
RESEARCH ARTICLE

A Hybrid AI System for Early Malware Detection and Automated Cybersecurity Risk Assessment

Nikhil Kashyap Adhikari

Independent researcher, University of the Cumberlands

Corresponding Author: Pending confirmation, **E-mail:** Nikhiladhkr55@gmail.com

ABSTRACT

The development of improved algorithms for malware identification and classification is necessary the considerable challenge posed by the increasing proliferation of malware in terms of digital security. When dealing with uneven datasets or complicated virus patterns, traditional detection approaches could fail. This research seeks to create efficient ML models for Android malware detection by utilizing two suggested approaches: a Multilayer Perceptron (MLP) model and a Hybrid XGBoost-Random Forest stacking model. Common classification metrics and five-fold cross-validation are utilized to train and assess the models. The Hybrid model showed impressive detection capabilities with an accuracy of 97.87%, while the MLP attained an accuracy of 97.38%. Results from classification using the MLP were comparable, and its performance was consistent throughout validation folds. The Hybrid model's interpretability was further enhanced by using SHAP analysis to uncover key Android properties. Overall, the research contributes two complementary machine-learning approaches for Android malware detection, offering accurate, reliable, and interpretable solutions for strengthening mobile application security.

KEYWORDS

Cybersecurity, Malicious Software, Android Malware Detection, Signature-Based Detection, Machine Learning, Explainable AI.

ARTICLE INFORMATION

ACCEPTED: 19 September 202

PUBLISHED: 23 October 202

DOI: 10.32996/jcsts.2026.8.11.3

1. Introduction

Cyberattacks are currently the most important issue in the field of contemporary technology. The term refers to exploiting a system's weaknesses for evil reasons, such as stealing from, modifying, or destroying it. A cyberattack is exemplified by malware. Any software or collection of instructions intended to damage a computer, user, company, or computer system is referred to as malware (Patel, 2026)(Arveti, 2026). Malicious software (also spelled "malware") may take many forms, including but not limited to viruses, Trojan horses, spyware, adware, rogue software, wipers, scareware, and ransomware. Any program code that executes independently of the user's awareness or permission is considered malicious software (Sethi, 2019)(Maghanaki, 2026). As more malware is developed for covert gains, signature-based detection methods are inadequate in providing essential cybersecurity. Thus, it is critical to create sophisticated and effective methods for detecting and reducing malware infections (Azeem, 2024)(Lingam, 2025).

In today's rapidly evolving digital landscape, malware is increasingly being crafted with sophisticated techniques that allow it to bypass traditional detection systems effectively and efficiently. While signature-based methods of malware detection remain prevalent, their limitations have become apparent over time (Alharbi, 2025)(Rane, 2024). Attackers, including both commercial entities and freelance hackers, are leveraging advanced obfuscation and evasion techniques to create new, resilient strains of malware that can avoid conventional detection approaches (Panja, 2025). As these traditional methods reach their functional limits, there is a pressing need to explore more adaptive and robust alternatives. Cybersecurity professionals have recently placed a premium on ML techniques for malware detection and family behavior prediction (Feng, 2023). The use of obfuscation and evasion techniques by attackers to avoid traditional detection measures is on the rise. For this reason, research into malware detection methods based on ML is gaining steam, especially those that can detect intricate and novel forms of cybercrime. Improving

cybersecurity in the face of ever-changing threats is driving this study toward better malware detection algorithms. The following are this study's main contributions:

- Developed a hybrid XGBoost–Random Forest stacking model for Android malware detection, combining the strengths of different ensemble learning techniques.

- The proposed model provides improved overall classification performance compared with MLP and conventional machine-learning classifiers.

- Utilized important Android application characteristics, including manifest permissions, API call signatures, intents, and command signatures.

- Evaluated model consistency using cross-validation, supporting its reliability across different data partitions.

- Applied SHAP analysis to identify influential features and provide clearer insights into the model's malware predictions.

- Provides a robust and interpretable framework that can support future development of automated Android malware detection systems.

The structure of this paper is as follows. A thorough analysis of relevant studies is provided in Section II. Section III describes the proposed methodological approach. Section IV presents the experiments, results and evaluation. Section V concludes the paper and identifies potential future research.

2. Literature Review

This section reviews existing Android malware detection techniques, focusing on traditional methods and ML approaches. It highlights their limitations and identifies research gaps that motivate the proposed models.

Srirangam et al. (2026) By using a CNN to automatically learn discriminative structural patterns linked with dangerous behavior, the suggested method converts executable binaries into grayscale pictures and does away with predetermined signatures. They constructed and analyzed two CNN architectures. The optimized model demonstrated strong detection performance with an accuracy of 91.72%, a precision of 90.28%, a recall of 95.0 %, and a F 1 -score of 92.58% (Srirangam, 2026).

Govindarajan et al. (2026) discusses the concept of UE-MCDA (Unified Evolutionary Multi-Domain Cybersecurity Detection Architecture), a machine learning architecture used to tackle both threats in one system using three datasets. ANN, which used GA-selected features, achieved 91.07 % on Android malware. Gradient Boosting had a score of 97.10 % on structured phishing. Random Forest scored 90.30% on raw URL phishing. The three pipelines do not rely on external data. This work supports UN SDG 9 (Govindarajan, 2026).

Kurniawan et al. (2025) create a powerful and effective ML system for detecting Android malware early on using static analysis. Six machine learning algorithms SVM, RF, KNN, NB, AdaBoost, and GB are used to identify malicious behavior in APK files. The dataset used consists of 2,084 Android applications, including 1,314 malware samples and 770 benign applications, obtained through a reverse engineering process. An experimental outcomes show that the Random Forest algorithm achieves the best performance with 97% accuracy and 95% precision, surpassing the performance of other algorithms (Kurniawan, 2025).

Prathapaneni et al. (2024) leverages an ensemble machine learning approach, including classifiers like Random Forest, K-nearest Neighbours, and others, for dynamic feature multi-classification using the CCCS-CIC-AndMal-2020 dataset. The research showed improvements in the accuracy of malware detection across various categories, with some categories showing a significant increase in accuracy. For instance, the accuracy for Adware improved from an average of 33.84% to 81.96%, and for Backdoor, it improved from 67.42% to 83.71% (al., 2024).

Vaghasiya et al. (2024) addresses the increasing security challenges in critical infrastructure systems due to cyber threats, particularly malware and propose a framework based on ML for robust malware classification. Our proposed framework includes a dataset of Windows API functions used by the software, preprocessing techniques, and the XGBoost model which achieves an accuracy of 86.98%. Using this model, we can detect and categorize five different types of malwares to provide security against malware attacks (Vaghasiya, 2024).

Bokolo, Jinad and Liu (2023) compares seven ML and DL techniques in detecting malware by using the extracted byte, opcode, and section codes. In this research, we aim to classify malware in nine different malware families correctly. First, the byte codes, the section codes, and the opcodes of the different malware applications are extracted and merged, and the classification is done by using the RF, DT, SVM, K-nearest neighbor, SGD, Logistic Regression, Naïve Bayes, and deep learning techniques. The result shows that the DL model outperforms other compared machine learning algorithms with an accuracy of 96% (Bokolo, 2023).

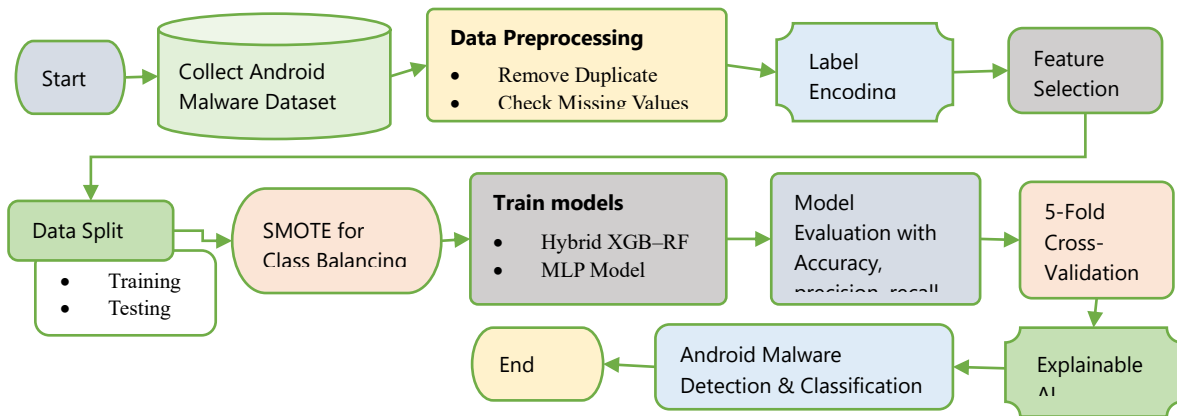
Comparative Summary of Recent Machine Learning and Deep Learning-Based Malware Detection Studies

Author, Year	Objective	Data / Dataset	Findings	Research Gap
Srirangam et al. (2026)	Developed an AI-based malware detector using CNNs and grayscale binary images.	Executable binary files	Optimized CNN achieved 91.72% accuracy, 90.28% precision, 95% recall, and 92.58% F1-score.	Limited evaluation of zero-day malware, behavioural characteristics, and generalization.
Govindarajan et al. (2026)	Proposed UE-MCDA for unified detection of Android malware and phishing attacks.	Three datasets covering Android permissions and phishing URLs.	ANN achieved 91.07%, Gradient Boosting 97.10%, and RF 90.30% performance.	Separate pipelines limit complete cross-domain integration and evolving-threat detection.
Kurniawan et al. (2025)	Proposed static-analysis ML framework for early Android malware detection.	2,084 APKs containing malware and benign applications.	RF performed best with 97% accuracy and 95% precision.	Static analysis can be affected by obfuscation and packing; dynamic features were limited.
Prathapaneni et al. (2024)	Used ensemble ML for dynamic multi-class Android malware classification.	CCCS-CIC-AndMal-2020	Adware accuracy improved from 33.84% to 81.96% and Backdoor from 67.42% to 83.71%.	Some malware classes remain difficult to classify; zero-day detection requires further study.
Vaghasiya et al. (2024)	Developed ML-based malware classification for critical infrastructure.	Windows API function dataset	XGBoost achieved 86.98% accuracy for five malware types.	Relatively limited accuracy and reliance on API-based features.
Bokolo et al. (2023)	Compared ML and DL models for malware-family classification.	Byte, opcode, and section codes	Deep learning achieved approximately 96% accuracy.	Limited investigation of explainability, adversarial robustness, and unseen malware.

The reviewed studies demonstrate that ML and DL techniques can effectively detect and classify malware, but several limitations remain. Current solutions are mostly based on static features, API calls, permissions, binary representations or on individual behavioral attributes. In addition, many studies assess models on one dataset only, which do not necessarily show cross-dataset generalization and resistance to new or unknown malware. Moreover, the number of integrations of multimodal features, explainable AI (XAI), adversarial robustness, and automated feature optimization in a single malware detection framework is limited. These restrictions make a more powerful, generalizable and explainable AI-based malware detection system more than desirable. The comparative summary of these studies is given in Table I.

3. Research Methodology

The Android Malware Dataset was preprocessed by removing duplicate records, encoding the class labels, eliminating highly correlated features, and standardizing the remaining features. After splitting the data into a training set and a testing set, SMOTE was used on the training set to fix the class imbalance. A hybrid XGBoost–Random Forest stacking model and an MLP model were trained for malware classification. Their performance was evaluated using accuracy, precision, recall, F1score, and AUC-ROC, while 5-fold cross-validation was used to assess generalization. SHAP analysis was also performed to interpret feature importance and identify the most influential features for Android malware detection.



Proposed Flowchart for Malware Detection

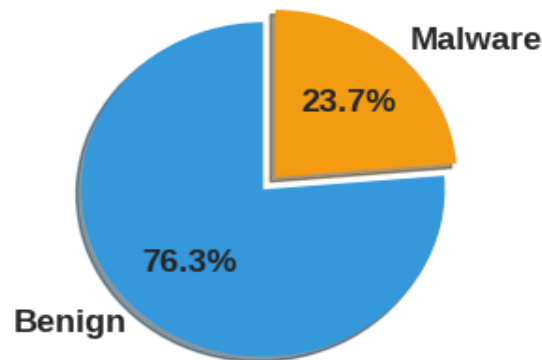
Each step of Fig. 1 the flowchart and the methodology are discussed in below:

3.1 Data Collection and Pre-Processing

The dataset utilized in this study is the Android Malware Dataset (Tiwari, 2021) from Kaggle, which has 15,036 samples of applications and 215 features. The dataset has two classes: Benign (B) and Malware (S). The features are the properties of Android apps such as 113 manifest permission features, 73 API call signature features, 23 intent features, 6 command signature features, and class label. The original data set had 7,776 duplicate records, and after eliminating the duplicates, 7,260 unique samples were obtained, with 5,540 being benign applications and 1,720 malware applications. No missing values were detected, thus creating a stable basis for further processing and malware classification experiments.

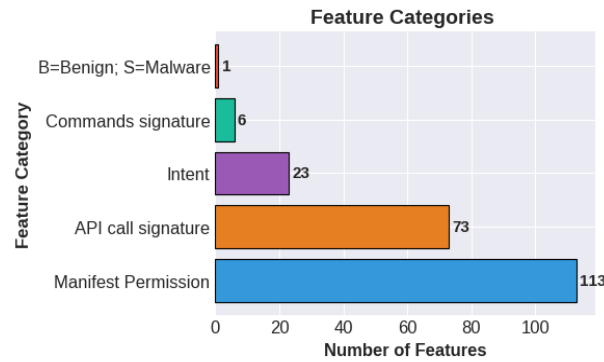
3.2 Exploratory Data Analysis (EDA)

EDA was performed to understand the class distribution and feature composition. Fig. 2 shows the percentage of malicious and benign occurrences in the data set. Benign class has the majority of instances with 76.3% and Malware class is 23.7%. This visualization shows the distribution of classes, with benign samples being the majority and malware the minority, essential for comprehending the makeup of a dataset for classification problems.



Class Distribution of Benign and Malware Samples

In application analysis, features are distributed in 5 categories in Fig. 3. The highest number of features are in the Manifest Permission category (113 features), followed by the API call signature (73 features) and Intent (23 features). Commands signature adds 6 features, B=Benign; S=Malware has 1 feature. The visualization emphasizes the relative usage of permission-based and API features to other types of features.



Distribution of Feature Categories

3.3 Label Encoding and Feature Selection

The class labels were changed from categorical values to numerical values so that it could be used by ML techniques. The dataset consists of the Benign class (B) and Malware class (S). Label encoding was performed with B = 0 and S = 1. So, every application has been assigned a class number corresponding to the benign or malicious nature of the application. The feature-selection process was done after label encoding. There were 215 features in the original data. No features were found to have zero variance, and therefore all 215 features were kept. Next, 4 highly correlated features were removed due to feature redundancy, based on the correlation coefficient of the features, which was greater than 0.95. To that end, 211 characteristics made it into the final collection, which were then used in the building of the model.

3.4 Train-Test Split

The preprocessed data was partitioned into 80% training data and 20% testing data, total of 5808 training data and 1452 testing data. The training set was used to develop the models and the test set was held for the final evaluation. The training data was also split and validated using 5-fold cross validation to check the stability and generalization of the model for different data splits.

3.5 Normalization of Data

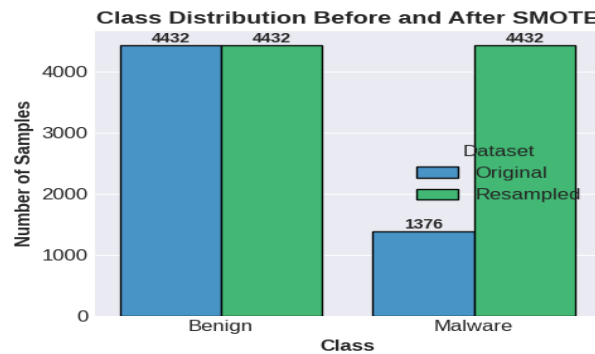
Normalization standardizes or scales data to a common range or format. Data normalizations help to prevent numerical instability and remove bias towards variables with higher values. In this study, Min-Max normalization techniques were employed to scale the features to a range between 0 and 1. Normalization can be expressed mathematically as Equation (1):

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (1)$$

X is the original value, x' is the normalized value, $\min(x)$ is the minimum value in the dataset, and $\max(x)$ is the maximum value of the feature.

3.6 Handling Class Imbalance

The dataset initially suffered from a significant class imbalance, where the Benign class (4,432 samples) heavily outnumbered the Malware class (1,376 samples). To address this, SMOTE was applied to oversample the minority Malware class to 4,432 samples, achieving a balanced distribution between the two classes. Fig. 4 shows the impact of data balancing with and without applying SMOTE.



Class Distribution Before and After SMOTE

3.7 Proposed Xgboost–Random Forest Stacking Model

XGBoost–Random Forest (RF) stacking model is proposed as a hybrid ensemble method for Android Malware Classification. In order to capture complex relationships among the features of the Android applications. XGBoost gradually creates decision trees to boost

the accuracy of the prediction, whereas Random Forest averages the predictions of multiple decision trees to make strong classification. In the stacking framework, the stacking procedure drawn from both models was passed to a meta-classifier to obtain the final prediction of malware or benign. The final test set was utilized as an independent test set to assess the model after the training dataset was balanced using SMOTE. The model was also validated using 5-fold cross validation to check model stability and generalization ability.

3.8 Multilayer Perceptron (MLP)

The Multilayer Perceptron (MLP) is a feed-forward artificial neural network consisting of an input layer, one or more hidden layers, and an output layer. Each neuron computes a weighted sum of inputs followed by an activation function in Equation. (2) and (3):

$$z_j = \sum_{i=1}^n w_{ij} x_i + b_j \quad (2)$$

$$a_j = f(z_j) \quad (3)$$

where x_i represents the input features, w_{ij} represents the weights, b_j is the bias, and f is the activation function. For binary malware classification, the output layer estimates the probability of the malware class, commonly using the sigmoid function in Equation (4):

$$P(y = 1 | x) = \frac{1}{1+e^{-z}} \quad (4)$$

During training, the network iteratively adjusted its weights and biases using backpropagation to minimize the classification loss. After training, the model was evaluated on the independent test set performance matrix. 5-fold cross-validation was also conducted to verify the stability and generalization performance of the MLP.

3.9 Model Evaluation

The model's Accuracy, Precision, and Recall rates are computed using a confusion matrix. Precision and Recall were also selected as evaluation metrics to provide a thorough assessment of the model's performance. TN is a measure of the number of samples when the projected and actual results are negative. The total positive samples (TP) are those for whom the actual and projected results are positive. The number of samples where the anticipated result is negative while the actual result is positive is equal to FN. FP is the count of samples where the expected value is positive but the actual value is negative. Equation (5) to (9) shows the computation formula.

$$Accuracy = \frac{TN+TP}{TN+TP+FN+FP} \quad (5)$$

$$Precision = \frac{TP}{TP+FP} \quad (6)$$

$$Recall = \frac{TP}{TP+TN} \quad (7)$$

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (8)$$

$$AUC = \int_0^1 TPR(FPR^{-1}(t)) dt \quad (9)$$

Precision is a statistic that quantifies the percentage of genuine positive predictions among all positive predictions generated by the model, whereas accuracy counts the percentage of right forecasts among all the predictions. The percentage of correct predictions relative to the total number of positive cases in the dataset is called recall, which is also called sensitivity. F1 score is a metric that combines precision and recall into a single value, providing a balanced assessment of the model's performance. An evaluation statistic for classification model performance is the AUC-ROC. This metric evaluates the model's discriminatory power across different threshold values for positive and negative classes.

3.10 Model Interpretability

Lastly, SHAP (Shapley Additive explanations) was used to explain the trained model and pinpoint the features that had the greatest influence on the predictions for malware on Android devices. This provided insight into the characteristics that were most influential in distinguishing benign applications from malware.

4. Results and Discussion

Python 3.9 on a 64-bit Windows 10 PC is used for the experiments presented in this article, which are built on an ML framework. The gear comprises an Intel(R) Core(TM) i7-8750H CPU running at 2.20 GHz and 2.21 GHz, an NVIDIA GeForce GTX 1050 Ti GPU, and CUDA 12.1. Table II shows that the suggested Hybrid model achieved 97.87% accuracy in total classification, which is better than the MLP model. Although the MLP achieved a slightly higher recall compared with Hybrid model, its overall F1-score was lower at 94.46%. Table III further demonstrates the consistency of both models across five folds. The Hybrid model achieved a mean F1-score of 0.9839 ± 0.0034 , while the MLP achieved 0.9819 ± 0.0025 , indicating stable and reliable generalization, with the Hybrid model showing a slightly higher average F1-score.

4.1 Experimental Performance of Proposed Models

Model	Accuracy	Precision	Recall	F1-Score
Hybrid	0.9787	0.9700	0.9390	0.9542
MLP	0.9738	0.9474	0.9419	0.9446

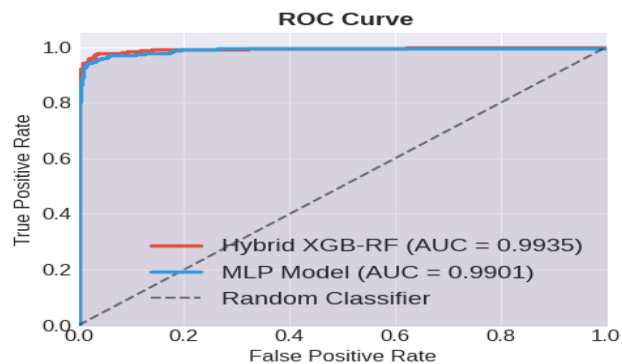
Five-Fold Cross-Validation Results

Model	F1-score					Mean F1
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Std. Dev.
Hybrid	0.9802	0.9813	0.9831	0.9899	0.9853	0.9839 ± 0.0034
MLP	0.9792	0.9787	0.9853	0.9831	0.9832	0.9819 ±

Hybrid XGB-RF Model		MLP Model	
True Label \ Predicted Label		True Label \ Predicted Label	
Benign (B)	1098	1090	18
Malware (S)	21	20	324

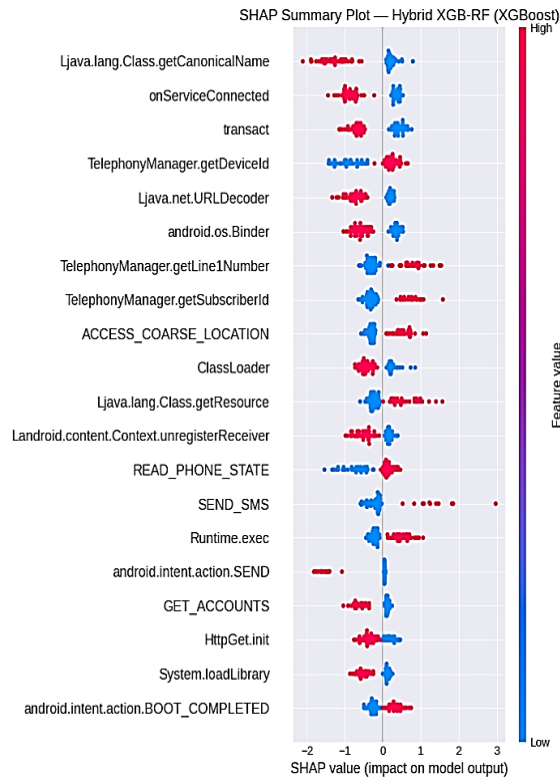
Confusion Matrices of the Hybrid XGB-RF and MLP Models

Fig. 5 presents confusion matrices comparing the detection capability of the Hybrid XGB-RF and MLP models. The Hybrid XGB-RF correctly classifies 1098 benign and 323 malware, and incorrectly classifies 10 benign samples as malware and 21 malware samples as benign. Compare this to the MLP, which has 1090 correct benign predictions, and 324 correct malware predictions, but with a slightly higher rate of FP and nearly equal false negative. The findings indicate that although both models perform similarly, the MLP model has a balanced performance in terms of malware detection, while the Hybrid XGB-RF model seems to have a little improvement in avoiding false alarms.



ROC Curve Comparison of Both Models

The Recurves of the Hybrid XGB-RF model and MLP model are shown in Fig. 6. The x-axis is the FPR and the y-axis is the True Positive Rate (TPR). The Hybrid XGB-RF model has an AUC of 0.9935, which is marginally better than the MLP model with an AUC of 0.9901. Both curves are very close to the top left corner, showing that the classification is very good and these curves have a good discriminative ability. The dashed diagonal line represents the random classifier (AUC = 0.5), providing a baseline for comparison.



SHAP Summary Plot for Model Output

The contribution of the top 20 features to the Hybrid XGB-RF model prediction are shown in Fig. 7. Features such as `Ljava.lang.Class.getCanonicalName`, `onServiceConnected`, `transact`, and `TelephonyManager.getDeviceId` show strong influence on the model output. The model output is considerably influenced by `getDeviceId`. Red points signify "high value" and blue points signify "low value" features. Positive SHAP values boost the prediction while negative values reduce the prediction, indicating the role of the individual features in the classification of Android malware.

4.2 Comparison and Discussion

The performance comparison presented in Table IV demonstrates the evolution of the conventional classifiers to the proposed models. The traditional AE and NB methods achieve marginal gains having accuracy of 75.79% and 52.75% respectively, which shows that the methods do not have sufficient robustness. The best performance is obtained by SMO with an accuracy of 91.8%, an F1 score of 91.3% and the lowest performance is obtained by NB. The suggested hybrid model achieves the maximum accuracy and performs better than any of the basic models. MLP's accuracy of 97.38% and 94.74% is also outstanding. The suggested hybrid model is the best model among the models assessed, according to the overall findings.

Comparison Between Base and Proposed Model Performance

Model	Accuracy	Precision	Recall	F1-score
AE [18]	75.79	92.15	66.78	77.44
NB [19]	52.75	52.80	58.60	50.10
RF [20]	89.2	88.5	89.0	88.7
SMO [21]	91.8	92.6	91.8	91.3
DT [22]	90	91	90	90
Hybrid	97.87	97.00	93.90	95.42
MLP	97.38	94.74	94.19	94.46

The suggested research can be seen as offering a number of significant advantages to the process of detecting malicious Android applications. Firstly, the combination of the XGBoost algorithm and Random Forest results in more efficient and reliable classification compared to traditional machine learning techniques and the MLP method. Secondly, the employment of ensemble learning allows for a better discrimination between benign applications and malware, while cross-validation confirms high consistency regardless

of the chosen dataset division. Moreover, the SHAP analysis helps reveal the key features of an Android application, making the whole detection process more understandable.

4.3 Novelty and Justification

The novelty of this study involves proposing a stacking framework for Android malware detection based on the combination of the capabilities of XGBoost and Random Forest approaches. It makes use of the advantages of both methods of ensemble learning. Furthermore, the proposed method considers a wide range of features of Android applications, such as manifest permissions, API calls signature, intents, command signature, thus making it possible to detect different behavior. In addition to that, using the SHAP-based approach provides an explanation of what features are responsible for malware predictions. There is a justification for this research since there is an ever-growing difficulty in classifying Android malware and the insufficiency of the existing classifiers in dealing with the diversity and imbalance of the data.

5. Conclusion and Future Work

Android malware detection is an important research area due to the continuous growth and changing nature of mobile threats. This research developed a hybrid XGBoost–Random Forest stacking model to improve the detection of malicious Android applications. The proposed approach showed better overall performance than the MLP and conventional machine-learning models. The cross-validation results also demonstrated consistent performance across different folds. SHAP analysis provided further insight into the important features contributing to malware classification, making the model more interpretable.

Despite these benefits, the study has some limitations. It uses a single publicly available dataset, which may not represent all types of Android applications and emerging malware. The selected features mainly describe static application characteristics and may not capture dynamic malware behavior. The model was also not tested on a large, independently collected real-world dataset. Future work can focus on evaluating the model with larger and more recent datasets. Dynamic features, application behavior, and network activity can be incorporated to improve detection. Advanced ensemble and deep-learning techniques can also be explored. Future research should further examine real-world deployment, computational efficiency, and continuous detection of newly emerging Android malware.

Funding: Funding information was not supplied in the manuscript.

Conflicts of Interest: A conflict-of-interest declaration was not supplied in the manuscript.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

- Ababneh, M., Al-Droos, A., & El-Hassan, A. (2024). Modern Mobile Malware Detection Framework Using Machine Learning and Random Forest Algorithm. *Comput. Syst. Sci. Eng.*, 48. <https://doi.org/10.32604/csse.2024.052875>
- Akhtar, M. S., & Feng, T. (2023). Evaluation of Machine Learning Algorithms for Malware Detection. *Sensors*, 23(2). <https://doi.org/10.3390/s23020946>
- Al Shihab, M. F., Hasan, M. M., Nadim, R., Bormon, M., & Ahmed Al Seum, S. (2025). Fingerprinting Malware Behavior: A Deep Learning Approach. In *Proceedings of the 12th International Conference on Next Generation Computing, Communication, Systems and Security, NSysS* (pp. 86–91). <https://doi.org/10.1145/3777555.3777573>
- Alhogail, A., & Alharbi, R. A. (2025). Effective ML-Based Android Malware Detection and Categorization. *Electron*, 14(8). <https://doi.org/10.3390/electronics14081486>
- Alkahtani, H., & Aldhyani, T. H. H. (2022). Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices. *Sensors*, 22(6). <https://doi.org/10.3390/s22062268>
- Arveti, S., Kalal, M., Bose, P., Yeruva, L. A., Inamdar, S., & Kanneganti, R. K. (2026). Model Driven and Modular Software Architecture for Scalable Development. In *International Conference on Computing Theory and Wireless Communications (ICCTWC)* (pp. 1–10). IEEE. <https://doi.org/10.1109/ICCTWC68241.2026.11583718>
- Azeem, M., Khan, D., Iftikhar, S., Bawazeer, S., & Alzahrani, M. (2024). Analyzing and comparing the effectiveness of malware detection: A study of machine learning approaches. *Heliyon*, 10(1). <https://doi.org/10.1016/j.heliyon.2023.e23574>
- Bokolo, B., Jinad, R., & Liu, Q. (2023). A Comparison Study to Detect Malware using Deep Learning and Machine learning Techniques. In *6th International Conference on Big Data and Artificial Intelligence, BDAI*. <https://doi.org/10.1109/BDAI59165.2023.10256957>

- Govindarajan, P., Babu, S., Khan, A. R., & G. S. (2026). Android Malware Detection Using Unified Evolutionary Multi-Domain Cybersecurity Detection Architecture. In 13th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1–6). IEEE. <https://doi.org/10.23919/INDIACom70271.2026.11526658>
- Kurniawan, F., Stiawan, D., Antoni, D., Yazid Idris, M., & Budiarto, R. (2025). A Robust and Efficient Machine Learning Framework for Enhancing Early Detection of Android Malware. *IEEE Access*, 13, 127183–127220. <https://doi.org/10.1109/ACCESS.2025.3589656>
- Lingam, R. (2025). AI-powered framework for evaluating child-friendly mobile applications. *Int. J. Hum. Comput. Intell*, 4(4). <https://doi.org/10.5281/zenodo.15624307>
- Liu, T., Li, Z., Long, H., & Bilal, A. (2023). NT-GNN: Network Traffic Graph for 5G Mobile IoT Android Malware Detection. *Electron*, 12(4). <https://doi.org/10.3390/electronics12040789>
- M. U. Rashid et al (2025). Hybrid Android Malware Detection and Classification Using Deep Neural Networks. *Int. J. Comput. Intell. Syst*, 18. <https://doi.org/10.1007/s44196-025-00783-x>
- Maghanaki, M., Keramati, S., Chen, F. F., & Shahin, M. (2026). Systematic Evaluation of Machine Learning and Deep Learning Models for IoT Malware Detection Across Ransomware, Rootkit, Spyware, Trojan, Botnet, Worm, Virus, and Keylogger. *Sensors*. <https://doi.org/10.3390/s26061750>
- N. Prathapaneni et al (2024). Dynamic Behaviour analysis and interpretation of Malware in Android devices using Ensemble Machine Learning. In 3rd International Conference on Artificial Intelligence for Internet of Things, AllIoT. <https://doi.org/10.1109/AllIoT58432.2024.10574581>
- Panja, S., Mondal, S., Nag, A., Prakash Singh, J., Jyoti Saikia, M., & Kumar Barman, A. (2025). An Efficient Malware Detection Approach Based on Machine Learning Feature Influence Techniques for Resource-Constrained Devices. *IEEE Access*, 13. <https://doi.org/10.1109/ACCESS.2025.3526878>
- Patel, P. (2026). Automated Detection of Insider Threats in Zero Trust Networks Through Machine Learning Techniques for Improved Security. In 2nd International Conference on Artificial Intelligence Systems (AIS) (pp. 460–465). IEEE. <https://doi.org/10.1109/AIS69919.2026.11621584>
- Rane, S., Yadav, S., Hambir, Y., & Gupta, A. (2024). AI-Powered Malware Detection: Leveraging Machine Learning for Enhanced Cybersecurity. *Nanotechnol. Perceptions*, 20(S9), 1331–1347. <https://doi.org/10.62441/nano-ntp.v20iS9.107>
- Sethi, K., Kumar, R., Sethi, L., Bera, P., & Patra, P. K. (2019). A novel machine learning based malware detection and classification framework. In International Conference on Cyber Security and Protection of Digital Services, Cyber Security. <https://doi.org/10.1109/CyberSecPODS.2019.8885196>
- Srirangam, A., Rishik, C., Singh, P., & Kumar, J. (2026). Image-Based Malware Detection Using Convolutional Neural Networks. In 1st International Conference on Artificial Intelligence & Machine Learning in Communication and Power Systems (AIMLCPS) (pp. 1–5). IEEE. <https://doi.org/10.1109/AIMLCPS68702.2026.11542468>
- Tiwari, S. (2021). Android Malware Dataset for Machine Learning. Kaggle.
- Vaghasiya, S., Gupta, R., Mohapatra, S., Tanwar, S., & Garg, D. (2024). ML-Orchestrated Malware Detection Framework for Windows APIs in Critical Infrastructure Systems. In 2nd International Conference on Computer, Communication and Control, IC4. <https://doi.org/10.1109/IC457434.2024.10486344>