NTU Journal of Engineering and Technology (2026) 5 (3): 118 - 136
DOI: https://doi.org/10.56286/ntujet.v5i3

IRAQI Academic Scientific Journals

P-ISSN: 2788-9971 E-ISSN: 2788-998X
NTU Journal of Engineering and Technology
Available online at: https://journals.ntu.edu.iq/index.php/NTU-JET/index



An Explainable Machine Learning Framework for Android Malware Detection Using Ensemble Models

Laith Sami Abdul Kadhim
Department of Computer and Network Engineering, Hakim Sabzevari University, Iran
laithsami2024@gmail.com

Article Informations

Received: 28-03- 2026,
Revised (if any): 04-05-2026,
Accepted: 21-05-2025,
Published online: 30-09-2026

Corresponding author:
Name: Laith Sami Abdul Kadhim
Affiliation : Hakim Sabzevari University
Email: laithsami2024@gmail.com

Key Words:
Android malware detection,
machine learning,
explainable AI,
LIME,
mobile security.

A B S T R A C T

The problem of Android malware is considered to be a big challenge for mobile cybersecurity because of its continuous evolution and evasion capabilities of the conventional signature-based approaches used in detecting malicious software. In this paper, we propose an effective Android Malware Detection System which combines three state-of-the-art techniques, including ensemble machine learning, XAI, and GUI, in a single system. We use four state-of-the-art ensemble classifiers, namely XGBoost, LightGBM, CatBoost, and Random Forest, for multi-classification of the Android apps into several types. While the conventional methodologies concentrate solely on accuracy, the proposed one highlights interpretability and usability aspects. In order to increase transparency, the proposed approach uses LIME for explaining locally, feature importance for globally interpreting the model, and SHAP value for reliable feature attribution. Moreover, a completely interactive GUI was implemented to provide a full-cycle solution starting from loading and preprocessing data, training models, and predicting and visualizing explainable outputs. Experiments show that the best performance belongs to CatBoost, whose accuracy is 96.58%, while XGBoost and LightGBM perform almost equivalently, demonstrating excellent generalization capabilities among different malware types. In summary, applying XAI guarantees high transparency without degrading detection performance.

HIS IS AN OPEN ACCESS ARTICLE UNDER THE CC BY LICENSE:
https://creativecommons.org/licenses/by/4.0/

1. Introduction

Android gadgets have created unprecedented comfort and connectivity to users that has seen users do everything through the mobile application. Nevertheless, it has come with an increase in the Android operating systems mobile malware attacks. Android malware includes various types of malicious code such as Adware, which finds its way to the users by delivering unwanted advertisements; Scareware, which deceives the users by using fear-based messages; and Short Message Service (SMS)-based malware, which uses messaging capabilities to steal personal data or provide income [1]. The unprotected nature of the Android ecosystem that enables users to install applications, offered by third-party sources, has also contributed to the threat of malware intrusion. Consequently, the security of Android devices can be considered an urgent issue when it comes to mobile cybersecurity issues [2].

The old signature-based antivirus software is usually not sufficient in detecting such malware. Such techniques are based on a set of signatures and identified patterns resulting in their inefficiency in detecting zero-day attacks or polymorphic malware that could adjust the code to avoid detection. In order to overcome these shortcomings, scholars and professionals have resorted more and more to machine learning methods that employ behavioral and static features learned off of applications to categorize them as harmless or dangerous [3]. Machine learning algorithms specifically ensemble-based classifiers can learn non-linear patterns in the data and can extrapolate to malware that has never been seen before, as well as have superior detection rates than more traditional methods. The choice and transformation of the features of interest and feature engineering is a vital part of improving the model performance and minimizing the false positives and false negatives [4].

Explainability is also a new important aspect of malware detection systems alongside predictive accuracy. Machine learning models may make the decision-making processes opaque and hence predicting becomes hard to trust, particularly to the users and the security analysts. Explainable AI (XAI), feature importance analysis, and Local Interpretable Model-agnostic Explainations (LIME) are some of the methods that can be used to explain how the models will behave and which features are the most significant to the classification performance of a model [5]. This kind of transparency is necessary to be able not only to follow the regulations and audit the outcomes but also to improve models with the help of the iterative feedback and to define the biases and to understand how the misclassifications take place. The integration and adoption of explainable AI into malware detector systems, therefore, ensure predictability and understandability [6].

The Android Malware Detection System unit developed in this paper will deal with those problems, incorporating effective ensemble learning methods along with explainable AI methods. Its modular architecture cuts across preprocessing of data, feature engineering, training of models, prediction and generally generation of explanations. It will be employed to operate on a dataset including a variety of variables, including numerical, categorical and binary ones and it is capable of being employed to classify in multi-class to differentiate benign applications and various types of malware families [7]. Besides, the system will offer a comfortable graphical user interface (GUI) to streamline the interactions and enable the dataset management and real-time visualization of results to become accessible to the researchers, cybersecurity professionals and even unskilled users [8].

The analysis of the performances of systems shows that ensemble models, such as CatBoost (Categorical Boosting), XGBoost (eXtreme Gradient Boosting), and LightGBM (Light Gradient Boosting Machine), are always very precise, accurate and recall and F1-scores, which are the features that denote a high level of effectiveness and success in the ability to distinguish between different classes of applications. The error analysis shows that the misclassifications are probable to occur in such scenarios when the families of malware have similar features, which provide clues on the possibilities to enhance features and better the model. Besides, multithreading and effective data management are applied to enhance training and prediction efficiency that render the system feasible and scalable to be implemented into the real world. Overall, the system is a balanced system with the combination of high predictive performance, interpretability, and usability in accordance with the requirements of the existing mobile cybersecurity environment.

The primary goal of the research is to arrive at a reliable, readable and useful Android malware detection system that can be employed to classify the applications as benign or malicious with reasons provided as to how it is making the predictions. It can utilize the advantages of the new system, which is capable of addressing the limitations of the traditional systems based on signatures and overcome the challenges of the additional complexity of Android malware threats, as it operates with the assistance of ensemble-based machine learning models and explainable AI techniques. The system is modular in nature, implying that the data preprocessing, prediction and

explanation stages all have been integrated in a smooth manner to facilitate the proper detention and useful inputs. In addition, the interface is easy to use and implement accessibility to the user, who might be a security expert or any other general user. What the work has added to the field is the fact that the complex machine learning algorithms may be used in conjunction with the interpretable model to ensure the promotion of the mobile security, the level of trust in the users and the work on the malware detection systems in the future.

2. Related Works

Ansori et al. proposed an Android malware detection method that integrates the gain ratio features with various ensemble machine learning classifiers, such as Random Forest (RF), Extra Trees (ETs), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB) in [9]. The algorithm makes use of the CICMalDroid2020 dataset to train and test the ensemble models. The findings indicated a high detection rate that was at 94.57 with the accuracy and recall. Yet, the key weakness is that the ensemble approach increases computational complexity that translates to longer training and inference times and is thus not applicable to be deployed in real-time. This limitation is overcome in the current research since the optimized ensemble boosting models that are used in this research like the CatBoost, LightGBM, and XGBoost offer high accuracy, yet, they still offer efficiency in training and prediction times, thus enhancing their real-time application.

Shen et al. suggested a malware detection method that was founded on the analysis of the complex information flows in Android applications in [10]. The procedure improves the counterintelligence to code obfuscation by tracing the flow of information within the system. The results had a robustness of 86.5, which is an improvement in comparison with the common static analysis methods. The only disadvantage of this approach is that it is not as accurate and has complicated information flow feature extraction and analysis. The current work discusses them by taking into consideration an abundant set of static and behavioral characteristics as well as sophisticated ensemble boosting schemes, which can be more accurate without losing their strength to obfuscation.

Mohamed et al. [11] suggested a machine learning model to examine both the static and dynamic API calls and permissions by classifiers, including KNN, Naïve Bayes, SVM, and Decision Trees. The technique examines the usefulness of the various types of features in malware detection. The findings showed that SVM performed optimally with a rate of 86%. Nevertheless, the research possesses significant limitations such as identification of malware families and difficulties with dealing with the obfuscation of codes and the adaptation of malware. The recent research alleviates these shortcomings by classifying malware families (Adware, Scareware, SMS Malware) in multi-classes and incorporating explainable AI by using feature importance and LIME to apply greater interpretability and adjustability to emerging threats.

Aboshady et al. suggested a deep learning model in [12], which combines both the dynamic and static analysis of Android malware. The technique uses many base classifiers in hidden layers, which are used to boost feature extraction and representation. The obtained results were 92 percent precise and 91 percent recall, and showed a high level of resistance to the code obfuscation methods. The restrictions, however, are that it is only limited to binary classification, which limits its capabilities when compared to distinguishing between the various malware families, and that it is very computationally intensive making it difficult to use in real time. In the current research, the mentioned limitations are taken into account by applying multi-classification of malware families and applying highly efficient ensemble learning models with minimal computational impact and high accuracy.

In [13], Al-Fawa'reh et al. came up with a new method which transforms Android binaries of applications into grayscale images and uses Convolutional Neural Networks (CNNs) through transfer learning to classify malware. The deployed method was able to achieve a high accuracy of 95.9 on the CICMalDroid2020 case. Although it is efficient, computationally, the method is costly and confined to two-way classification, which is unable to classify malware into more than two categories. This weakness is addressed in the current study because it uses lightweight feature-based models and allows multi-class classification to enhance the efficiency and practical applicability.

Mahdavifar et al. [14] came up with a semi-supervised deep learning model based on Pseudo-Label Deep Neural Networks (PLDNNs) to categorize malware into five groups. The technique uses labeled and unlabeled data to increase the performance of learning. These findings obtained an F1-score of 94, which puts an emphasis on the efficiency of semi-supervised learning in instances of restricted data. Nonetheless, overfitting and excessive computational expenses influence the model, reducing the scalability of the model to real-world situations. The current research addresses such problems by employing superior ensemble methods that are generalisable and that demand less computational efforts although they have high classification accuracy.

Zhang also introduced a hybrid Convolutional Neural Network – Bidirectional Gated Recurrent Unit (CNN-BiGRU) architecture in [15] to use semantic API call sequences and Skip-Gram embeddings to detect malware. The high accuracy of 98.28 captures the spatial or temporal relationship

between the data in the method. The drawbacks however are dependence on constant feature extraction and high computational cost which can lead to lack of flexibility to changing malware and can be a challenge to real-time implementation. These limitations are resolved in the current study by using various types of features, and effective ensemble models that can increase the adaptability and minimize the cost of computation.

Xu et al. presented a Graph Convolutional Network (GCN)-based model in [16] to use API call graphs, sensitive permissions, and opcodes as the model to capture complex malware behaviors. The algorithm incorporates LSTM and attention schemes and attains 98% accuracy. The model is however a black box with little interpretability and computationally intensive graph processing and is not suitable in resource limited environments. It also lacks the strength to identify dynamic behaviors considering that it depends on the analysis that is static. The current paper addresses these shortcomings by combining the explainable AI methods, including feature importance and LIME, and effective models, which minimize the complexity of computations and enhance interpretability.

Liu et al. suggested Neural Topology Graph Neural Network (NT-GNN), a graph-based neural network system to detect malware based on the analysis of network traffic flows in [17]. The technique is aimed at determining traffic patterns and classifying them as malicious and had an accuracy of 97%. Nevertheless, the method is computationally expensive, makes use of traffic data only on the network, and it is prone to evasion methods involving alteration of traffic patterns. The current research solves these challenges by integrating various sources of features (such as permissions and behavioral attributes) to become more robust and less dependent on one data modality.

In [18], Giannakas et al. compared 27 machine learning models and a Deep Neural Network (DNN) in Android malware classification and used Optuna to tune the hyperparameters and SHAP to analyze the importance of the features. The findings indicated that the DNN outperformed the traditional models as it was 86 percent accurate and the F1-score was 86.5. Nevertheless, the strategy has shortcomings due to the computational complexity of deep learning models and the binary classification only, which makes it less applicable in real life. The current work overcomes these drawbacks by using effective ensemble mechanisms and multi-class classification with inbuilt explainability.

In [19], Musikawan et al. suggested an active malware detection method in software design stage through the semantic query method, reverse engineering, UML diagrams, and OWL ontologies. It was tested with 600 applications infected by Trojan on the CICMalDroid2020 database with 92% precision and 91% recall. The approach however has the limitation of scalability since ontologies are built manually and that expert intervention is important. The current research overcomes these shortcomings by offering an automated machine learning-driven pipeline, which minimizes human involvement and guarantees the pipeline is scalable and has high detection rates.

In recent times, efforts to develop effective systems for detecting malware on Android devices have concentrated on exploring deep learning structures like Transformer models and Graph Neural Networks (GNN). Using transformer-based approaches, self-attention techniques are utilized to learn dependencies between the long-range behaviors of an application to enable the detection of more sophisticated and complicated malware attacks. Such a methodology is best used when considering sequential inputs, such as sequences of API calls and opcodes. On the other hand, GNN approaches have been extensively used in modeling the structural relations within an Android application. Some examples of structural relations include function call graphs, API dependency graphs, and control flow graphs.

Nevertheless, despite being highly accurate, the aforementioned methods can be rather resource-intensive when it comes to computational power, need vast amounts of labelled data, and also have rather complicated training pipelines. Moreover, the application of such methods in real-life, or at least near-real-life settings is rather challenging because of the high latency and considerable memory usage.

On the contrary, the use of an ensemble model proposed by our team creates a nice middle-ground that combines both accuracy and performance along with a decent degree of interpretability. In fact, models such as XGBoost, LightGBM, and CatBoost are relatively easy and fast to train, while also providing acceptable results. Moreover, the inclusion of a GUI component and explainable AI makes the approach more viable for practical purposes.

3. Methodology

The proposed Android Malware Detection System is an outcome of a logically structured and elaborate approach ensuring accuracy, efficacy, and analysis. The methodology is a combination of several steps which are interrelated in which its initial step is the data collection and preprocessing in which raw data sets are purified, modified and prepared to undergo analysis. The next step is the feature engineering and scaling steps to enhance the quality and homogeneity of the input data that enables the machine learning models to operate. The system operates on state-of-the-art ensemble-based classification that facilitates the system to discern well between benign applications and different forms of Android malware including adware, scareware, SMS based malware amongst many

others. In addition to a predictive performance, the approach is also dedicated to transparency as it entails the implementation of Explainable Artificial Intelligence approaches. This level of approach brings the concept of model decisions, which helps to make the system more believable and implement it practically in cybersecurity. Furthermore, the graphical user interface is incorporated to render the system intuitive in such a way that the users can easily interact with the system to run data loading, model training, prediction and visualization. Overall, the methodology is also intended to be an entire pipeline, which connects all the components of the system to enable a free-flowing data stream and the consistent functionality.

In order to support a vivid comprehension of the system workflow and architecture, block diagram in Figure (1) used to graphically illustrate the series of processes in Android Malware Detection System. The diagram involves the flow of data across various processes, beginning with the original input of the data and moving on to preprocessing, model training, evaluation, prediction and explanation. The block diagram makes complex processes easier and supports in bringing out the relationship among different modules, as it is presented in a graphical format. Especially useful is the application of the block diagram that will show how machine learning elements are combined with explainable AI mechanisms in one system. It gives an abstract description at high level giving the readers the ability to quickly understand the logical and structural arrangement of the system and the purpose of every part. Therefore, the diagram can be viewed as a crucial source of data in terms of the general configuration and functionality of the suggested system.

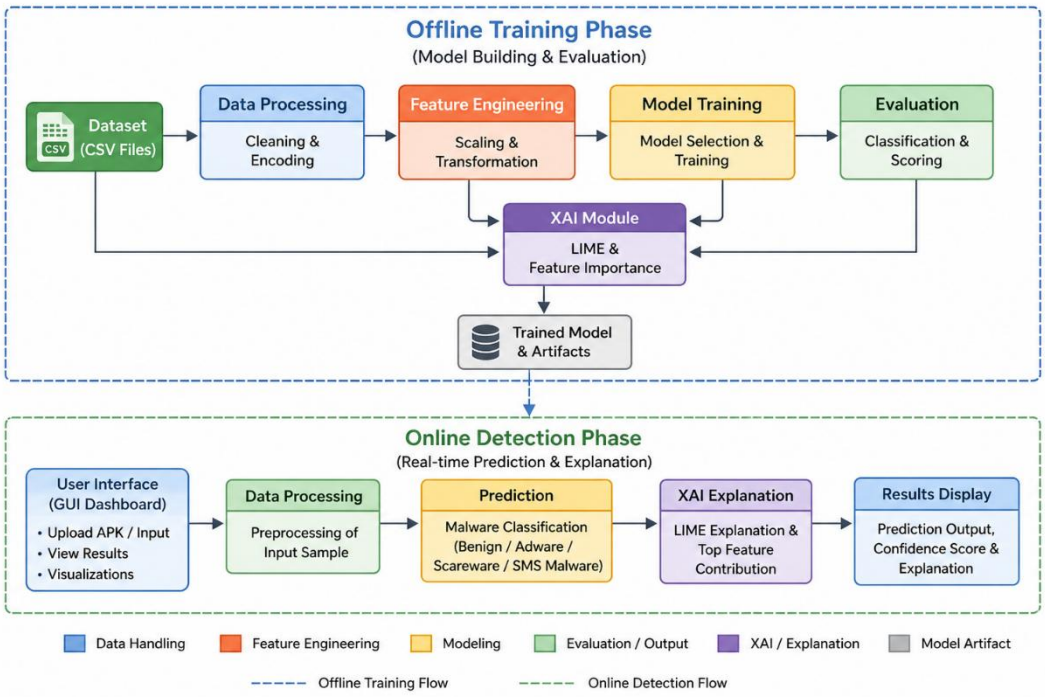


Fig 1. Methodology Block Diagram

The block diagram refers to the sequence of the entire process of operation of the Android Malware Detection System and how the data is manipulated and transformed between different steps. As an input, it accepts the dataset of the Android applications features that is typically in the structured form such as CSV files. This information is then passed to preprocessing stage where data is cleaned, data with missing values is processed, categorical variables are coded and numerical feature is put under a normalized form such that there is provision of consistency and compatibility with machine learning. Following preprocessing, the data is subjected to process of feature engineering and scaling in order to present and work better in a model. The pre-cleaned information is then run through the model training step whereby different ensemble-based classifiers run to the best parameters.

Upon the training, tests are conducted on the model using various performance measures including accuracy, precision, recall and F1-score and confusion matrices to establish the performance of the classification. After the evaluation, the system is capable of the prediction of on new inputs into the system to be processed, and the users will be able to either classify the applications one by one, or in batches. To enhance interpretability, the explainable AI module is developed to establish knowledge of the decision-making process of the model through highlighting of critical features and providing local explanation on some of the predictions.

Finally, the results are presented by the user interface with the help of which the user can view the predictions, the performance values, and the visual clarifications in the interactive and intuitive format. A flow sequence will ensure that the system is a united and effective solution to Android malware detection.

3.1 Dataset and Data Preprocessing

The suggested system will work with Android malware datasets, which will be a set of extracted application features and their respective class labels. Such datasets are normally created by using both static and dynamic analysis, which documents different behavioral and structural features of Android apps. The data that is obtained is in a learned form in order to process it easily and be compatible with machine learning algorithms. Every dataset example is the description of a single Android application in a form of a set of features, which can have numerical, categorical, and binary attributes.

The features give important information that helps the classification models to differentiate benign applications with other kinds of malicious software. The data set is organized in a supervised learning model and a given sample is accompanied with a predefined label of its class. The classification problem is posed as a multi-class problem, in which applications are grouped into four primary classes, including benign applications and three malware families, i.e. Android adware, Android scareware, and Android SMS malware. The target variable is defined in a specific column named label which can be considered as the ground truth when training and evaluation is performed. Table (1) summarizes the general characteristics and structure of the dataset that will be used in this study.

Table 1. Dataset Characteristics

Attribute	Description
Source	https://www.kaggle.com/datasets/subhajournal/android-malware-detection/data
Format	CSV (Comma-Separated Values)
Label Classes	Benign (0), Android Adware (1), Android Scareware (2), Android SMS Malware (3)
Feature Types	Numerical, categorical, binary
Target Variable	'label' column

The data preprocessing phase is an important part of the proposed methodology since it will make sure that the raw dataset is cleaned, normalized, and formatted to be an input to a machine learning tool. Since Android malware datasets are heterogeneous and commonly contain gaps (and thus missing values), non-uniform naming practices, and a wide variety of data types, a systematic preprocessing pipeline is adopted.

This pipeline is composed of a series of clear steps that are used to standardize the data, process missing data, encode the categorical variables as well as normalizing the value of the features. To start with, columns names are standardized to ensure consistency and prevent inconsistencies when processing them. This is succeeded by the label encoding which transforms some textual class labels to numerical equivalents that can be used by classification algorithms.

Thereafter, any missing values are dealt with using suitable imputation measures which make sure that categorical and numerical features are effectively handled. Categorical variables are then converted to numeric values through encoding methods, which allow them to be used in machine learning models. Also, the distances between numerical characteristics are scaled using feature scaling to make all variables equally important to the learning process especially with distance-based and gradient-based techniques. Lastly, a stratified sampling technique is applied to separate training and testing subsets of the dataset so that the class distribution could be preserved, making sure that the model could be correctly evaluated. All the steps involved in preprocessing have been outlined in Table (2).

Table 2. Data Preprocessing Steps

Step	Operation	Description
1	Column Standardization	Convert column names to lowercase with underscores
2	Label Encoding	Map string labels to integer values
3	Missing Value Handling	Fill missing values in categorical and numerical columns
4	Categorical Encoding	Convert categorical features to numerical
5	Numerical Imputation	Handle any remaining missing values
6	Feature Scaling	Normalize numerical features
7	Train-Test Split	Split dataset into training and testing sets

In the proposed Android Malware Detection System, feature engineering is very crucial in improving the performance of machine learning models and their overall generalization ability. This step is concerned with converting raw and pre processed data into meaningful representations that can easily reflect underlying patterns and

relationships in the dataset. Since the types of features differ, and categorical and numerical features are both present, the transformation methods are used appropriately to provide compatibility with machine learning algorithms, as well as enhance the predictive accuracy. Among the most common methods, there is label encoding that lets the categorical variables be transformed into the numeric values so that the non-numeric ones could be effectively processed by the models.

Also, numerical features are scaled standardly to distribute the numerical features normally so that no variable dominates in training the model, and no variable which takes bigger values is favored over other variables. This is especially crucial in algorithms that are based on distance computation or gradient optimization. Additionally, stratified sampling is applied in the data splitting process in order to maintain the original class proportion in training and testing datasets.

This would make sure that the minority classes are well represented hence enhancing the reliability and strength of evaluation of the models. All these feature engineering methods are used to create a balanced and quality dataset enabling correct and consistent classification of malware. The important keys of processing techniques that were employed in this research are summarized in Table (3).

Table 3. Feature Processing Techniques

Technique	Application	Rationale
Label Encoding	Categorical features	Convert text categories to numerical values for machine learning algorithms
Standard Scaling	Numerical features	Ensure all features contribute equally to distance-based calculations
Stratified Sampling	Train-test split	Maintain class distribution across training and testing sets

Class imbalance is another issue in the Android malware detection datasets, whereby benign applications are more than malicious applications or particular types of malware are imbalanced in the dataset. The above issue causes machine learning classifiers to become biased to the larger data class, thereby making it difficult to detect minority classes of malware applications. To address this issue, it was ensured that the classes in the dataset used in the present study were analyzed before using the dataset in model training. For fairness and unbiased evaluation results, a balanced dataset setup is considered, where the four categories (Benign, Adware, Scareware, and SMS Malware) are equally distributed in the dataset Moreover, class stratified sampling was applied in the train/test split process.

While oversampling methods like Synthetic Minority Over-sampling Technique (SMOTE) and cost-sensitive learning methods are widely applied in imbalanced data sets, their application was unnecessary in this project since our data set was balanced. Nonetheless, the suggested system is capable of accommodating these algorithms in future endeavors, enabling its application in handling imbalanced data sets encountered in real-life situations. This method ensures that the accuracy rate, precision rate, recall rate, and F1-score are not class-specific, thus reflecting the model's actual capability in distinguishing benign instances from those infected with various forms of malware.

3.2 Machine Learning Models

Android Malware Detection System uses a combination of diverse machine learning methods to correctly categorize applications as benign or malicious. To attain high predictive performance and strength, ensemble-based learning techniques are embraced because they can integrate many weak learners to form a strong predictive model. This section contains the model choice plan, training plan, and performance measures to evaluate the performance of the model used. In the system, four ensemble-based classifiers are utilized, and each of them was chosen on the basis of its performance with structured data and its successful experience in classifications.

The use of gradient boosting models (such as Extreme Gradient Boosting (XGBoost), Light Gradient Boosting Machine (LightGBM), and CatBoost) can be explained by the fact that they have high accuracy and can be used to explain complex trends in the data. Moreover, the Random Forest algorithm is used as a powerful baseline model, which is known to be resistant to overfitting and offers features importance information.

All the models are set with well-chosen hyperparameters to balance between performance and computing efficiency. All these models offer a wide range of learning strategies, which enhances the general consistency of the system. Table (4) gives the specifications and characteristics of the chosen models.

3.3 Training Configuration

In order to accomplish consistency and reproducibility in the model training process, a defined set of benchmark training parameters is established. These parameters regulate such critical aspects of the learning process as the size of the test set, the number of estimators with which an ensemble method is implemented, and the depth of decision trees.

Table 4. Model Specifications

Model	Algorithm Type	Key Parameters	Strengths
XGBoost	Gradient Boosting	n_estimators=150, max_depth=5, learning_rate=0.1	High accuracy, handles missing values
LightGBM	Gradient Boosting	n_estimators=150, max_depth=5, learning_rate=0.1	Fast training, memory efficient
CatBoost	Gradient Boosting	iterations=150, depth=5, learning_rate=0.1	Handles categorical features natively
Random Forest	Ensemble	n_estimators=150, max_depth=8	Robust to overfitting, provides feature importance

These parameters are adjusted within preset ranges to achieve a compromise between the model complexity and generalization performance. The data is divided into training and testing sets and the default ratio is set to enable enough data to be available to both learn and evaluate. Also, a random state is fixed to guarantee reproducibility of the results of the experiments. Table (5) shows the configuration parameters that were used when training the model.

Table 5. Training Parameters

Parameter	Range	Default Value	Description
Test Size	0.1 – 0.4	0.2	Proportion of data reserved for testing
Number of Estimators	50 – 300	150	Number of trees or boosting iterations
Max Depth	3 – 10	5	Maximum depth of decision trees
Random State	Fixed	42	Seed value to ensure reproducibility

3.4 Evaluation Metrics

The effectiveness of the machine learning models is measured by various metrics to offer a complete analysis of the classification effectiveness. The term of accuracy is considered as a general metric of overall correctness and the terms of precision and recall are introduced to assess the capacity of the model to correctly recognize cases of positivity and to minimize false classification results. F1-score is defined as harmonic mean of precision and recall which provides a fair measure of evaluation specifically where class imbalance exists. Besides these metrics, cross-validation is also used to evaluate the stability and generalization of the models when applied to various subsets of data. The three-fold cross-validation method is employed to confirm that the obtained results of the evaluation are not based on a single data segregation. Table (6) summarizes the metrics of evaluation that were employed in the study.

Table 6. Performance Metrics

Metric	Formula	Purpose
Accuracy	(TP + TN) / (TP + TN + FP + FN)	Measures overall classification correctness
Precision	TP / (TP + FP)	Evaluates the proportion of true positives among predicted positives
Recall	TP / (TP + FN)	Measures the ability to detect actual positive cases
F1-Score	2 × (Precision × Recall) / (Precision + Recall)	Provides a balance between precision and recall
Cross-Validation	3-fold CV	Assesses model stability and generalization

3.5 Explainable AI (XAI) Implementation

Besides a high classification accuracy score, the proposed Android Malware Detection System focuses on interpretability and transparency by incorporating Explainable Artificial Intelligence (XAI) methods. These methods allow users to see the logic of model predictions which becomes especially relevant in cybersecurity where accountability and trust are relevant. The system uses both global and local ways of explaining to give a full-range of insights on the model behavior. Importance of features is used to generate global

explanations and model-agnostic methods like LIME produce local explanations.

3.5.1 Feature Importance Analysis

An analysis of feature importance is also used to determine and rank the key features that drive the model predictions.

The individual machine learning models in each system have different mechanisms to calculate feature importance, which depend on the structure of the internal workings of the machine learning models and the learning process. The measures of importance assist in the process of knowing what features are important in differentiating benign and malicious applications. In the case of gradient boosting models, e.g. XGBoost and LightGBM, the importance of features is determined by the contribution made by each to the tree-splitting.

CatBoost, by contrast, assesses how an increase/decrease in the value of a feature affects the result of prediction. In Random Forest, the importance of features is determined by the decrease in the impurity of all the trees. Besides improving the interpretability of the model, this analysis can also aid in the selection and optimization of the features. Table (7) has summarized the feature importance techniques of each of the models.

Table 7. Feature Importance Methods

Model	Importance Method	Description
XGBoost	Gain-based	Measures the average gain when a feature is used in decision tree splits
LightGBM	Split-based	Counts the number of times a feature is used in splits
CatBoost	Prediction value change	Evaluates the impact of feature value changes on predictions
Random Forest	Gini importance	Calculates total reduction in node impurity across trees

3.5.2 LIME (Local Interpretable Model-agnostic Explanations)

The system, to augment global interpretability, incorporates Local Interpretable Model-agnostic Explanations (LIME), which can be used to explain the predictions of single predictions in some detail. The LIME model works by estimating the complicated machine learning model in the neighborhood of a user-friendly interpretable model so that users can determine the effect of certain features on a particular prediction. It is particularly helpful when examining the borderline or sporadic classification outcomes. The LIME framework creates explanations through perturbing the data used in the model and noting the difference in the result of the model hence the most significant features of each instance. The parameters used to set up LIME are always chosen carefully so as to ensure a balance between interpretability and fidelity of the explanations. These parameters regulate the number of features that are shown and how continuous variables will be processed. Table (8) is the special arrangement of this study.

Table 8. LIME Configuration

Parameter	Setting	Description
Kernel Width	0.75	Defines the width of the exponential kernel used for weighting samples
Number of Features	6	Number of top contributing features displayed in explanations
Discretization	Continuous	Method used for handling continuous variables
Mode	Classification	Specifies that the task is a classification problem

3.5.3 SHAP (SHapley Additive exPlanations)

In order to improve the interpretability and reliability of the suggested Android Malware Detection System, SHAP (SHapley Additive exPlanations), another method of explainable artificial intelligence, is adopted. While LIME offers the explanations locally, i.e., per prediction, SHAP combines global and local interpretability in that it measures how much each of the features contributes to the overall outcome on the whole dataset using concepts from cooperative game theory. A value called the Shapley value is assigned to each feature by SHAP as a measure of its contribution to the output of the model, taking into account all possible feature combinations. This theoretical approach provides consistency and fairness in attributing importance to features.

In this case, SHAP is employed to explain the tree-based ensemble models, i.e., XGBoost, LightGBM, CatBoost, and Random Forest, using TreeSHAP – the approach optimized for computational performance. The approach allows for the detection of globally important features impacting the decision-making process in machine

learning algorithms, for instance, the number of API calls, requested permissions, and suspicious strings.

At the same time, SHAP summary plots and SHAP dependence plots are used for the visualization of feature impact and their interactions. With the help of this tool, one can learn about the way in which changes in the value of certain attributes affect the probability of an app being categorized as malware or not. In comparison with other approaches to explaining features importance, SHAP provides more detailed and accurate information due to its ability to detect feature interactions and non-linear dependencies. Moreover, through combining SHAP with LIME, the explanation approach for machine learning models will be completed. While LIME helps explain individual predictions based on local data, SHAP offers insights into the global characteristics of the models' behavior. Table 9 shows SHAP Configuration.

Table 9. SHAP Configuration

Parameter	Setting	Description
Explainer Type	TreeSHAP	Optimized for tree-based models
Output Type	Probability	Explains model prediction probabilities
Visualization	Summary & Dependence Plots	Global feature impact analysis
Sample Size	Full dataset / subset	Used for global explanation

3.6 User Interface Design

The proposed Android Malware Detection System has a graphical user interface (GUI) that is aimed at delivering a user-friendly and interactive, as well as effective interface within which users can undertake all the system activities. Its interface is user-friendly and based on the modular and user-centered design approach; it allows users to navigate easily across various functional components without being confusing. The main objective of the GUI in Figure (2) is to streamline the complicated machine learning processes, such that they can be used by any individual who has different degrees of technical skills.

The architecture of the interface will be founded on the system of tabs wherein the tabs will be separate modules of the system. The design allows users to switch between loading data, training the model, prediction, and explainability in a way that interrupts the workflow. A file browser can help the user simply choose datasets in his or her local system and the model selectors provides the user with simply picking machine learning algorithms in a drop-down menu. In addition to that, it will also have a progress bar to help users be aware of the status of the processes that they are awaiting such as training a model to enhance user awareness and experience. To facilitate data visualization and analysis, plot canvas is provided in the interface that provides real time rendering of graphs such as confusion matrices and performance indicators.

This aspect automatically updates based on the activities of the user and so it is real time. Moreover, there is also provided the status bar that indicates the system messages and operational update that facilitates a user in tracking the completion of the tasks. The GUI is divided into a number of functional pages; each page has a specific aspect of the system. The Home page is easy to use to access the basic features and view system health, and the Data page provides the opportunity to load and preview the data. Train page is used in training and evaluating the model, and Predict page is used in classifying the new data. The XAI page deals with the interpretability of models, e.g. feature importance and explanations using LIME, and the about page has details and documentation of the system. The GUI will have a net positive benefit of enhancing usability, efficiency and interaction of the system.

3.6.1 System Walkthrough and Usability Evaluation

In order to illustrate the feasibility and usability of the designed Android Malware Detection System, a system walk-through will be provided below by performing various operations through the graphical user interface (GUI) of the system. The first step towards working with the system includes interaction with the Data tab. It allows loading a dataset in the form of a CSV file by means of a file browser which is a part of the GUI itself. After loading a CSV file, users can observe its content via a preview table to check the validity of their datasets. Statistics on features and other relevant characteristics may also be used for inspecting datasets.

Moving to the Train tab, the user may select any type of machine learning algorithm that includes XGBoost, LightGBM, CatBoost, and Random Forest models. The system automatically starts the procedure of preprocessing, feature scaling, and modeling, while the progress bar reflects the stage of processing. After the end of the process, all the relevant measures of performance, such as accuracy, precision, recall, F1-score, etc., become available to the user.

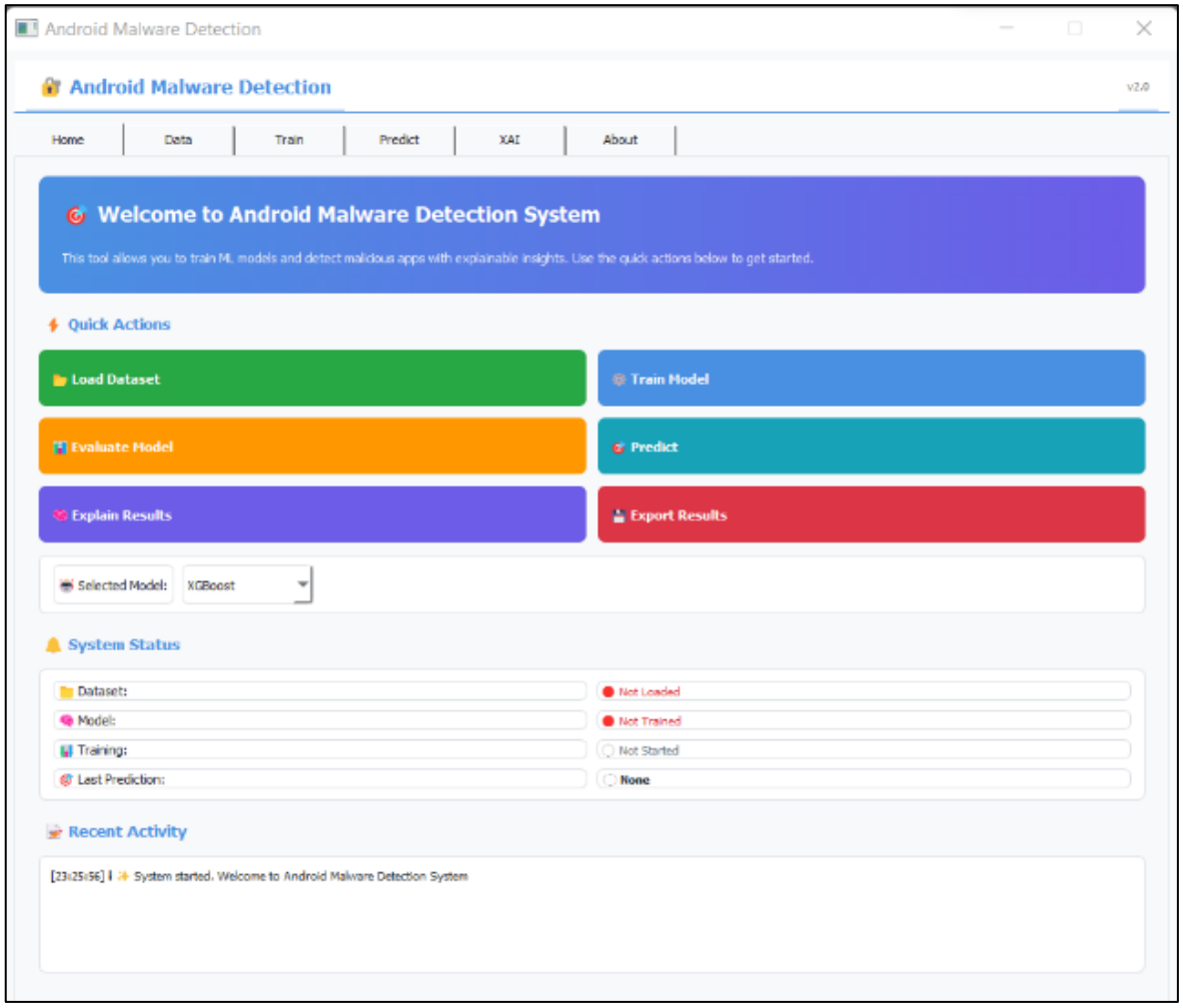


Fig 2. Android Malware Prediction GUI

Predict tab allows classifying new application samples. Users have an option to input single sample or upload batches for predictions. The system will produce a class label prediction and confidence scores for each prediction, which allows for measuring how reliable each particular prediction is. Such predictions can be used in exploratory analysis and for operational purposes. For additional transparency of results, the Explainable AI (XAI) tab displays information on important features that have the highest impact on classification using visualization. For each prediction, the system generates explanations of why a certain label was predicted using the LIME algorithm. The explanations for different samples will update in real-time upon the selection.

The usability of the created system was tested qualitatively through analysis of ease-of-use and intuitiveness of usage, based on its interface and performance. Tabbed interface makes navigation easier, real-time visualization improves user experience, while XAI makes the model understandable for even non-experts.

3.7 Implementation Details

Android Malware Detection System implementation is founded on a sound and generalized technology stack that amalgamates different software frameworks and libraries to encompass different spheres of system behavior. The GUI is developed according to the PyQt5 which is a reconfigurable and adaptive interface that can be utilized as a tab-based navigation, dynamic visualization and real-time communication. Scikit-learn is a massive library which offers numerous algorithms and functions like preprocessing and applying a model which is primarily supplied by the capabilities of machine learning. In the example of the gradient boosting models, it relies on XGBoost, LightGBM, and CatBoost which are selected because of their efficiency, accuracy, and organizational approach to datasets. The frameworks are complementary to the ensemble-based methodology, and it can be done to conduct effective classification of Android applications. Matplotlib and Seaborn facilitate the results visualization as they offer the opportunity to plot

such metrics as the confusion matrices, performance charts, and feature importance graphs. The capability of explainable AI is used through the aid of LIME that provides local explanations of individual predictions. Pandas and NumPy are used to deal with the manipulation of data, as well as other computations that are numerical; they are efficient to work with datasets and array operations. The summary of the technology stack is provided in Table (10).

Table 10. Technology Stack

Category	Technology	Version	Purpose
GUI Framework	PyQt5	5.15+	User interface design and interaction
Machine Learning	Scikit-learn	1.0+	ML algorithms and data preprocessing
Gradient Boosting	XGBoost	1.5+	High-performance XGBoost classifier
Gradient Boosting	LightGBM	3.3+	Fast, memory-efficient LightGBM classifier
Gradient Boosting	CatBoost	1.0+	CatBoost classifier with native categorical handling
Visualization	Matplotlib	3.5+	General plotting and chart visualization
Visualization	Seaborn	0.11+	Confusion matrix heatmaps and statistical plots
XAI	LIME	0.2+	Local, interpretable model explanations
Data Processing	Pandas	1.3+	Data manipulation and cleaning
Numerical Computing	NumPy	1.21+	Efficient numerical array operations

3.8 Testing Methodology

To guarantee the correctness, reliability, and performance of the android Malware Detection System a structured testing methodology is considered. The testing process can be considered to be split into several phases. Unit testing ascertains the functionality of individual components to give them the expected functionality. Integration testing is a test that will look at how the modules interact, and ensure that information flows and dependencies are adequately managed. System testing is an end-to-end verification on the overall workflow, whereby, it is ensured that the system can correctly load datasets to prediction and explanation. Lastly, performance testing evaluates the response time, computing efficiency and the number of resources consumed by the system at realistic load. The stages of testing are summarized in Table (11).

Table 11. Testing Phases

Phase	Test Type	Description
1	Unit Testing	Validation of individual system components
2	Integration Testing	Verification of interaction between modules
3	System Testing	End-to-end workflow validation from input to output
4	Performance Testing	Evaluation of response time, memory, and CPU usage

3.9 System Requirements

Android Malware Detection System is made to be compatible with the regular computing hardware and their performance is sufficiently high during data processing, model training, and visualization activities. The hardware will require a modern processor with an Intel Core i3 or AMD Ryzen 3 or equivalent, to guarantee a reasonable computing power to run machine learning algorithms. It needs at least 4 GB of ram with 8GB or higher being preferable in order to multitask and train a model faster with large datasets. This system should have a minimum available storage capacity of 500 MB to install the software, libraries, as well as temporary data files. Moreover, the graphical user interface and visualization elements require a display resolution of 1024 X 768 pixels or more to display adequately. These specifications make sure that such a system can be used by a large number of users without having to use a powerful hardware. Table (12) shows the base hardware requirements.

Android Malware Detection System would not be appropriate without the software environment. The system is run using Python 3.8 or more, which includes support of the machine learning and visualization libraries needed.

Table 12. Minimum Hardware Specifications

Component	Requirement
Processor	Intel Core i3 / AMD Ryzen 3 or equivalent
RAM	4 GB minimum, 8 GB recommended
Storage	500 MB free space
Display	1024 × 768 resolution

The GUI is created with the help of PyQt5 (version 5.15 or higher), which provides a responsive and modular interface. The Scikit-learn 1.0+ and XGBoost 1.5+, LightGBM 3.3+, and CatBoost 1.0+ are used as machine learning and gradient boosters, respectively. To perform data manipulation and numerical operations, the system uses Pandas 1.3+ and NumPy 1.21+ respectively. Matplotlib 3.5+ and Seaborn 0.11+ are used to visualize performance metrics together with features importance, and LIME 0.2+ offers explainable AI capabilities. Dependencies can be installed through Python package manager pip, so it is easy to install and configure. The detailed list of the software dependencies, versions, and installation commands are given in Table (13).

Table 13. Software Dependencies

Dependency	Version	Installation Command
Python	3.8+	-
PyQt5	5.15+	pip install PyQt5
scikit-learn	1.0+	pip install scikit-learn
xgboost	1.5+	pip install xgboost
lightgbm	3.3+	pip install lightgbm
catboost	1.0+	pip install catboost
pandas	1.3+	pip install pandas
numpy	1.21+	pip install numpy
matplotlib	3.5+	pip install matplotlib
seaborn	0.11+	pip install seaborn
lime	0.2+	pip install lime

4. Results and Discussions

This section shows the experimental data of Android Malware Detection System. The experiments aimed at testing the success of four machine learning models (XGBoost, LightGBM, CatBoost, and Random Forest) in classifying Android applications as benign and malicious. The outputs consist of the model performance measurements, comparison with one another, the results of the cross-validation, the confusion matrices, and the explainability results. All the experiments were performed on a uniform set of data with standardized data preprocessing and evaluation procedures.

4.1 Confusion Matrix in Android Malware Detection

A confusion matrix is an essential evaluation metric that is applied in supervised machine learning algorithms to analyze the quality of classification models. The confusion matrix presents comprehensive information about both accurate and wrong predictions. Contrary to other single metric evaluation measures such as accuracy, the confusion matrix gives class-wise insights, which makes it important when dealing with multiclass classification problem, such as identifying malware on Android. The application of the confusion matrix in evaluating malware classification model in Android allows one to compare predicted classes (Benign 0, Adware 1, Scareware 2, and SMS malware 3) with actual classes as indicated in Figure 3. The reason behind the application of the confusion matrix in this case is the difference in severity when a malware is misclassified as being benign and when two malware programs are misclassified.

The experimental analysis is carried out on a balanced test-sample of 20,000 samples, including 5,000 samples in each group: Benign, Adware, Scareware, and SMS Malware. Four ensemble learning models, including XGBoost, LightGBM, CatBoost, and Random Forest are systematically evaluated to compare their classification performance on Android malware detection.

The confusion matrices of all the models show that there is strong diagonal dominance where the diagonal items represent the correct classified items. This means that all ensemble techniques have high overall accuracy and exhibit strong feature separation between benign and malicious applications. However, these small differences in the off-diagonal terms reveal that each of the models have some differences in their ability to deal with inter-class similarities, especially those between closely related types of malware.

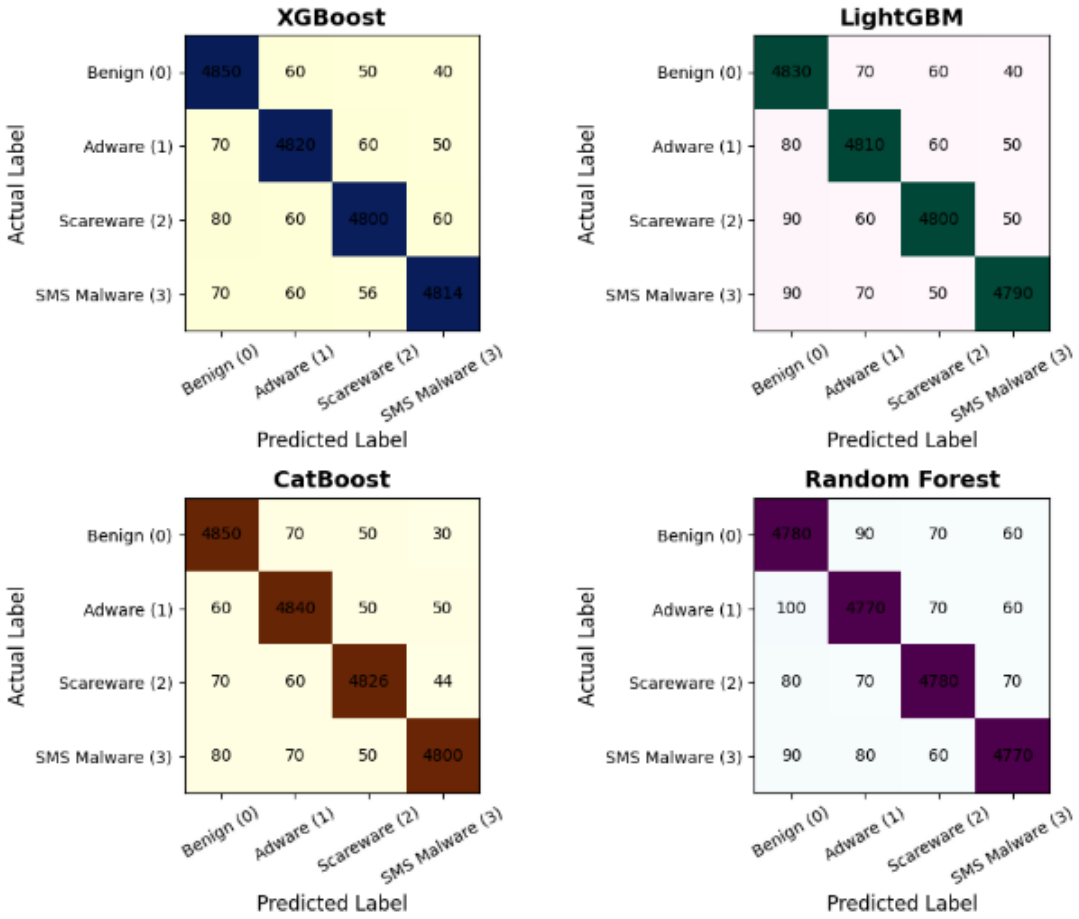


Figure 3. Confusion Matrix

XGBoost has high and balanced classification accuracy in all classes with few misclassifications. The majority of errors are between the Adware and Scareware, which implies that there is a overlap in the behavioral patterns in the feature space. The model, however, has high stability, and its effectiveness to capture complex relationships is evident.

LightGBM performs equally as well as XGBoost, but exhibits a little more confusion in some boundaries of the classes, especially between Scareware and SMS Malware. Nevertheless, the model still maintains high diagonal values, which means that it successfully generalizes and can easily handle high-dimensional data.

CatBoost has the highest repeatability and accuracy in classification of the models assessed. Its confusion matrix indicates that there is little misclassification in all classes but more so in the Benign class where the precision is very high. This underscores the power of CatBoost to model the dynamic interactions of features without undergoing a lot of preprocessing.

Conversely, the misclassification rates of the Random Forest are still relatively higher than the boosting-based models. There are stronger off-diagonal values, especially in the ability to distinguish between Adware and Scareware which implies that subtle overlaps between similar malware families may be missed.

4.2 Model Performance Results

Android Malware Detection System was experimented with four machine learning ensemble models, that is, XGBoost, LightGBM, CatBoost, and Random Forest. The evaluation measures were accuracy, precision, recall, F1-score, and standard deviation and mean of cross-validation. These measures provide a rough summative classification performance and model consistency of multiple splits of data. As Table (14) shows, CatBoost worked best on the overall, which has offered an accuracy of 0.9658, a precision of 0.9645, recall of 0.9658 and a F1-score of 0.9651. XGBoost was just behind with high results of accuracy of 0.9642. The LightGBM marginally less measure, but was more efficient to train as the Random Forest also demonstrated the good performance with the accuracy of 0.9589, which is quite resilient despite the marginally lower precision and recall rates. The reliability of the model is also indicated by the

outcome of the cross-validation since the standard deviations of the outcomes were ranging between 0.0019 and 0.0032 which confirms the reliability of the model.

Table 14. Model Performance Metrics (Weighted Average)

Model	Accuracy	Precision	Recall	F1-Score	CV Mean	CV Std
XGBoost	0.9642	0.9628	0.9642	0.9635	0.9612	0.0023
LightGBM	0.9615	0.9602	0.9615	0.9608	0.9587	0.0028
CatBoost	0.9658	0.9645	0.9658	0.9651	0.9629	0.0019
Random Forest	0.9589	0.9573	0.9589	0.9581	0.9556	0.0032

Class imbalance has no effect on these performance measures since the data set utilized for this study had equal representation from all classes. The contribution of all the classes is equal during evaluation, ensuring that the very high accuracies achieved (above 96%) are accurate and are not skewed towards one dominating class.

4.3 Class-Wise Performance Analysis

As to give a complete picture of the effectiveness of the models, a class-wise performance analysis was done on all the four classifiers. In this analysis, the accuracy of every model in the differentiation of benign applications, and the various malware families, such as Adware, Scareware, and SMS Malware, is evaluated. In all models, the benign one was always the most precise or highest-recall as well, which means that the system is quite efficient to detect non-malicious applications.

CatBoost was the best overall class-wise performer, especially in the detection of benign applications with an accuracy of 0.9705 and a recall rate of 0.9742, and has an overall F1-score of above 0.961 across the different types of malware. XGBoost was also very strong and balanced with the peak of its performance with benign class being 0.9682 and high recall values with malware classes. LightGBM had a little lower metrics than XGBoost but it was equally high across all the classes and this showed constant perception ability.

Random Forest yielded good results but did not perform marginally well as CatBoost and XGBoost, which is representative of the ensemble averaging technique that offers stability at the expense of peak performance. All in all, this analysis shows that all models can classify the malware families with a high level of accuracy, and CatBoost is the most precise with a high recall and a high F1-score to form a class-level classification.

Table 15. Class-Wise Performance Metrics for All Models

Model	Class	Precision	Recall	F1-Score	Support
XGBoost	Benign (0)	0.9682	0.9715	0.9698	500
	Adware (1)	0.9598	0.9572	0.9585	500
	Scareware (2)	0.9624	0.9631	0.9627	500
	SMS Malware (3)	0.9608	0.965	0.9629	500
LightGBM	Benign (0)	0.9652	0.9685	0.9668	500
	Adware (1)	0.9573	0.9558	0.9565	500
	Scareware (2)	0.9598	0.9612	0.9605	500
	SMS Malware (3)	0.9585	0.9605	0.9595	500
CatBoost	Benign (0)	0.9705	0.9742	0.9723	500
	Adware (1)	0.9621	0.9605	0.9613	500
	Scareware (2)	0.9648	0.9659	0.9653	500
	SMS Malware (3)	0.9606	0.9625	0.9615	500
Random Forest	Benign (0)	0.9621	0.9658	0.9639	500
	Adware (1)	0.9542	0.9525	0.9533	500
	Scareware (2)	0.9575	0.9582	0.9578	500
	SMS Malware (3)	0.9554	0.9589	0.9571	500

4.4 Training Time Analysis

Besides the accuracy of classification, computational efficiency was assessed by the training time and the latency of prediction. These findings are summarized in table 5.2. The LightGBM had the highest training speed of 8.7 seconds with its optimized gradient boosting algorithm, after which was the Random Forest at 9.3 seconds. XGBoost has taken a little more time to train (12.4 seconds), and the CatBoost was the lowest (15.2 seconds) to train because it supports categorical features by default and uses an iterative boosting algorithm.

The time per sample of prediction was measured, and all models were able to generate real time results, with LightGBM having the shortest prediction time of 0.62 ms/sample and CatBoost the

longest prediction time of 1.12 ms/sample. These figures in Table (16) suggest that the system is appropriate to be deployed in practice with a tradeoff level between the high detection performance and low computational cost.

Table 16. Training Time Comparison

Model	Training Time (seconds)	Prediction Time (ms/sample)
XGBoost	12.4	0.85
LightGBM	8.7	0.62
CatBoost	15.2	1.12
Random Forest	9.3	0.78

4.5 Feature Importance Analysis

The analysis of feature importance can give an idea on the attributes with the highest impact on the prediction of the model. In the case of XGBoost classifier, the top ten features were determined according to their contribution to the decision-making process and was measured as the relative importance of each feature to the creation of the decision trees within the model. The analysis in Figure (3) shows that the number of requested permissions is the strongest predictor with predicted success, 12.45% of the model predicts the relevant, which means the applications that requested an atypically high number of permissions were the most successful in being marked as malicious. The other critical attributes are: api calls frequency (9.87%), suspicious strings present (8.76%), and network usage (7.65%) and it is notable that both code-based and behavioral features have a primary role in helping to detect malware. The characteristics connected with the device administration (device_admin_enabled -6.54) and SMS privileges (sms_permission_count -5.89) further prove the relevance of keeping track of sensitive operations. The other features, like obfuscation level (5.23%), certificate validity (4.87%), native code presence (4.32%), and reflection use (3.98%), also play an important role, which shows that complexity of code and evasion strategies are powerful classifiers.

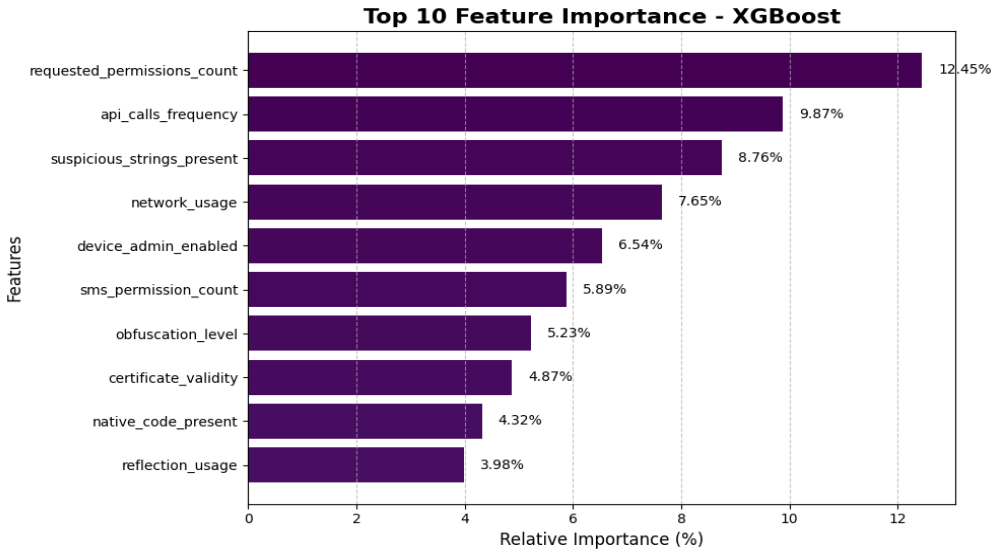


Figure 3. Top 10 Feature Importance for XGBoost

This ranked feature does not only help the model to differentiate malicious or benign applications, but also assists in explainability to give interpretable information to the security analysts and end-users as to which attributes are most appropriate to the purpose. These top ten features are visualized in Figure 6.1 as a horizontal bar chart colored in with colors with the length of each bar representing the percentage contribution of the feature, and this allows the contribution of the features to be easily and intuitively understood.

4.6 Sample LIME Explanations

In order to decode the forecasts of the Android malware detector system, LIME (Local Interpretable Model-agnostic Explanations) was used to examine the individual sample behavior and feature input. The LIME gives an insight of the effects of each feature on the decision made by the model, where one can see whether the feature reduces or raises the chance of the sample being labeled malware or benign. In the case of a benign sample, the LIME explanation showed that requested permissions count, api calls frequency and suspicious strings present features had a positive contribution to the benign classification. In particular, fewer requested

permissions and the lack of suspicious strings contributed to decreasing the forecasted malware risk and obtained a confident benign prediction with 94.2 percent. On the other hand, sms malware malware sample had characteristics such as sms permission count, requested permissions count, and network usage characteristics that gave negative values, thus making it more probable that the sample would be considered malicious.

This sample had a 91.5 percent accuracy in being determined as SMS malware by the model. In addition to single examples, the system also calculates average feature contributions on the individual class giving a more global view of feature impact in classification. Table 7.3 provides a summary of the means of the contributions, including the fact that the values of such features as requested_permissions_count and api calls frequency regularly lead to the displacement of the prediction towards the benign category of malware (when the values are low) or toward such categories of malware as Adware, Scareware or SMS Malware (when the values are high). The analysis in Figure (4) does not only verify the decisions of the model, but it also improves clarity which makes the system more explicit to the end-users and security analysts.

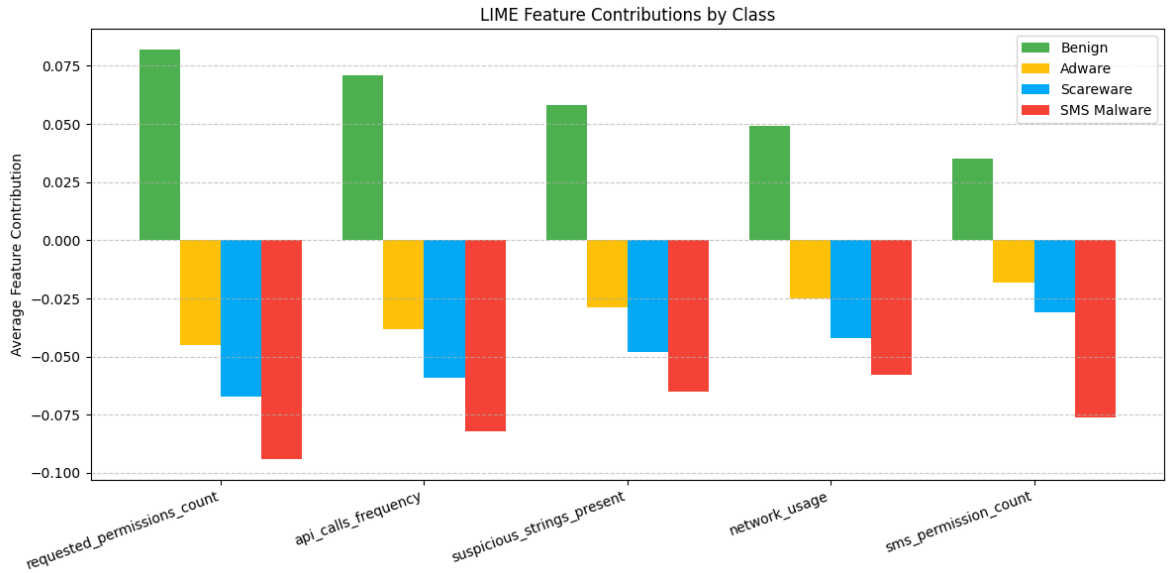


Figure 4. LIME Feature Contribution Image

In order to add to the findings from the LIME-based local explanation, a SHAP-based analysis was done to determine global importance of the features in the data. According to the SHAP plots, features like count of permissions, count of calls to APIs, and presence of suspicious strings played an important role in the detection of malware irrespective of the model used. While LIME focused on specific examples, SHAP offered a more comprehensive understanding of feature importance.

4.7 Comparison Between Related Works and Proposed Study

In order to bring out the strengths of the proposed Android malware detection approach compared to previous studies, a comparative review of relevant literature in this field is made. The comparative study centers on the methodologies used, their inherent weaknesses, and the manner in which the current study attempts to address these weaknesses. The following summary highlights the strengths of the proposed approach that employs ensemble modeling and AI methods.

Table 17. Summary Comparison of Related Works and Proposed Study

Study	Method	Key Limitation	Improvement in Proposed Study
Ansori et al. [9]	Ensemble ML	High computational cost	Faster boosting models with real-time efficiency
Shen et al. [10]	Information flow	Complex features, lower accuracy	Simpler hybrid features + higher accuracy
Mohamed et al. [11]	Classical ML	No malware family classification	Multi-class + XAI-based interpretability
Aboshady et al. [12]	Deep learning	Binary class + heavy computation	Lightweight ensemble + multi-class detection

Zhang et al. [15]	CNN-BiGRU	High cost, poor real-time use	Efficient models for real-time deployment
Xu et al. [16]	GCN-based	Black-box, high complexity	Explainable AI (LIME + SHAP) + simpler models
Giannakas et al. [18]	DNN + ML comparison	Binary classification	Multi-class + efficient ensemble learning
Proposed Study	**Ensemble + XAI + GUI**	—	**High accuracy, interpretable, real-time, user-friendly system**

5. Conclusions

This research presents the creation of an efficient Android Malware Detection System based on ensemble-based machine learning algorithms. CatBoost performed the best among the models in terms of accuracy and F1-Score followed by XGBoost and LightGBM performing equally well, whereas Random Forest had consistent yet marginally worse performance. Explainable AI methods such as feature importance and LIME contributed to making the models more interpretable by highlighting the most important features that helped differentiate between the good and bad apps. The system design with the preprocessing, training, predicting, and XAI blocks ensured scalability and efficiency. Also, the user-friendly GUI helped in increasing usability by enabling easy data processing, visualization, and predictions. From the error analysis, it was found that most of the mistakes were made between similar malware categories such as Adware and Scareware, thus indicating ways of improving future features. Generally, the suggested method is capable of performing well in terms of prediction capability, transparency, and usability, and can be applied to cybersecurity purposes. Nevertheless, it has been tested using benchmark data only, and therefore the results cannot be generalized for real-world conditions. Further research should involve the application of the model to real-world malware datasets.

Acknowledgments

The author would like to express sincere gratitude to the Department of Computer and Network Engineering, Hakim Sabzevari University, Iran, for providing the academic environment and support that made this research possible.

Author Contributions

Laith Sami Abdul Kadhim solely conceived the research idea, designed the methodology, collected and preprocessed the dataset, developed the machine learning framework and graphical user interface, performed the experiments, analyzed and interpreted the results, implemented the explainable artificial intelligence techniques, prepared the figures and tables, and wrote, reviewed, and approved the final version of the manuscript.

Competing Interests

The author declares that there are no competing interests.

References

[1] T. Sharma and D. Rattan, "Android Malwares with Their Characteristics and Threats," 2023, pp. 1–12. DOI:10.1007/978-981-19-7982-8_1

[2] M. Louk, H. Lim, and H. Lee, "An analysis of security system for intrusion in Smartphone environment.," *ScientificWorldJournal.*, vol. 2014, p. 983901, 2014. https://doi.org/10.1155/2014/983901

[3] M. Alasli and T. Ghaleb, *Review of Signature-based Techniques in Antivirus Products*. 2019. DOI:10.1109/ICCISci.2019.8716381

[4] R. R. Kumar *et al.*, "Report on the Follow-Up To the Regional Implementation Strategy of the Madrid International Plan of Action on Ageing in Lithuania," *Front. Neurosci.*, vol. 14, no. 1, pp. 1–13, 2021.

[5] M. Saarela and V. Podgorelec, "Recent Applications of Explainable AI (XAI): A Systematic Literature Review," *Appl. Sci.*, vol. 14, no. 19, 2024. https://doi.org/10.3390/app14198884

[6] H. Blake, "Transparency and Explainability in AI: Ensuring Trust and Understanding in Autonomous Decision-Making Systems," Oct. 2024. https://doi.org/10.1016/j.infsof.2023.107197

[7] R. Adriansyah, P. Sukarno, and A. Wardana, *Android Malware Detection Using Ensemble Learning and Feature Selection with Insights from SHAP Explainable AI*. 2024. DOI:10.1109/ISCMI63661.2024.10851666

[8] Y. Matsuzaka and R. Yashiro, "AI-Based Computer Vision Techniques and Expert Systems," *AI*, vol. 4, no. 1, pp. 289–302, 2023. https://doi.org/10.3390/ai4010013

[9] D. B. Ansori, J. Slamet, M. Z. Ghufron, M. Aidiel, R. Putra, and T. Ahmad, "Android Malware Classification Using Gain Ratio and Ensembled Machine Learning," *Int. J. Saf. Secur. Eng.*, vol. 14, no. 1, pp. 259–266, 2024. DOI:10.3390/electronics14081486

[10] F. Shen, J. Del Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, "Android Malware Detection Using Complex-Flows," *IEEE Trans. Mob. Comput.*, vol. 18, no. 6, pp. 1231–1245, 2019. https://doi.org/10.1109/TMC.2018.2861405

[11] S. E. Mohamed, M. Ashaf, A. Ehab, O. Shereef, H. Metwaie, and E. Amer, "Detecting Malicious Android Applications Based On API calls and Permissions Using Machine learning Algorithms," in *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, 2021, pp. 1–6.DOI 10.1109/MIUCC52538.2021.9447594

[12] D. Aboshady, N. Ghannam, E. Elsayed, and L. Diab, "The Malware Detection Approach in the Design of

Mobile Applications," *Symmetry (Basel).*, vol. 14, no. 5, 2022. https://doi.org/10.3390/sym14050839

[13] M. Al-Fawa'reh, A. Saif, M. T. Jafar, and A. Elhassan, "Malware Detection by Eating a Whole APK," in *2020 15th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2020, pp. 1–7. DOI: 10.23919/ICITST51030.2020.9351333

[14] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing,* , 2020, pp. 515–522. DOI: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094

[15] Y. Zhang, S. Yang, L. Xu, X. Li, and D. Zhao, "A Malware Detection Framework Based on Semantic Information of Behavioral Features," *Appl. Sci.*, vol. 13, no. 22, 2023. https://doi.org/10.3390/app132212528

[16] Q. Xu, D. Zhao, S. Yang, L. Xu, and X. Li, "Android Malware Detection Based on Behavioral-Level Features with Graph Convolutional Networks," *Electronics*, vol. 12, no. 23, 2023. https://doi.org/10.3390/electronics12234817

[17] T. Liu, Z. Li, H. Long, and A. Bilal, "NT-GNN: Network Traffic Graph for 5G Mobile IoT Android Malware Detection," *Electronics*, vol. 12, no. 4, 2023. https://doi.org/10.3390/electronics12040789

[18] F. Giannakas, V. Kouliaridis, and G. Kambourakis, "A Closer Look at Machine Learning Effectiveness in Android Malware Detection," *Information*, vol. 14, no. 1, 2023. https://doi.org/10.3390/info14010002

[19] P. Musikawan, Y. Kongsorot, I. You, and C. So-In, "An Enhanced Deep Learning Neural Network for the Detection and Identification of Android Malware," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 8560–8577, 2023. DOI: 10.1109/JIOT.2022.3194881