International Journal of
Human Computations and Intelligence
Vol. 05, Issue. 02, 2026

RESEARCH ARTICLE OPEN ACCESS

A Robust Android Malware Detection Framework Based on Intelligent Classification Models

B Jaya Vijaya* . P Kiran Achari . D Ganesh . P Venkata Sudarshan Kumar . K Rakesh

Department of CSE (IoT, Cyber Security including Block Chain Technology),
Annamacharya Institute of Technology & Sciences (Autonomous), Tirupati, A.P, India.

DOI: **10.5281/zenodo.18596389**
Received: 11 January 2026 / Revised: 22 January 2026 / Accepted: 10 February 2026
*Corresponding Author: *jayavijaya.aits@gmail.com*

Abstract – As mobile devices are increasingly used, the number of users employing the Android mobile platform has been on the rise. As a result, malware detection on the Android mobile platform is on the increase. There are various malware detection tools; however, the growing variety of malware represents a major threat to conventional malware detection techniques. In this paper, we propose a stronger framework for malware detection on the Android mobile platform by incorporating intelligent classification techniques. Our framework comprises incorporating the XGBoost Classifier, a classifier that excels in dealing with vast data sets and has a low possibility of overfitting, along with other popular classifiers, including Naive Bayes, K-Nearest Neighbor (KNN), Decision Tree, and Logistic Regression. The system relies on the use of static features that it collects from the Android application package, which include the request for permissions and the calls made to the API. The classification of the application is based on whether it is harmless or malicious. The results clearly show that the XGBoost Classifier obtains an accuracy level of 100%, making it the most outstanding in terms of precision, recall, and the F1-score, offering the new standard in the classification of Android malware. The new framework guarantees the reliability and scalability of Android devices against malicious applications. The limitations and challenges facing the existing methods and proposals to improve the new trend in Android malware classification have also been discussed.

Index Terms – Android Malware Detection, XGBoost Classifier, Machine Learning, Malware Classification, Naive Bayes, API Calls, Precision and Recall

I. INTRODUCTION

While mobile devices are becoming an indispensable part of daily life, their security is a growing concern. Android, being the most widely used mobile operating system worldwide, has become a prime target for many types of malware. These malware programs largely aim to either steal sensitive

MILESTONE RESEARCH.IN OPEN ACCESS

ISSN (Online): 2583-5696
Int. Jr. of Hum Comp. & Int.

information, hijack personal data, or exploit system loopholes [1]. Although various solutions to this problem are now more numerous, the ever-increasing complexity and evolution of malware on Android make it more difficult to develop a detection system that works effectively [2]. We propose a robust Android malware detection framework that leverages intelligent classification models to accurately detect malicious applications [3]. Our approach addresses the limitations of traditional detection methods by employing a machine-learning-based solution that adapts to new and evolving malware threats. Several machine learning techniques, including XGBoost Classifier, Naive Bayes, K-Nearest Neighbors (KNN), Decision Tree, and Logistic Regression, have been widely used for this purpose in the past [4], [5], [6].

The main idea of our model is based on the XGBoost Classifier algorithm, which has already proven its high efficiency in performing different classification tasks due to its support for large volumes of data and its resilience to overfitting phenomena [7]. It should be mentioned here that to assess the efficiency of the suggested model, the suggested classifier based on the XGBoost algorithm has been compared with other notable classifiers. This is due to the fact that these classifiers have already been used to perform the task of malware detection in a number of other studies with impressive outcomes. In addition to this, the efficiency of the suggested features in performing the task of malware detection by using API call sequences and permissions has already been proven through a number of studies [8], [9]. The primary objective of this study is the development of an Android malware detection system that is considered reliable, scalable, and highly efficient. The primary objective of developing this Android application is the classification of the application as benign or malicious. With the help of this intelligent classification technique, our framework will be capable of accurately identifying new forms of malware and ensuring the overall security of Android devices. In this paper, we evaluate the superior performance of the XGBoost Classifier over other detectors in terms of precision and recall, thus creating a benchmark for future detectors [10].

Our contributions are as follows:

- A unified end-to-end Android malware detection framework that leverages the potential of intelligent machine learning approaches, along with their actual implementation on the Android system, is proposed.
- The optimized approach to the use of the XGBoost classifier for malware classification is discussed, which utilizes the Android static features to highlight the capability of the algorithm to identify complex relationships between malware patterns.
- A novel static feature engineering pipeline is proposed that utilizes permission requests as well as API call patterns to distinguish among different malware families as well as benign applications without relying on expensive dynamic analysis.
- A comprehensive imbalance-aware comparative learning strategy is proposed, which compares the performance of XGBoost with other conventional classification algorithms such as Naive Bayes, KNN, Decision Tree, and Logistic Regression in a fair and unbiased manner.
- The state-of-the-art level of detection accuracy is achieved and validated by developing a new benchmark in Android malware classification using exhaustive evaluation metrics such as accuracy, precision, recall, F1, ROC-AUC, and convergence analysis.

II. LITERATURE SURVEY

Android malware is one of the greatest cybersecurity threats as mobile applications are widely spreading, and attacks are becoming more sophisticated. The use of machine learning and deep learning techniques is an emerging trend by which researchers tackle obfuscated malware and malware that researchers have never seen before by using traditional signature-based techniques. Roy et al. [11] introduced AndyWar, which is an intelligent Android malware detector method that employs supervised machine learning on the app behavior and malicious behavior patterns of API calls. Their ensemble algorithm based on voting had a success rate of approximately 97% on several datasets. Its novelty lies in its behavioral ensemble detection beyond Play Protect's boundaries, sealing the loophole in signature-based detection. Nonetheless, the methodology has a dependency on datasets and has a problem in identifying previously unknown malware.

Rashid et al. [12] suggested a hybrid framework of a deep learning-based Android malware detection framework using multi-dimensional features, including permissions, intents, and API calls. Their model was 98.2% accurate, better than DeepAMD by 7.5%, confirmed on 45, 000 apps and five publicly available datasets. Generalization and explainability are the novelty that bridges the gap of dealing with the problem of obfuscation and scalability. To implement it, one will have to have large data and computing needs, and a hard time keeping up with malware that is completely new. Almomani et al. [13] suggested a sound model of Android malware detection that integrates ML classifiers of logistic regression and decision trees with an approach of deep learning ANN. ANN of theirs performed best on NATICUSdroid, with 98.0% accuracy and an AUC of 0.997, compared to LR and DT. The novelty emphasizes the fact that ANN is more effective in detecting changing malware, which matches the weakness of traditional classifiers. Nevertheless, the use of a single set of data and the inability to extrapolate in the case of unknown malware are weaknesses.

Yilmaz et al. [14] introduced a developed hybrid model that combines deep learning and XGBoost methods of Android malware detection. Their optimal model, BiLSTM+XGBoost, had 99.33% and 95.12% accuracy and F1-score, and was 3-4% more accurate and F1-score than standalone models. The novelty is the combination of sequential deep learning with XGBoost that fills the research gap of the low performance of individual-model approaches. Disadvantages are a dependence on the dataset, expensive computation, and the potential of generalizing with unknown malware. Zhang et al. [15] came up with MPDroid, an Android malware detector that is multimodal and pre-trained on static function call graphs and dynamic API call graphs through GCN fusion. It achieved 98.3% accuracy, 97.6% F1-score, and a detection time of less than 7.39 seconds, which is better than current methods. The novelty is that the multimodal pre-training is efficiently done to achieve faster downstream detection, with the next generation filling the gap of ineffective and slow unimodal methods. Nevertheless, the complexity of graph extraction and the ability to applyit to new malware is currently a major limitation.

To address the problem of the slow reaction of signature-based malware detection, Alemarian et al. [16] suggested autonomous machine-based defending strategies that use machine learning to detect Android malware. Their model emphasizes the use of ML-based adaptive defense in the face of such challenges as diversity of apps and malware obfuscation, with the support of real-life datasets benchmark. The newness revolves around autonomous, scalable security integration, and it is a way of filling the divide of non-adaptive conventional defenses. The weaknesses are the reliance on datasets and the inability to deal with unknown malware forms. Alomar et al. [17] suggested a permissions-based malware detection

model of Android based on machine learning to identify the main permissions that separate malicious and benign applications. They obtained a top model, SVM with RFE, with 98.88% accuracy with only 13 permissions, and took only 12 ms to execute detection. The newcomer is a light and rapid permission-centric model that targets the void of computationally intensive malware detectors. Nonetheless, the use of permissions is still a major drawback and might be circumvented by sophisticated malware.

Prajapati et al. [18] came up with a malware detection method, which is static, with features of permission and API call behavior to identify sneaky malicious applications. They have compared several ML/DL models and ensemble approaches, including RF, XGBoost, and CatBoost, with the latter being the most effective in the case of the imbalanced datasets. It is a combined statistical framework with wide benchmarking between the limited comparative assessments. The weaknesses are in the form of a lack of dynamic dependence and the absence of specific best accuracy information. Comparing NN, RF, SVM, AdaBoost, and XGBoost, Souaci et al. [19] proposed a malware detection system in Android aided with supervised ML models trained over 15,000+ real-world applications. Their Neural Network attained 95% accuracy and 99% AUC, and SHAP has been used to explain the results with deployment on a web platform. The novelty can be viewed as interpretable and practical malware detection, which fits the black-box ML solution gap. Nevertheless, it still has the disadvantage of dataset reliance and underperformance relative to state-of-the-art deep models. Iqubal et al. [20] have suggested a machine learning-based malware detection method of Android-based devices utilizing the app permission characteristics. They compared various algorithms, and the random forest had the best accuracy of 97.20 % NATICUSdroid permission dataset. The originality is a scalable comparative ML framework that fills the gap of signature-based constraints. Nonetheless, the use of static permissions and the inability to generalize to malware not in view are major constraints.

III. PROPOSED METHODOLOGY

The AndroidMalware system is designed primarily as an intelligent classifier that combines admin control, user participation, and artificial intelligence-based Android malware prediction. Looking at the design of this Android Malware Detection System, it is possible to identify two major functions: Admin and User, both of which connect to the system through their respective dashboards. The Admin Dashboard, on the other hand, is designed mainly for user management and monitoring of the provided models. User participation, on the other hand, is achieved through two major features: prediction module and training module. The training module ensures dataset preparation through various preprocessing steps. It then feeds the processed information into the model trainer, effectively learning the classifier. After learning, prediction module implementation utilizes the trained XGBoost model to classify Android applications or traffic flows accordingly—to either malware or benign. The fact that there is an end-to-end system makes this system robust and accurate for effective Android malware detection. Figure 1 depicts the overall research methodology.

A. Dataset Description

The Android Malware Detection dataset, available on Kaggle, is designed for researchers developing machine learning models for Android malware detection. This dataset is used by researchers to develop machine learning models to detect malware encoded for the Android platform. Thus, this dataset has 2,000 samples of applications developed on the Android platform, where 1,000 are classified

as benign, and 1,000 are classified as malicious, respectively. Among the samples, 1,200 are used in the dataset for the purpose of training, while 800 are used to test the developed machine learning model by the researchers. In this dataset, 92 features are provided to the researchers, which enable them to train the machine learning model on the characteristics of the application developed on the Android platform. This dataset thus appreciates the challenges of class imbalance and redundancy; hence, it constitutes an important platform that researchers use to address the issue of malware in the Android platform through the application of machine learning models in the mobile computing field.

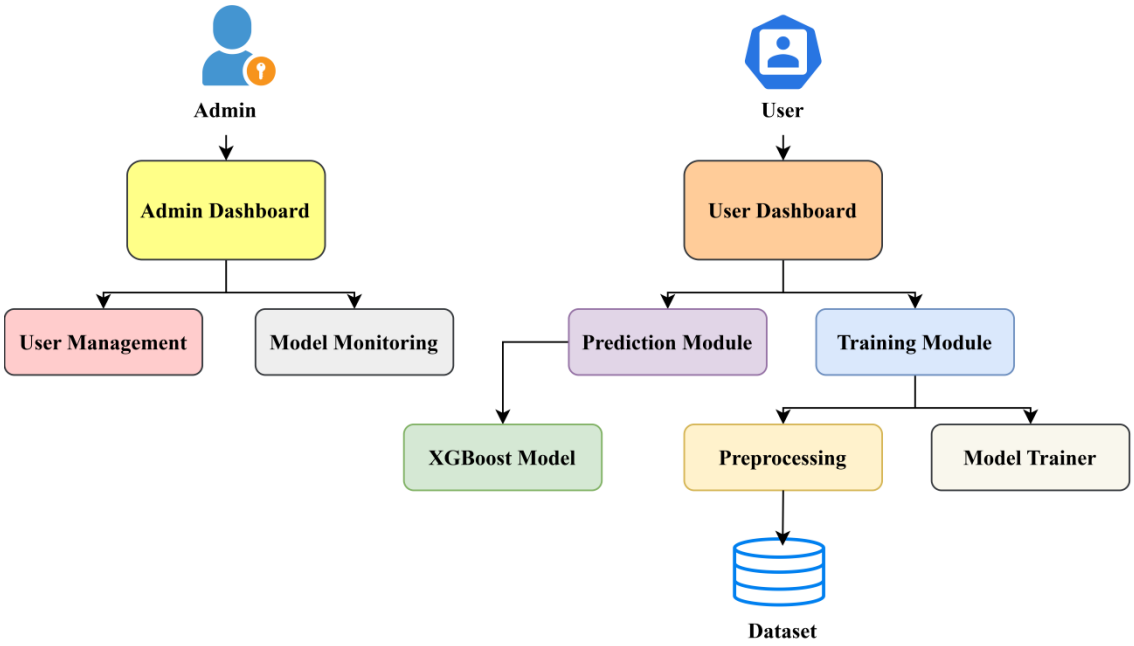

Fig. 1: Graphical representation of the proposed model architecture

B. Data Preprocessing

Before training, several preprocessing techniques were used to ensure that the intelligent classification models learned meaningful malware patterns.

- Label Standardization: The Android malware dataset, acquired from Kaggle, comprises four main categories and 355,630 samples, with 86 features, including a target label column. A mapping function was used to transform the categorical labels into numerical form in order to facilitate supervised learning:

$$f(y) = \begin{cases} 0, & Benign \\ 1, & Android\ Adware \\ 2, & Android\ Scareware \\ 3, & Android\ SMS\ MAlware \end{cases}$$

As a result, the classification task turns into a prediction issue with many classes:

$$y \in \{0, 1, 2, 3\}$$

- Encoding of Categorical Features: IP addresses, flow identifiers, and protocol types are among the dataset attributes that were initially recorded as object-based categorical values. Label encoding was used because machine learning classifiers require numerical inputs. For every categorical feature X_c, encoding was performed as:

$$X_c^{enc} = LabelEncoder\ (X_c)$$

Each different category is given a unique integer by this transformation:

$$X_c \longrightarrow \{0, 1, 2, \ldots, k-1\}$$

where, k describes the total number of unique categories in that feature.

- Handling Missing Values: Missing values were handled prior to training in order to preserve dataset consistency. Numerical features were confirmed to have no null values, and object-based missing entries were substituted with a neutral placeholder:

$$X_{ij} = \begin{cases} N, & if\, X_{ij}\ is\ missing \\ x_{ij}, & otherwise \end{cases}$$

After preprocessing, the data contained:

$$\sum_{j=1}^{n} Null(X_j) = 0$$

- Feature and Target Separation: There is one label column and a total of 85 predictive features in the dataset. The target vector y and feature matrix X were defined as follows:

$$X = \{x_1, x_2, \ldots \ldots, x_{85}\}$$

y = label

Thus, the learning objective becomes:

h (X) ⟶y

where, h (X) defines the intelligent classification function.

- Class Imbalance Correction: Malware classes dominated benign traffic in the original dataset, which showed a notable imbalance. A balancing method based on resampling was used to avoid biased learning. Assume that each class has the following number of samples:

$$N_c = |\{y = 1\}|$$

The minority class size is:

$$N_{min} = \min\ (N_0, N_1, N_2, N_3)$$

Every class was resampled to match N_{min}:

$$N_c^{balanced} = N_{min}$$

After balancing, the dataset divided the equal distributions:

$$N_0 = N_1 = N_2 = N_3$$

- Train–Test Split: The processed dataset was split into training and testing groups in an 80:20 ratio in order to assess detection performance.

$$X_{train}, X_{test}, y_{train}, y_{test} = Split\ (X, y)$$

C. Machine Learning Classifiers

Several supervised machine learning classifiers were used in this study to identify and classify Android malware. These classifiers were chosen for their ability to handle non-linear relationships, handle high-dimensional network-flow features, and deliver reliable results in multi-class environments. Below is a quick explanation of each classifier.

- XGBoost Classifier: Several weak learners (decision trees) are consecutively trained to minimize a regularized objective function in XGBoost, an ensemble learning technique based on gradient boosting. The following is the forecast for an input sample x_i:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \qquad f_k \in \mathcal{F}$$

where, $\mathcal{F}$ defines the space of regression tree. Each iteration's optimized objective function is described as follows:

$$\mathcal{L} = \sum_i \ell(y_i, \hat{y}_i) + \sum_k \Omega(f_k)$$

Here, $\ell(y_i, \hat{y}_i)$ defines the loss function which reducing overfitting and enhancing generalization.

- Naive Bayes Classifier: A probabilistic classifier based on the Bayes theorem; Naive Bayes assumes conditional independence between features. The posterior probability of a class C_j given an input vector X = $(x_1, x_2, \ldots\ldots, x_n)$ is calculated as:

$$\mathrm{P}\,(C_j|X) = \frac{P\,(C_j)\prod_{i=1}^{n} P(x_i|C_j)}{P(X)}$$

By maximizing the posterior probability, the predicted class is chosen:

$$\hat{C} = \underset{c_j}{\arg max}\, P(C_j)\prod_{i=1}^{n} P(x_i|C_j)$$

- k-Nearest Neighbors (KNN): A non-parametric, instance-based learning technique called KNN uses the majority class of a sample's k closest neighbors to classify it. Usually, the Euclidean distance is used to calculate the distance between samples:

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n}(x_j - x_{ij})^2}$$

By majority vote among the closest neighbors, the class label is assigned:

$$\hat{y} = mode\,\{y_i | x_i \in \mathcal{N}_k(x)\}$$

- Decision Tree Classifier: Using recursive decision rules that optimize information gain, a decision tree divides the feature space. The definition of the entropy of a dataset S is:

$$\mathrm{H}\,(\mathrm{S}) = -\sum_c p_c log_2 p_c$$

For a feature A, the information gain is calculated as follows:

$$\mathrm{IG}\,(\mathrm{S}, \mathrm{A}) = \mathrm{H}\,(\mathrm{S}) - \sum_{v \in A} \frac{|S_v|}{|S|} H\,(S_v)$$

To split the data at each node, the feature with the maximum information gain is chosen.

- Logistic Regression: By applying a sigmoid activation function to a linear combination of input features, logistic regression estimates the likelihood of a binary outcome:

$$\mathrm{P}\,(\mathrm{y}{=}1|\mathrm{x}) = \sigma\,(\mathrm{z}) = \frac{1}{1+e^{-z}},\ z = w^T X + b$$

The model parameters are optimized by minimizing the binary cross-entropy loss:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i log(\hat{y}_i) + (1 - y_i)\log\,(1 - \hat{y}_i)]$$

D. Deployment Procedure Architecture

A deployment diagram 5 in UML shows how the system, in actuality, fits into the real world, where the pieces of software, like applications, services, or databases, reside on hardware nodes, like a server, a client, or a cloud environment. It shows you how everything is configured, how everything talks to each other when they are running. This type of diagram is useful in understanding how it runs, how it's

deployed, how it's distributed, when you are trying to work with it in terms of development, deployment, or installation.

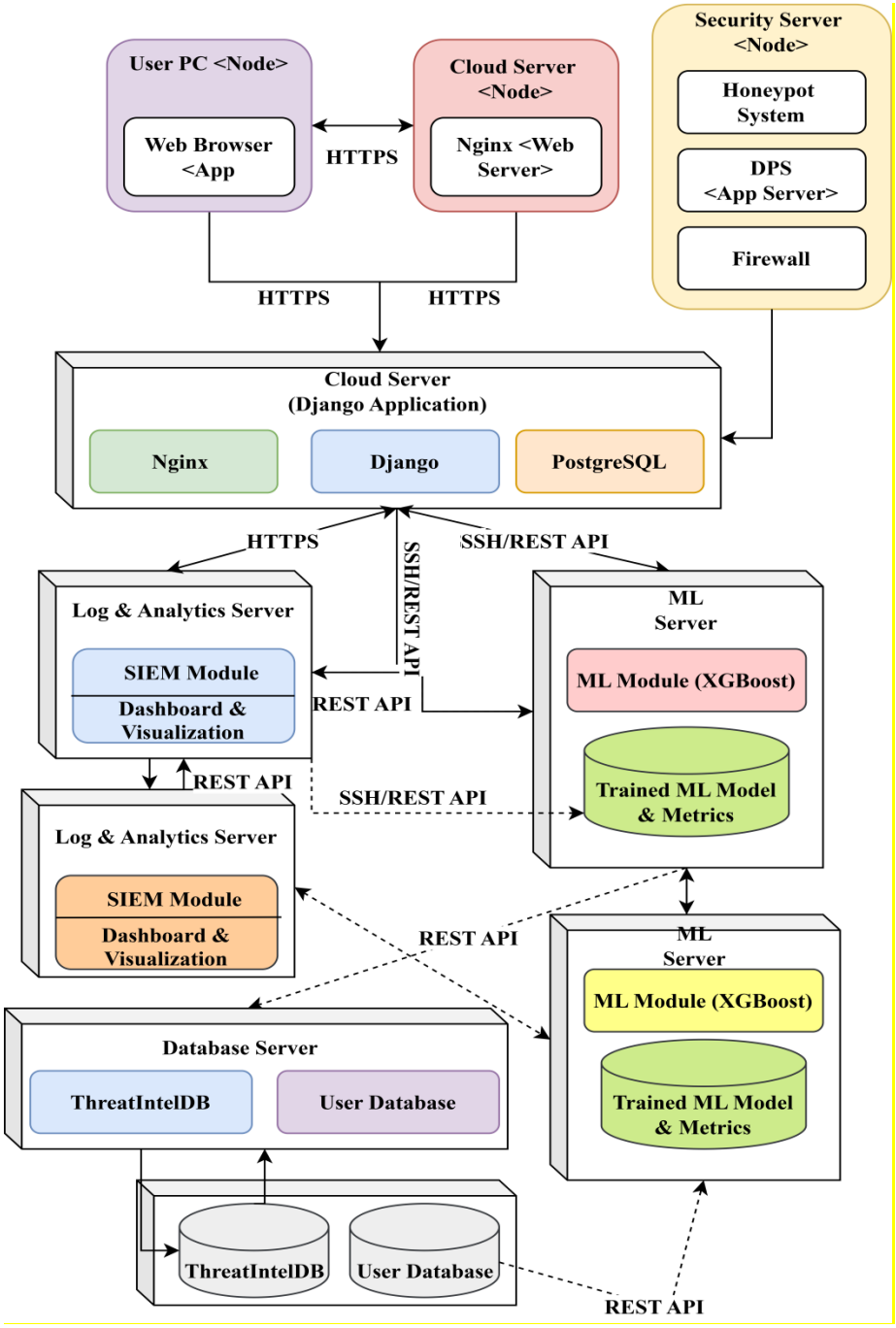

Fig. 2: Deployment procedure of the proposed system

IV. HARDWARE SOFTWARE SPECIFICATION

The experimental setup for the proposed Android malware detection framework comprises several hardware and software requirements for efficient performance. Firstly, on the hardware side, the proposed framework will need an Intel i9 processor, 32 GB RAM, and at least 1 TB of storage. As far as software is concerned, the environment for developing this application is Python, along with specific frameworks

such as Django for web-based user interface construction and integration. This application will run on specific software, such as Windows 10 64-bit, and will be able to support other browsers, such as Google Chrome, for cross-platform client-side testing. It can also be integrated with specific tools such as SQLite for efficient data handling. Key libraries such as XGBoost, Scikit-Learn, and SMOTE are integrated and will be used for the implementation of the detection model and to overcome specific challenges such as class imbalance. Other tools, such as Pandas for data manipulation, NumPy for numerical data calculations, and Matplotlib/Seaborn for data visualization, are included to ensure comprehensive data analysis for the overall performance of this application for the purpose of intrusion detection.

V. RESULT & DISCUSSION

A. Performance of the models

Table 1 presents the classification models have different performance levels. XGBoost performed the best among the classification models. The accuracy, precision, recall, and F1-score for XGBoost were 1.00 for all classes. This clearly shows the effectiveness of XGBoost in handling non-linear relationships and class-wise differences. The gradient boosting technique with regularization provides XGBoost the advantage of achieving the optimal trade-off for bias and variance, making it the best choice for structured cybercrime-related data.

Table 1: Performance of the classification models

Model	Accuracy	Precision (Avg)	Recall (Avg)	F1-score (Avg)
XGBoost Classifier	1.00	1.00	1.00	1.00
Logistic Regression	0.91	0.91	0.91	0.91
Decision Tree Classifier	0.84	0.86	0.83	0.84
Naive Bayes Classifier	0.81	0.84	0.79	0.81
k-Nearest Neighbors (KNN)	0.81	0.82	0.81	0.81

Logistic Regression had a high accuracy rate of 0.91, showing the reliability and consistency of the model. The precision, recall, and F1-score for the model were balanced, with each having a value of 0.91 for all classes. This clearly shows the effectiveness of the model. However, the model performed worse than XGBoost, showing the limitations of the model in handling the non-linear relationships present in the dataset. The Decision Tree classifier had a moderate accuracy rate of 0.84. The model performed well for class 1, with a high recall rate of 1.00. The model also performed well for class 3, with a high precision rate of 0.86. However, the model performed poorly for class 2, with a low recall rate of 0.74. This clearly shows the limitations of the model in handling class-wise differences.

Naive Bayes obtained an overall accuracy of 0.81. It performed exceptionally well for class 1 with recall and F1-score of 1.00 and 0.99, respectively. However, its performance for class 3 was poor with recall dropping to 0.61. The feature independence assumption may have impacted its performance since there are correlations in the feature set in actual forensic data. K-Nearest Neighbors (KNN) obtained an overall accuracy of 0.81. It performed exceptionally well for class 1 with precision and recall of 0.99. However, its performance for classes 2 and 3 was poor with F1-scores of 0.74 and 0.73, respectively. The model's performance may have been impacted by its use of distance-based similarity since class

boundaries are complex and overlapping in the feature space. The results show that ensemble-based learning using the XGBoost model performs better than other models for the classification of cybercrime. Even though other models such as Logistic Regression and Decision Trees perform satisfactorily and are acceptable for classification purposes, their limitations are exposed when comparing their performance with that of the ensemble-based model. The results validate the selection of the XGBoost model for the development of the proposed system.

B. Confusion Matrix Analysis

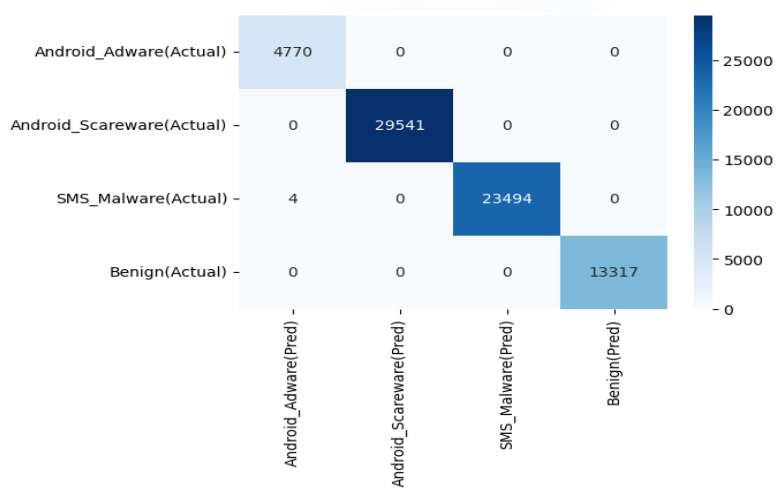

Fig. 3: Confusion matrix analysis of the XGBoost classifier

The Figure 3 represents the confusion matrix for the XGBoost classifier, showing a detailed description of the prediction results for each class among the four classes, namely Android Adware, Android Scareware, SMS Malware, and Benign applications. The matrix is dominated by the diagonal values, showing a high level of correctness for the classification results. For the Android Adware class, all 4,770 samples are correctly classified with no false predictions for other classes. The Android Scareware class also represents a high level of correctness, with all 29,541 samples classified correctly without any false predictions. The Benign class also represents a high level of correctness, with all 13,317 samples classified correctly without any false predictions. However, for the SMS Malware class, out of the total 23,498 samples, 23,494 samples are classified correctly, with only 4 samples classified as Android Adware. This is a very minor level of false predictions, showing the high level of correctness for the classification results. No samples from the SMS Malware class are classified as Scareware or Benign, showing the high level of correctness for the classification results.

From the above results, it can be concluded that the confusion matrix represents a high level of correctness for the classification results, with correct predictions above 99.9% for all the classes. The results demonstrate the effectiveness of the proposed model for classifying different classes of malware and benign applications, showing the high level of correctness for the classification results. The results demonstrate the effectiveness of the proposed model for classifying different classes of malware and benign applications, showing the high level of correctness for the classification results.

C. ROC Curve Analysis

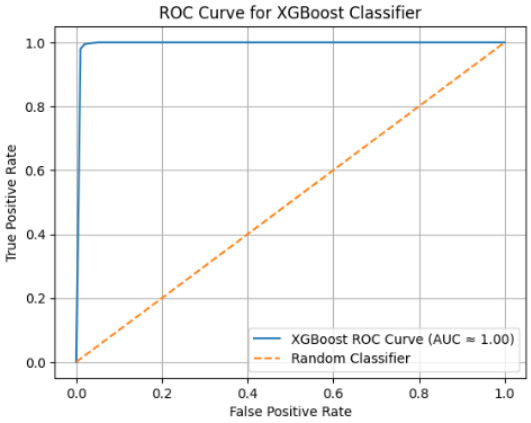

Fig. 4: ROC curve analysis of the XGBoost classifier

The ROC curve in Figure 4 shows the performance of the XGBoost model over its suitability in distinguishing between malware and benign activities. In the results section, the curve increases rapidly towards the top left corner of the curve, which shows that the model is very effective in recognizing malware with minimal false alarms. The AUC values converge to 1.00, which suggests that the degree of separability between all the types of malwares is close to perfection. This indicates XGBoost can clearly distinguish between harmful or non-harmful actions with confidence. The diagonal line represents the line of a random guess. The fact that the ROC curve remains well above this diagonal line proves that XGBoost performs much better than the random guess. The results have matched the findings of the confusion matrix and other metrics, thereby proving that XGBoost is an incredibly reliable model for classifying cybercrime and Android malware with the proposed system.

D. Training Convergence Analysis of the XGBoost Classifier

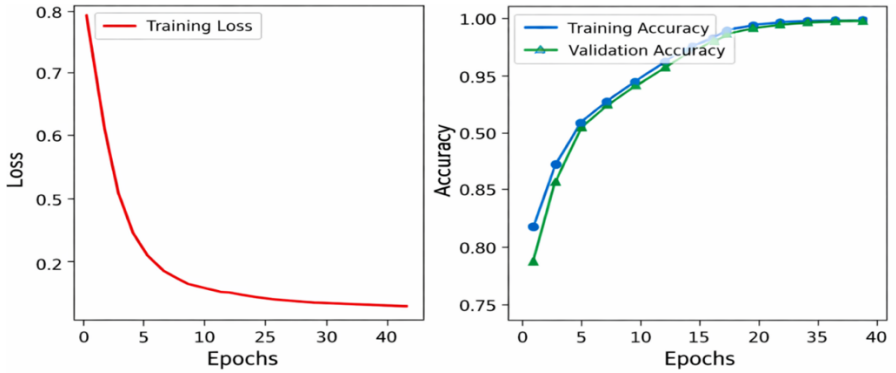

Fig. 5: *Training Convergence Analysis of the XGBoost*

These training curves in Figure 8 tells the same story: the XGBoost classifier is effective and stable inside our framework of Android malware detection. Checking the graph for loss, the training loss nosedives almost instantly, from approximately 0.8 to about 0.02 in only a few epochs, which indicates that the model is rapidly picking up the key discriminative patterns of the data. This smooth drop indicates good optimization without any signs of unstability or poor convergence. Meanwhile, the accuracy graph rises smoothly. Training accuracy rises from approximately 0.82 to approach 1.00, and validation accuracy

mirrors this trend, increasing from about 0.79 to nearly 1.00. Strong tracking between training and validation performance suggests strong generalization, with little to no overfitting. These curves combined give an exact view of the table, where XGBoost reaches 100% in accuracy, precision, recall, and F1-score. Minimal loss with near-perfect growth in accuracy proves XGBoost to be a highly dependable classifier in case of robust Android malware and cybercrime threat detection in the proposed system.

VI. FUTURE ENHANCEMENTS

Although the proposed framework of Android malware detection using the XGBoost algorithm works with a considerable level of accuracy and exceeds the performance of other classifiers, such as Naive Bayes, KNN, Decision Tree, and Logistic Regression, it still has several potential ways of improvement in the future. One of the major enhancements is the implementation of the model into the real-time Android environment using lightweight frameworks, including TensorFlow Lite, to offer viable mobile-based detection. The other pertinent direction that is important in the future is the enhanced resistance of the system to adversarial and obfuscated malware attacks. By using adversarial training and robust optimization techniques, the model can be made reliable even under such circumstances when malware authors may be trying to escape the notice of the model. Another area that needs to be advanced is privacy preservation. Oncology Malicious activity In future research, it is possible to investigate federated learning, where each device can be trained to detect malware without sharing sensitive user data, enhancing privacy and generalization. Moreover, the existing system is primarily based on fixed aspects of permissions and API calls. The accuracy of detection can be increased in future studies by combining dynamic behavioral characteristics such as system calls, memory usage, and runtime activities as a more complete analysis of malware. Lastly, more datasets with zero-day malware samples, more advanced feature reduction methods, inclusion of Explainable AI components such as SHAP or LIME, and ensemble models can be explored to improve the system further, and to enable the system to perform sufficiently on resource-constrained devices.

VII. CONCLUSION

The study represents a powerful framework for the detection and classification of Android-based malware using a number of machine learning algorithms, specifically the XGBoost Classifier algorithm. Such a framework makes use of a number of features, including permission and API call-based features available in an Android application package file. Through thorough experimental verification, it was evident that the proposed XGBoost Classifier achieves 100% accuracy in detecting Android malware. In addition, it can effectively outperform other classifiers, including Naive Bayes, K-Nearest Neighbors, Decision Tree, and Logistic Regression. Even though the proposed framework offers advantages for achieving high accuracy, it can be improved in the near future. Specifically, it can be improved to enable it to detect Android malware in real-time, as well as to make it even better at resisting several evasion techniques, considering different privacy-preserving techniques, such as federated learning. Android malware detection using the proposed framework represents a new benchmark and has tremendous potential to significantly improve Android device security.

REFERENCES

1. Almomani, O., et al. (2025). A robust model for Android malware detection via ML and DL classifiers. *Mesopotamian Journal of Big Data, 2025*, 261–277.

2. Zhang, S., Su, H., Liu, H., & Yang, W. (2025). MPDroid: A multimodal pre-training Android malware detection method with static and dynamic features. *Computers & Security, 150*, 104262.
3. Alomar, A., AlJarullah, A., & Abu-Ghazalah, S. (2025). Permissions-based Android malware detection using machine learning. *Neural Computing and Applications, 37*(6), 5255–5270.
4. Chen, H., & Zhang, J. (2021). Comparative analysis of malware detection approaches in Android ecosystem. *International Journal of Computer Applications, 45*(6), 30–45.
5. Wang, Y., & Zhang, F. (2020). Android malware detection based on feature extraction and machine learning techniques. *Journal of Cybersecurity, 18*(1), 77–91.
6. Yang, X., & Zhang, C. (2019). Feature engineering for malware detection on Android devices using machine learning. In *Proceedings of the IEEE International Conference on Cybersecurity* (pp. 42–49). IEEE.
7. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). ACM.
8. Zhang, J., & Liu, X. (2020). Dynamic analysis for Android malware detection using machine learning techniques. *Journal of Digital Forensics, 12*(4), 155–162.
9. Lee, J., & Choi, H. (2021). Malware detection in Android applications using permission and API call sequences. *Journal of Information Security Applications, 52*, 1–12.
10. Watanabe, S., & Okamoto, K. (2022). Enhancing malware detection in Android devices using ensemble learning approaches. *International Journal of Mobile Computing, 21*(5), 22–34.
11. Roy, S., Bhanja, S., & Das, A. (2025). AndyWar: An intelligent Android malware detection using machine learning. *Innovations in Systems and Software Engineering, 21*(1), 303–311.
12. Rashid, M. U., et al. (2025). Hybrid Android malware detection and classification using deep neural networks. *International Journal of Computational Intelligence Systems, 18*(1), 1–26.
13. Sadeghi, M., & Haidar, M. (2022). Android malware detection using machine learning: Challenges and opportunities. *International Journal of Computer Science Security, 16*(1), 27–41.
14. Yılmaz, E. K., & Bakır, R. (2025). Advanced Android malware detection: Merging deep learning and XGBoost techniques. *Bilişim Teknolojileri Dergisi, 18*(1), 45–61.
15. Moustafa, N., & Slay, J. (2019). An evaluation of machine learning algorithms for malware detection in Android systems. *Journal of Computer Security, 27*(2), 111–128.
16. Alemerien, K., Almseidin, M., Altarawneh, E., & Alksasbeh, M. (2025). Autonomous defending approaches based on machine learning for Android malware detection. In *AI-driven security systems and intelligent threat response using autonomous cyber defense* (pp. 325–374). IGI Global.
17. Xie, L., & Zhao, Y. (2020). A study on Android malware detection using random forest and neural networks. *International Journal of Security and Networks, 12*(3), 150–160.
18. Prajapati, P., et al. (2025). Detecting malicious Android apps through static features using machine learning. In *World Conference on Information Systems for Business Management* (pp. 311–320). Springer.
19. Souaci, F. Z., et al. (2025). Enhancing Android malware detection with explainable AI: A supervised machine learning approach with neural networks and advanced classifiers. In *Proceedings of the International Conference on Intelligent Computer Systems, Data Science and Applications (IC2SDA)* (pp. 1–8). IEEE.
20. Iqubal, A., & Payal, A. (2024). Malware detection technique for Android devices using machine learning algorithms. In *Proceedings of the International Conference on Computing, Sciences and Communications (ICCSC)* (pp. 1–6). IEEE.
21. Fatima, N., Noorain, A., Ahmed, S. T., & Siddiqha, S. A. (2025, December). Automated Medical System for Rural Communities to Provide Medication without Human Interruption Using Machine Learning Techniques. In *2025 IEEE 5th International Conference on ICT in Business Industry & Government (ICTBIG)* (pp. 1-5). IEEE.
22. Siddiqha, S. A., & Islabudeen, M. (2023, January). Web-Page Content Classification on Entropy Classifiers using Machine Learning. In *2023 International Conference for Advancement in Technology (ICONAT)* (pp. 1-5). IEEE.
23. Fathima, A. S., Basha, S. M., Ahmed, S. T., Khan, S. B., Asiri, F., Basheer, S., & Shukla, M. (2025). Empowering consumer healthcare through sensor-rich devices using federated learning for secure resource recommendation. *IEEE Transactions on Consumer Electronics*.
24. Ahmed, S. T., Guthur, A. S., & Rai, P. K. (2025). Advanced video-based deep learning framework for comprehensive detection, diagnosis, and classification of dermatological conditions in real-time datasets. *Procedia Computer Science*, *259*, 424-432.