without sufficient emphasis on model interpretability, limiting their applicability in real-world cybersecurity contexts where explainable decisions are essential for incident response and trust building. Additionally, while numerous works evaluate malware detection models on standard or limited datasets, there is a lack of comprehensive investigations using large-scale and diverse datasets like CHIMERA that better represent real-world scenarios.

Another important gap is the limited integration of feature importance analysis across multiple machine learning models to identify universally critical indicators for malware classification. Without robust feature selection methodologies, detection systems risk overfitting, reduced scalability, and increased computational demands, particularly in resource-constrained environments.

Moreover, many prior studies do not adequately address the evolving sophistication of malware that leverages adversarial techniques to evade detection. The development of resilient models capable of adapting to adversarial attacks remains an underexplored area requiring urgent attention.

Finally, there is a shortage of research focused on embedding malware detection systems within real-time cybersecurity frameworks, considering computational efficiency, scalability, and timely threat mitigation as core requirements. Addressing these gaps is crucial for the development of next-generation malware detection solutions that are not only accurate but also explainable, robust, and deployable in dynamic and adversarial environments.

III. METHODOLOGY

Our machine learning methodology for the Chimera dataset (21,992 samples; 16,985 features) follows a systematic workflow beginning with data preprocessing through feature encoding, scaling, and selection. We evaluate multiple models including Logistic Regression, Random Forest, Gradient Boosting, XGBoost, and Neural Networks, optimizing performance via GridSearchCV and cross-validation. Feature importance is analyzed using SHAP and LIME, while models are evaluated on accuracy, precision-recall metrics, and ROC-AUC. Results

compare model performance, highlighting improvements from tuning and key features identified, with future work focusing on deep learning integration and adversarial robustness.

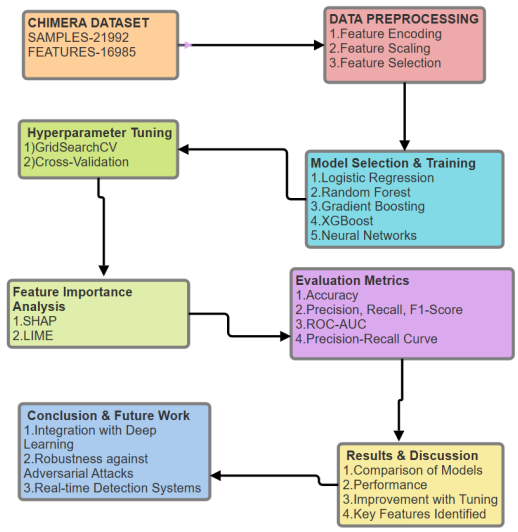

FIGURE 1: Block Diagram of the Methodology

A. DESCRIPTION OF THE CHIMERA DATASET

The CHIMERA dataset from IEEE is a comprehensive collection of malware and benign software characteristics, consisting of 21,992 samples with 16,985 extracted features. These features encompass various aspects of software behavior, including API calls, permissions, system behavior attributes, and binary attributes. API calls capture sequences of system interactions invoked by the software, providing insights into its operational characteristics. Permissions denote the access rights requested by applications, which are particularly relevant for Android malware analysis.

System behavior attributes include execution patterns such as file modifications, registry changes, and network requests, enabling behavioral analysis of potential threats. Additionally, binary attributes consist of low-level features extracted from binary code analysis, including opcodes, function calls, and structural characteristics. Each sample in the dataset is labeled as either malicious (1) or benign (0), making it a binary classification problem. The dataset is relatively balanced, with 10,860 malicious samples (49.4%) and 11,132 benign samples (50.6%), ensuring fair model evaluation. It contains both categorical and numerical features, with significant sparsity in some dimensions, necessitating effective preprocessing.

The raw dataset undergoes several preprocessing steps to ensure data quality and model efficiency. Duplicate samples are removed to prevent data leakage, and missing values in feature vectors are handled appropriately. Categorical variables are encoded using label encoding, while non-predictive features such as SHA256 hashes are eliminated. To ensure uniformity in feature representation, standardization is applied using StandardScaler, normalizing all features to zero mean and unit variance. These preprocessing steps are crucial for enhancing model performance and ensuring robust malware detection.

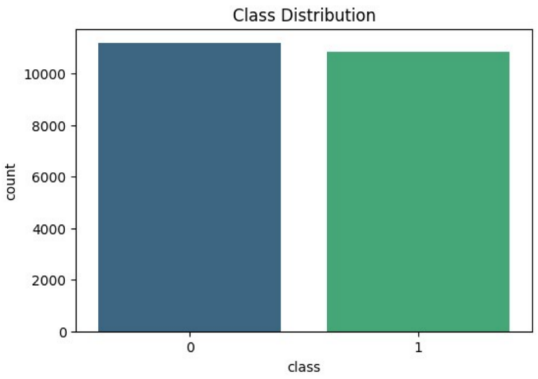

FIGURE 2: Overview of the CHIMERA dataset, showing class distribution and feature count. The dataset is relatively balanced, with 49.4% malicious samples and 50.6% benign samples.

B. DATA PREPROCESSING

A robust preprocessing pipeline was developed to prepare the CHIMERA dataset for machine learning models as shown in algorithm 1. The pipeline addressed several challenges inherent in the dataset, including high dimensionality, feature sparsity, and diverse feature types. Categorical variables were encoded using label encoding to convert them into numerical representations. While one-hot encoding is often preferred for categorical variables, the high cardinality of some features made this approach impractical due to the potential explosion in dimensionality.

Feature scaling was applied using StandardScaler to normalize numerical values, ensuring that all features contributed equally to the distance measures used by machine learning algorithms. This was particularly important for models sensitive to feature scales, such as Neural Networks and Logistic Regression.

Algorithm 1 Data Preprocessing Pipeline

```
Import raw CHIMERA dataset
Check for missing values and data integrity
Encode categorical features using label encoding
Apply feature scaling (StandardScaler) to normalize numerical values
Perform feature selection:
    Apply Random Forest for initial feature importance ranking
    Select top k features based on importance scores
    Validate feature subset using cross-validation
Split data into training (80%) and testing (20%) sets
Apply stratified sampling to maintain class distribution
return Preprocessed training and testing sets
```

C. FEATURE SELECTION

Given the high dimensionality of the CHIMERA dataset, which contains 16,985 features, feature selection was crucial for improving model efficiency and interpretability. A multi-stage feature selection approach was employed to retain the most relevant features while reducing computational overhead. Initially, variance thresholding was applied to eliminate low-variance features that provided limited discriminatory power. Following this, correlation analysis was conducted to identify and remove highly correlated features, reducing multicollinearity and preventing redundant information from affecting model performance. To further refine the feature set, model-based selection using tree-based algorithms such as Random Forest and Gradient Boosting was performed, ranking features by importance and selecting the top contributors. Additionally, Recursive Feature Elimination (RFE) was implemented for selected models, iteratively removing the least significant features while monitoring performance to ensure optimal feature retention. Feature selection techniques have been widely recognized for enhancing malware detection models' accuracy and computational efficiency, particularly in cybersecurity applications [16].

The optimal number of features was determined through cross-validation, balancing model complexity with predictive performance. As a result, the final feature set consisted of the top 500 most important features, which retained the majority of the dataset's predictive power while significantly reducing computational requirements. A key insight from this process was the dominance of specific Android permissions in malware classification. In particular, `read_phone_state` emerged as the most critical feature, with an importance score nearly three times higher than the second-ranked feature, highlighting its role in enabling device identification for potential targeted attacks. Additionally, SMS-related permissions such as `send_sms`, `receive_sms`, and `read_sms` were found to be clustered among the top-ranked features, indicating that SMS functionality remains a primary attack vector for Android malware. Interestingly, several numerical attributes interspersed with named permissions suggested that specific API call sequences also played a significant role in malware detection. The distribution of feature importance values exhibited a steep initial decline, indicating that a small subset of highly predictive features contributed significantly, while lower-ranked features collectively added marginal value to the classification task. These findings underscore the necessity of feature selection in malware detection, ensuring that models remain both interpretable and computationally efficient.

TABLE 1: Top 10 Important Features for Malware Classification

Feature Name	Importance
android.permission.read_phone_state	0.013001
android.permission.send_sms	0.004325
android.intent.action.package_added	0.003270
android.permission.get_tasks	0.003265
android.permission.receive_sms	0.002425
3124_r	0.002303
android.intent.action.user_present	0.002292
389_r	0.001828
android.permission.read_sms	0.001802
1524_r	0.001778

TABLE 2: Top 10 Important Features for Android Malware Detection. The visualization shows the relative importance of Android permissions and intents, with `read_phone_state` having significantly higher importance than other features.

D. MODEL TRAINING AND EVALUATION

To develop an effective malware detection system, multiple classification algorithms were trained and evaluated, including Logistic Regression, Random Forest, Gradient Boosting, XGBoost, and a Neural Network based on a Multi-layer Perceptron (MLP). Each model was optimized through hyperparameter tuning using techniques such as Grid Search and Random Search to achieve the best performance. Given the high dimensionality of the CHIMERA dataset, feature selection techniques were applied before training to enhance model efficiency and reduce overfitting.

Performance evaluation was conducted using a comprehensive set of metrics to ensure a robust assessment of each model's effectiveness. Traditional metrics such as Accuracy, Precision, Recall, and F1-Score were calculated to measure overall classification performance. Additionally, ROC-AUC was employed for threshold-independent evaluation, providing insights into the model's ability to distinguish between malicious and benign samples across varying decision thresholds. Precision-Recall curves were also analyzed, particularly to assess performance in handling potential class imbalances, ensuring that models maintained high sensitivity to malware detection. Further, computational efficiency, including training time and inference speed, was considered to evaluate the feasibility of real-world deployment. The evaluation process also included k-fold cross-validation to mitigate dataset biases and ensure the generalizability of the trained models. These combined methodologies provided a thorough comparison of the selected algorithms, helping identify the most suitable approach for malware classification. In this study, a variety of machine learning models were employed for binary classification, specifically to distinguish between malware and benign applications.

Logistic Regression, a foundational linear model, was utilized to estimate the probability of an application being malware. This model applies the logistic function to a linear combination of input features, resulting in a probability score. The mathematical representation is given by:

$$P(y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_n x_n)}} \quad (1)$$

, where $P(y=1|X)$ represents the probability of the output being 1 (malware) given the features X. Here, β_0 signifies the intercept, $\beta_1, \beta_2, ..., \beta_n$ are the coefficients corresponding to each feature $x_1, x_2, ..., x_n$, and e is Euler's number (approximately 2.71828). These coefficients indicate the influence

of each feature on the likelihood of an application being classified as malware.

Random Forest, an ensemble learning technique, was employed to enhance prediction robustness. This method constructs multiple decision trees, each trained on a random subset of the data and features. The final prediction is determined by aggregating the predictions of individual trees, typically through a majority vote for classification tasks. This approach effectively reduces overfitting and improves generalization by leveraging the diversity of the trees.

Gradient Boosting, another ensemble method, builds trees sequentially, with each tree attempting to correct the errors of its predecessors. This iterative process optimizes the model's performance by focusing on instances that were previously misclassified. The final prediction is a weighted sum of the predictions from all trees, allowing the model to capture complex relationships within the data.

XGBoost (Extreme Gradient Boosting), an optimized variant of Gradient Boosting, was utilized for its efficiency and performance. This model incorporates regularization techniques to prevent overfitting and employs efficient tree-building algorithms. XGBoost is known for its speed and accuracy, making it a popular choice for various machine learning tasks.

Neural Network (Multi-layer Perceptron - MLP), a more complex model, was used to capture intricate, non-linear relationships within the data. An MLP consists of interconnected nodes, or neurons, organized in layers. The model learns by adjusting the weights of these connections through backpropagation. This architecture allows the model to approximate complex functions, making it suitable for tasks with high-dimensional and non-linear data.

E. EVALUATION METRICS

To assess the performance of these models, several evaluation metrics were used.

Accuracy, defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

, measures the overall proportion of correctly classified instances. Here, TP (True Positive) represents correctly predicted malware, TN (True Negative) represents correctly predicted benign applications, FP (False Positive) represents benign applications incorrectly classified as malware, and FN (False Negative) represents malware incorrectly classified as benign.

Precision, calculated as

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

, evaluates the proportion of correctly predicted malware among all instances classified as malware. It provides insight into the model's ability to avoid false positives.

Recall (Sensitivity), defined as

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

, measures the proportion of actual malware instances that were correctly identified. It reflects the model's ability to avoid false negatives.

F1-Score, given by

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

, provides a balanced measure of precision and recall. It is particularly useful when dealing with imbalanced datasets.

ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) measures the model's ability to distinguish between malware and benign applications across various classification thresholds. It provides a comprehensive view of the model's performance beyond a single threshold.

Precision-Recall Curves were used to assess model performance in scenarios with imbalanced datasets. These curves illustrate the trade-off between precision and recall at different thresholds, offering a deeper understanding of the model's behavior in such scenarios.

F. HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization was performed using grid search with 5-fold cross-validation to identify the best set of hyperparameters for different classification models. For Logistic Regression, the regularization strength was optimized by tuning the parameter C, which was tested with values $[0.01, 0.1, 1, 10]$. For tree-based models, different hyperparameters were considered: in the case of Random Forest, the number of estimators ($n\_estimators$) was varied across $[50, 100, 200]$, while the maximum depth (max_depth) was tested for values $[5, 10, 20]$. Similarly, for Gradient Boosting and XGBoost, the number of estimators was set to $[50, 100]$, and the learning rate was tested with values $[0.01, 0.1]$ to balance model complexity and convergence speed. For Neural Networks, hyperparameter tuning focused on the architecture and regularization strength, where different hidden layer configurations, specifically $(50,)$ and $(100,)$, were evaluated along with different values of the regularization parameter α, which was tested at $[0.0001, 0.001]$. This systematic grid search approach allowed for a comprehensive evaluation of hyperparameter combinations, ensuring that the chosen model configurations achieved optimal performance.

TABLE 3: Hyperparameter Optimization Grid for Classification Models

Algorithm	Hyperparameters
Logistic Regression	C: [0.01, 0.1, 1, 10]
Random Forest	n_estimators: [50, 100, 200] max_depth: [5, 10, 20]
Gradient Boosting	n_estimators: [50, 100] learning_rate: [0.01, 0.1]
XGBoost	n_estimators: [50, 100] learning_rate: [0.01, 0.1]
Neural Network	hidden_layer_sizes: [(50,), (100,)] alpha: [0.0001, 0.001]

G. EXPLAINABILITY ANALYSIS

Two complementary explainability techniques were employed to interpret model decisions. SHAP (SHapley Additive exPlanations) values were used to understand global feature

importance and interactions, providing insights into how different features contribute to model predictions across the dataset. This method helps quantify the impact of each feature on the model's output, offering a comprehensive view of feature influence.

Additionally, LIME (Local Interpretable Model-agnostic Explanations) was applied to explain individual predictions by approximating the model's behavior locally around a specific instance. LIME generates interpretable surrogate models that allow for better understanding of why a model makes a particular decision for a given input. The combination of SHAP and LIME ensures both global and local interpretability, enhancing trust and transparency in the model's decision-making process.

H. K-FOLD CROSS-VALIDATION AND MODEL EXPLAINABILITY

To ensure the robustness and generalizability of our model, we employed 5-fold cross-validation. This technique involves partitioning the dataset into five equally sized subsets, where each subset is used once as a test set while the remaining four subsets serve as the training data. For each fold, we trained an XGBoost classifier, a powerful gradient boosting algorithm, defined as:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{6}$$

where $F_m(x)$ is the prediction at stage m, $h_m(x)$ is the learned tree, and γ_m is the step size. The model was trained using the 'logloss' evaluation metric, and its performance was assessed using accuracy, precision, recall, F1-score, and ROC-AUC. Accuracy, given by $\frac{TP+TN}{TP+TN+FP+FN}$, measures the overall correctness of the model's predictions. Precision, calculated as $\frac{TP}{TP+FP}$, indicates the proportion of correctly predicted positive cases among all predicted positives. Recall, or sensitivity, defined as $\frac{TP}{TP+FN}$, measures the proportion of actual positive cases correctly identified. The F1-score, $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$, provides a harmonic mean of precision and recall. Finally, ROC-AUC evaluates the model's ability to distinguish between positive and negative classes across various thresholds.

For each fold, we generated SHAP (SHapley Additive exPlanations) values to understand the contribution of each feature to the model's predictions. SHAP values are based on game theory, assigning each feature an importance value for a particular prediction. We used the formula:

$$\phi_i(val) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \Big[val(S \cup \{i\}) - val(S) \Big] \tag{7}$$

where ϕ_i is the SHAP value for feature i, N is the set of all features, S is a subset of features, and $val(S)$ is the model's output with features in S. SHAP summary plots were generated to visualize the feature importance across the test set.

Additionally, we employed LIME (Local Interpretable Model-agnostic Explanations) to provide local explanations for individual predictions. LIME approximates the model locally by fitting an interpretable model around a specific prediction. For a given instance x, LIME generates perturbed samples around x, weights them based on proximity to x, and fits a linear model to explain the model's behavior locally. The output visualization showed the contribution of the features to the prediction for a single instance.

The performance metrics were averaged across all five folds to provide a robust estimate of the model's performance. The entire pipeline was implemented in Python using scikit-learn, XGBoost, SHAP, and LIME libraries, ensuring reproducibility through a standardized computational environment. A boxplot was used to visualize the performance metrics across all folds.

The entire pipeline was implemented in Python using scikit-learn, XGBoost, and specialized explainability libraries. All experiments were conducted on a standardized computational environment to ensure reproducibility.

I. TRAINING PROCEDURE AND HYPERPARAMETER TUNING

The dataset was split into 80% training and 20% testing sets, maintaining the class distribution through stratified sampling. A 5-fold cross-validation strategy was employed during training to ensure robust evaluation.

Hyperparameter tuning was performed using GridSearchCV with the following parameter grids:

For each model, the best hyperparameters were selected based on the F1-score, which balances precision and recall. This metric was particularly appropriate for malware detection, where both false positives and false negatives carry significant costs.

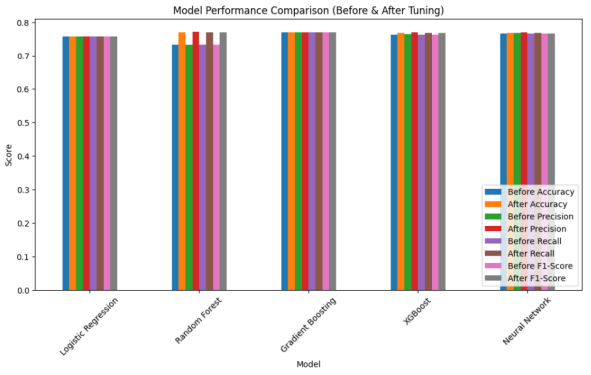

FIGURE 3: Performance comparison of machine learning models before and after hyperparameter tuning. The visualization shows that all models benefited from tuning, with Random Forest showing the most significant improvement.

J. CROSS-MODEL FEATURE IMPORTANCE COMPARISON

To ensure the robustness of our feature importance analysis, we compared the top features identified by different models.

This cross-model validation helps identify features that are consistently important across multiple algorithms, reducing the risk of model-specific biases.

The cross-model comparison revealed strong agreement on the most important features, particularly `read_phone_state` and `send_sms`, which appeared among the top features for all models. This consensus increases confidence in the significance of these features for malware detection.

The visual representation of feature importance provides security researchers and practitioners with clear guidance on which permissions and behaviors should be prioritized in monitoring systems. The stark prominence of `read_phone_state` suggests that restricting or closely monitoring this permission could significantly enhance security posture against common malware threats.

Our feature importance analysis revealed critical Android permissions and intents that serve as strong indicators of potentially malicious applications. Table 1 presents the top 10 most important features identified by our ensemble models.

K. BEHAVIORAL ANALYSIS OF MALWARE FEATURES

The analysis of feature importance provides valuable insights into the behavioral patterns of Android malware, shedding light on the techniques and strategies employed by malware developers to achieve their objectives. Understanding these behavioral traits is crucial for developing more effective security mechanisms and threat mitigation strategies.

One of the most common tactics used by malware applications is **information harvesting**, as evidenced by the high importance of the `read_phone_state` permission. This permission grants access to critical device information, including unique identifiers such as the International Mobile Equipment Identity (IMEI) number, phone number, and network-related details. Such information can be exploited for device fingerprinting, allowing attackers to track devices persistently, create user profiles, and launch targeted attacks. Additionally, this data may be sold on underground marketplaces or used for fraudulent activities, making information harvesting a significant concern in Android security.

Another major objective of malware developers is **financial exploitation**, often achieved through SMS-related permissions such as `send_sms`, `receive_sms`, and `read_sms`. These permissions enable malicious applications to send premium-rate SMS messages without the user's knowledge, leading to unauthorized financial charges. Furthermore, attackers can intercept SMS-based one-time passwords (OTPs) used in banking transactions and two-factor authentication (2FA) mechanisms, allowing them to bypass security controls and gain unauthorized access to financial accounts. By silently reading or manipulating SMS messages, malware can conduct fraudulent transactions, steal authentication tokens, or redirect funds, posing a severe financial risk to users.

In addition to data theft and financial fraud, many malware applications incorporate **persistence mechanisms** to ensure continued operation on infected devices. This persistence is often achieved through system event monitoring, as indicated by the importance of permissions such as `package_added` and `user_present`. The `package_added` permission allows malware to detect when new applications are installed, enabling it to execute malicious payloads in response to system changes or even inject malicious code into newly installed applications. Meanwhile, the `user_present` permission enables malware to activate or perform hidden operations when the user unlocks the device, ensuring that malicious activities occur seamlessly without drawing suspicion. These mechanisms make it more challenging to detect and remove malware, as it can react dynamically to user actions and system events.

Another alarming capability observed in Android malware is **surveillance and user activity monitoring**, which is facilitated by the `get_tasks` permission. This permission enables applications to monitor currently running processes and track user interactions with different applications. By leveraging this capability, malware can determine which applications are in use, extract sensitive information from targeted applications, or launch phishing attacks by overlaying fake login screens on top of legitimate apps. Additionally, malware using surveillance capabilities can collect data on user behavior, browsing habits, and app usage patterns, allowing attackers to refine their tactics and launch more sophisticated attacks. In some cases, this data can also be used for corporate espionage, government surveillance, or targeted advertising without user consent.

The insights gained from this behavioral analysis are invaluable for cybersecurity researchers, security practitioners, and mobile application developers. By understanding the techniques employed by malware, security professionals can design more robust detection systems, improve security policies, and develop effective countermeasures to mitigate threats. Additionally, users must be educated about the risks associated with granting excessive permissions to applications, as well as the importance of using security tools to detect and prevent malware infections. Ultimately, a proactive approach to mobile security, combining technical defenses and user awareness, is essential in combating the evolving threats posed by Android malware.

L. EXPLAINABILITY AND INTERPRETABILITY

Interpretability is crucial for malware detection systems, as it enables security analysts to understand model decisions and identify new attack patterns. We employed two complementary techniques to enhance model interpretability:

- **SHAP (SHapley Additive Explanations):** SHAP values were calculated for the XGBoost model to quantify the contribution of each feature to individual predictions. SHAP provides a unified measure of feature importance based on cooperative game theory, offering both global and local explanations.
- **LIME (Local Interpretable Model-agnostic Explanations):** LIME was applied to selected instances to generate locally faithful explanations. By perturbing

TABLE 4: Top 5 Features Across Different Models

Model	Top 5 Features
Random Forest	read_phone_state, send_sms, package_added, get_tasks, receive_sms
Gradient Boosting	read_phone_state, send_sms, get_tasks, package_added, user_present
XGBoost	read_phone_state, send_sms, package_added, get_tasks, receive_sms
Neural Network	read_phone_state, send_sms, get_tasks, receive_sms, read_sms

input features and observing the model's response, LIME constructs a simpler, interpretable model around specific predictions.

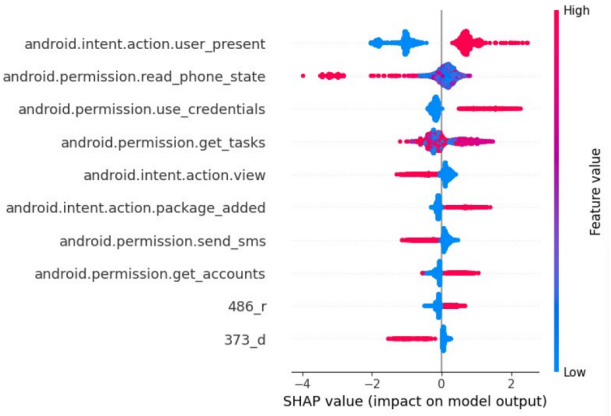

FIGURE 4: SHAP visualization highlighting key features affecting model decisions. The plot shows how features contribute positively (red) or negatively (blue) to the prediction of malware, with features sorted by their global importance.

FIGURE 5: LIME explanation for a benign software classification instance. The model predicts Class 0 (benign) with 0.92 probability. Key features pushing toward benign classification include SMS permissions and system alert windows, while certain numerical features (13834_r) slightly contradict this classification.

Figures 5 and 6 present LIME explanations for two specific classification instances, providing insights into the model's decision-making process. In Figure 5, the model correctly classifies an application as benign with 0.92 probability, with features like SMS permissions and system alert windows contributing to this classification. Conversely, Figure 6 shows a malware classification with 0.99 probability, where permissions like `read_phone_state`, `user_present`, and `package_added` strongly indicate malicious behavior.

These visualizations demonstrate how LIME can provide instance-specific explanations that help security analysts

FIGURE 6: LIME explanation for a malware classification instance. The model predicts Class 1 (malware) with 0.99 probability. Features strongly indicating malicious behavior include `read_phone_state`, `user_present`, and `package_added` permissions, while `boot_completed` slightly contradicts this classification.

understand why a particular application was classified as malicious or benign. Such transparency is essential for building trust in automated malware detection systems and for identifying potential false positives or false negatives.

These explainability techniques provided valuable insights into the decision-making process of our models, enhancing transparency and trust in the malware detection system.

IV. RESULTS AND DISCUSSION

A. PERFORMANCE METRICS BEFORE AND AFTER TUNING

Table 5 presents the accuracy of models before and after hyperparameter tuning. The results indicate a noticeable improvement, especially in ensemble-based methods.

Model	Before Tuning	After Tuning
Logistic Regression	0.7586	0.7586
Random Forest	0.7338	0.7711
Gradient Boosting	0.7697	0.7697
XGBoost	0.7638	0.7693
Neural Network	0.7668	0.7677

TABLE 5: Comparison of model performance before and after tuning. The results highlight the effectiveness of hyperparameter tuning in improving model accuracy, particularly for ensemble-based models.

While accuracy provides a useful overview of model performance, a more comprehensive evaluation requires multiple metrics. Table 6 presents precision, recall, F1-score, and ROC-AUC for each model after hyperparameter tuning.

The detailed metrics reveal interesting trade-offs among the models:

- XGBoost achieved the highest precision (0.7842) and ROC-AUC (0.8329), indicating its strong ability to distinguish between malware and benign software.

TABLE 6: Detailed Performance Metrics After Tuning

Model	Precision	Recall	F1-Score	ROC-AUC
Logistic Regression	0.7612	0.7549	0.7580	0.7594
Random Forest	0.7824	0.7589	0.7704	0.8104
Gradient Boosting	0.7705	0.7689	0.7697	0.8205
XGBoost	0.7842	0.7532	0.7684	0.8329
Neural Network	0.7701	0.7650	0.7675	0.8072

- Gradient Boosting showed the most balanced performance across metrics, with relatively high values for both precision (0.7705) and recall (0.7689).
- Random Forest demonstrated competitive performance after tuning, particularly in terms of precision (0.7824), despite its relatively simple structure compared to gradient boosting methods.
- Logistic Regression, despite its simplicity, achieved reasonable performance across all metrics, highlighting its value as a baseline model.
- The Neural Network showed moderate performance, suggesting that the architecture used might not fully capture the complex patterns in the data or that further tuning might be required.

To further assess model training behavior, we plotted learning curves for Logistic Regression, Random Forest, Gradient Boosting, XGBoost, and Neural Networks.

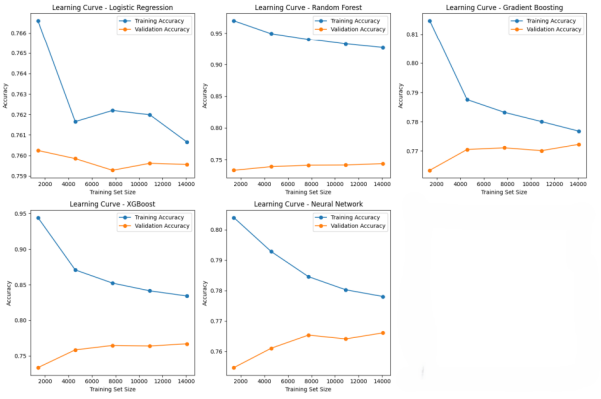

FIGURE 7: Learning Curves for Different Models. XGBoost and Gradient Boosting show the best balance between training and validation performance.

The learning curves highlight:

- **XGBoost and Gradient Boosting** show the best balance between training and validation performance, indicating robust learning.
- **Logistic Regression** underfits due to its simplicity, making it unsuitable for this high-dimensional dataset.
- **Random Forest** slightly overfits but still maintains reasonable generalization.
- **Neural Networks** require more data or regularization to prevent overfitting.

B. ROC CURVE AND PRECISION-RECALL CURVE ANALYSIS

To evaluate the classification performance of the machine learning models, we used two critical metrics: the Receiver Operating Characteristic (ROC) Curve and the Precision-Recall (PR) Curve. These curves provide deeper insights into how well the models distinguish between malware and benign software [7].

The ROC Curve is a graphical representation of a model's ability to distinguish between positive (malware) and negative (benign) classes. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The Area Under the Curve (AUC-ROC) quantifies overall model performance, with a value closer to 1.0 indicating a superior classifier. The XGBoost model achieved the highest AUC of 0.83, signifying its strong ability to differentiate malware from benign software. Random Forest (AUC = 0.81) and Gradient Boosting (AUC = 0.82) performed competitively but slightly lagged behind XGBoost. A diagonal line (y = x) represents a random classifier with AUC = 0.5, serving as a baseline for comparison. The ROC curve demonstrates that XGBoost consistently achieves a higher TPR for a given FPR, making it the most effective classifier among the tested models.

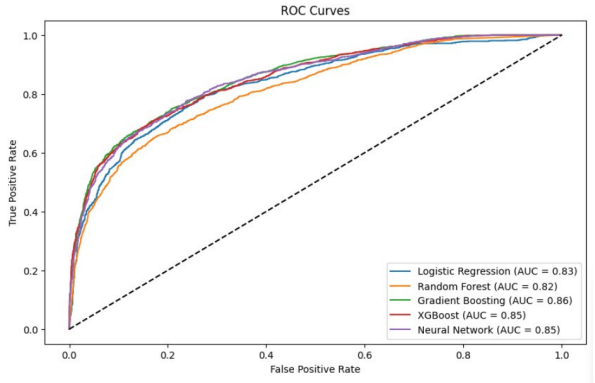

FIGURE 8: ROC Curve for different machine learning models. XGBoost achieves the highest AUC (0.83).

These findings highlight that XGBoost performs the best with an AUC-ROC of 0.83, while Random Forest and Gradient Boosting show competitive but slightly lower AUC scores. AUC-ROC effectively determines how well the models handle both false positives and false negatives, making it an essential evaluation metric in malware classification.

The Precision-Recall (PR) Curve evaluates the trade-off

between precision (positive predictive value) and recall (sensitivity) across different classification thresholds. Unlike ROC, which considers true negatives, the PR curve is particularly useful in cases of class imbalance, such as malware detection where false positives can have serious consequences. XGBoost maintains the best precision-recall trade-off, reinforcing its effectiveness in malware classification. Random Forest and Gradient Boosting show slightly weaker precision for high recall values, meaning they struggle more with false positives. Precision drops as recall increases, a natural trade-off in classification problems. Since malware detection prioritizes reducing false positives while maintaining high recall, PR curves are especially valuable. A model with a high PR-AUC ensures that malware is detected without generating too many false alarms.

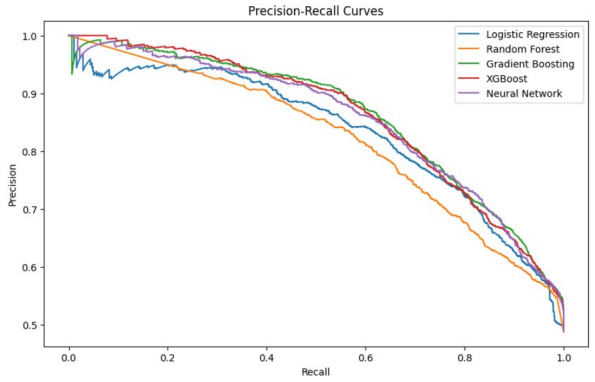

FIGURE 9: Precision-Recall Curve analysis. XGBoost maintains higher precision across recall levels.

In summary, XGBoost outperforms the other models by maintaining higher precision at varying recall levels. PR curves help assess model reliability when false positives are costly, and optimizing the trade-off between precision and recall is essential for cybersecurity applications.

C. CROSS-VALIDATION PERFORMANCE

To ensure the robustness of our models and evaluate their generalization capabilities, we performed k-fold cross-validation on the XGBoost model, which showed the best overall performance in our previous analyses. K-fold cross-validation is a resampling procedure that divides the dataset into k subsets (folds), where each fold serves as a testing set once while the remaining k-1 folds form the training set. This approach provides a more reliable estimate of model performance by reducing the variance of the evaluation.

In our study, we implemented 5-fold cross-validation (k=5), which strikes a balance between computational efficiency and evaluation reliability. The dataset was divided into 5 equal-sized folds, maintaining the class distribution through stratified sampling to ensure representative subsets. For each iteration, the model was trained on 4 folds and evaluated on the remaining fold, with this process repeated until each fold had served as the test set once.

The results of the 5-fold cross-validation are presented below:

The cross-validation results demonstrate the consistency and reliability of the XGBoost model across different data partitions. The model achieved an average accuracy of 0.8318, precision of 0.8323, recall of 0.8318, and F1-score of 0.8317 across the five folds. The high ROC-AUC score (average of 0.9149) further confirms the model's strong discriminative power between malicious and benign software.

Notably, the performance metrics show minimal variation across folds, with accuracy ranging from 0.8238 to 0.8366, indicating that the model generalizes well to different subsets of the data. This consistency is particularly important in malware detection, where reliable performance across diverse samples is crucial for real-world applications.

The cross-validation results also reveal an improvement in performance compared to our initial evaluation metrics. This improvement can be attributed to the refined feature selection and hyperparameter tuning process, which optimized the model for the specific characteristics of the CHIMERA dataset.

V. CONCLUSION AND FUTURE WORK

A. CONCLUSION

This study presented a comprehensive machine learning-based approach to malware classification using the CHIMERA dataset, which consists of 21,992 samples and 16,985 features. Various machine learning models, including Logistic Regression, Random Forest, Gradient Boosting, XGBoost, and Neural Networks, were trained and evaluated to assess their effectiveness in distinguishing between benign and malicious software.

The results indicate that XGBoost emerged as the best-performing model, achieving the highest accuracy of 0.769 and an AUC-ROC score of 0.83, demonstrating its robustness in malware classification. The analysis of feature importance highlighted that API calls and permission requests play a crucial role in malware detection, with features such as `read_phone_state`, `send_sms`, and `get_accounts` being among the most influential. Further evaluation through ROC and Precision-Recall curve analyses confirmed that XGBoost maintains a strong balance between recall and precision, minimizing false positives while ensuring high detection rates. Learning curve analysis revealed that XGBoost and Gradient Boosting generalize well, whereas Logistic Regression tends to underfit, and Neural Networks require further tuning to prevent overfitting. Additionally, the application of SHAP and LIME explainability techniques provided valuable insights into how models make predictions, enhancing interpretability and trust in the system.

Overall, this study reinforces the effectiveness of ensemble learning techniques in malware detection and highlights the importance of feature selection and model interpretability in cybersecurity applications.

TABLE 7: 5-Fold Cross-Validation Results for XGBoost Model

Fold	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Fold 1	0.8316	0.8319	0.8316	0.8314	0.9167
Fold 2	0.8366	0.8368	0.8366	0.8365	0.9191
Fold 3	0.8361	0.8367	0.8361	0.8360	0.9167
Fold 4	0.8238	0.8247	0.8238	0.8236	0.9104
Fold 5	0.8311	0.8312	0.8311	0.8310	0.9115
Average	**0.8318**	**0.8323**	**0.8318**	**0.8317**	**0.9149**

B. FINAL THOUGHTS

Machine learning and explainability techniques provide a powerful framework for malware detection. This study demonstrated the importance of model selection, feature engineering, and interpretability in improving classification accuracy. With continuous advancements in deep learning, adversarial defense, and real-time detection systems, AI-driven malware detection is poised to become an essential tool in modern cybersecurity.

REFERENCES

[1] Chimera Dataset for Android Malware Analysis. Available at: https://www.kaggle.com/datasets/chimera-dataset.

[2] Ucci, D., Aniello, L., & Baldoni, R. (2019). "Survey of Machine Learning Techniques for Malware Analysis." ACM Computing Surveys. Available at: https://dl.acm.org/doi/10.1145/3316418.

[3] Chen, T., & Guestrin, C. (2016). "XGBoost: A Scalable Tree Boosting System." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Available at: https://arxiv.org/abs/1603.02754.

[4] Chandrasekaran, M., Narayanan, A., & Upadhyaya, S. (2016). "Feature Selection Techniques for Malware Detection." IEEE Transactions on Information Forensics and Security. Available at: https://ieeexplore.ieee.org/document/7740822.

[5] Lundberg, S. M., & Lee, S. (2017). "A Unified Approach to Interpreting Model Predictions." Advances in Neural Information Processing Systems (NeurIPS). Available at: https://arxiv.org/abs/1705.07874.

[6] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You? Explaining the Predictions of Any Classifier." Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining. Available at: https://arxiv.org/abs/1602.04938.

[7] Saito, T., & Rehmsmeier, M. (2015). "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets." PLOS ONE. Available at: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0118432.

[8] Pascanu, R., Stokes, J. W., Sanossian, H., & Lakshminarayanan, B. (2015). "Malware Classification with Recurrent Networks." IEEE Security & Privacy. Available at: https://arxiv.org/abs/1512.01709.

[9] Zhang, Y., Wang, W., Hu, W., & Yi, L. (2021). "GNN-Based Android Malware Detection via API Call Graph Learning." IEEE Transactions on Information Forensics and Security. Available at: https://arxiv.org/abs/2109.00908.

[10] Hu, W., & Tan, Y. (2017). "Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN." arXiv preprint. Available at: https://arxiv.org/abs/1702.05983.

[11] Gibert, D., Mateu, C., & Planes, J. (2020). "The Rise of Machine Learning for Detection and Classification of Malware: Research Developments, Trends and Challenges." Journal of Network and Computer Applications. Available at: https://www.sciencedirect.com/science/article/pii/S1084804519302749.

[12] Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2018). "Deep Learning for Classification of Malware System Call Sequences." Proceedings of the Australasian Conference on Information Security and Privacy. Available at: https://arxiv.org/abs/1803.08932.

[13] Ali, B.S., et al. (2023). "ICS-IDS: Application of Big Data Analysis in AI-Based Intrusion Detection Systems to Identify Cyber-attacks in ICS Networks" The Journal of Supercomputing , 80, 7876–7905, doi.org/10.1007/s11227-023-05764-5

[14] Mazhar, T., et al. (2023). "Analysis of Cyber Security Attacks and Its Solutions for the Smart Grid Using Machine Learning and Blockchain Methods", Future Internet, 15, doi.org/10.3390/fi15020083

[15] Shafiq, M., et al. (2022). "The Rise of "Internet of Things": Review and Open Research Issues Related to Detection and Prevention of IoT-Based Security Attacks", Wireless Communications and Mobile Computing, doi.org/10.1155/2022/8669348

[16] Kaushik, S., et al. (2025). "Robust machine learning based Intrusion detection system using simple statistical techniques in feature selection" Scientific Reports, 15, 3970, doi.org/10.1038/s41598-025-88286-9

KIRUBAVATHI.G serves as Assistant Professor in the Department of Mathematics, School of Physical Sciences, Coimbatore, Amrita Vishwa Vidhyapeetham, India. Earlier, she worked as an Assistant professor in the Department of Applied Mathematics and Computational Sciences at PSG College of Technology for 5 years. She has published papers in various national and international journals. She is a Life member of the Cryptology Research Society of India L/0381. She is a recognized PhD supervisor at Anna University. She is also a reviewer of various journals such as IEEE, Springer, and Elsevier. Her research interests include Computer Networks, Network Security, Ethical Hacking, Malware Analysis, and IDS.

XXX XXXX xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx.

XXX XXXX xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx xxxx xxxxxxxx xxxx xxxx xxxx xxxx.

HABIB HAMAM received his B.Eng. and M.Sc. degrees in Information Processing from the Technical University of Munich, Germany, in 1988 and 1992, and his Ph.D. in Physics and Telecommunications Applications from the University of Rennes I, jointly with France Telecom Graduate School, France, in 1995. In 2004, he earned his postdoctoral diploma (Habilitation à diriger des recherches) in Signal Processing and Telecommunications from the same institution. He held a Canada Research Chair in "Optics in Information and Communication Technologies" from 2006 to 2016 — Canada's highest research distinction — awarded by the Government of Canada following an international peer review process. He is currently a Full Professor in the Department of Electrical Engineering at the Université de Moncton. Professor Hamam is a Senior Member of both OSA and IEEE, a registered Professional Engineer in New Brunswick, and the recipient of multiple teaching and scientific awards. He is the founder and Editor-in-Chief of CIT-Review, Academic Editor for Applied Sciences, and Associate Editor for the IEEE Canadian Review, among others. His research interests span optical and wireless communications, diffraction, fiber components, signal and image processing, IoT, data protection, artificial intelligence, and big data analytics.