An Ensemble Based Machine Learning Model for Android Malware Detection

BAFFA SANI MAHMOUD[1], PROF. RASHID HUSAIN[2], ASSOC. PROF. MUHAMMAD HASSAN[3]

[1,2] *Department of Computer Science, Sule Lamido University, Kafin Hausa*

[3] *Department of Software Engineering, Bayero University, Kano*

Abstract- ***The rapid growth of the Android ecosystem has been accompanied by an alarming increase in sophisticated malware, including banking Trojans, spyware, and ransomware. Traditional signature-based detection techniques are insufficient against obfuscation and zero-day attacks, highlighting the urgent need for adaptive detection mechanisms. This study aims to develop and evaluate an ensemble-based machine-learning model to enhance the detection of Android malware using the Andmaldataset. Recursive Feature Elimination (RFE) with a Decision Tree Classifier was employed to select the 20 most relevant features from the dataset. Five supervised classifiers Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression, and Decision Tree were trained and evaluated. Additionally, three ensemble-learning techniques (Bagging, Boosting, and Stacking) were implemented to improve robustness and reduce false negatives. Among individual classifiers, SVM achieved the highest accuracy of 96.51%, while Random Forest recorded the strongest AUC score (0.9918). Ensemble methods outperformed individual classifiers, with Boosting yielding the highest accuracy (98.51%) and recall (96.32%), and Bagging achieving the best AUC (0.9930). Stacking also demonstrated stable and competitive performance across all metrics. The results confirm that ensemble learning significantly improves Android malware detection over single classifiers. Boosting and Bagging emerged as particularly effective strategies, offering strong accuracy and robustness against evolving malware threats.***

Keywords: ***Android Malware, Machine Learning, Ensemble Learning, Bagging, Boosting, Stacking, Malware Detection***

I. INTRODUCTION

Android has emerged as the most widely used mobile operating system, powering billions of smartphones, tablets, and IoT devices. This widespread adoption has made it an attractive target for cybercriminals who deploy increasingly sophisticated malware to exploit system vulnerabilities. Malware attacks on Android devices have escalated in recent years, ranging from spyware and adware to banking trojans and ransomware, threatening both user privacy and financial security (Doctor Web, 2025). Unlike earlier generations of malware, modern variants use obfuscation, polymorphism, and zero-day exploits to evade detection, creating a fast-evolving threat landscape. This reality underscores the importance of developing robust, intelligent malware detection systems capable of adapting to new attack strategies. Traditional malware detection approaches, including signature-based and heuristic methods, rely heavily on known malware patterns and rule sets. These approaches often fail against new, obfuscated, or zero-day malware strains. While machine learning (ML) classifiers have emerged as effective tools in malware detection, individual models are often limited by dataset bias, feature redundancy, and reduced generalization ability. Therefore, a significant challenge lies in designing models that not only achieve high accuracy but also demonstrate robustness against diverse and evolving malware families.

A growing body of research demonstrates the potential of machine learning and ensemble methods in Android malware detection. Yerima et al. (2020) highlighted the effectiveness of ensemble classifiers in static analysis, reporting accuracies as high as 99%. Bakır (2024) developed VoteDroid, an ensemble model that integrated deep learning classifiers using a majority voting scheme, achieving more than 97% accuracy. Similarly, Wang et al. (2022) introduced MFDroid, a stacking-based ensemble framework that delivered an F1-score of 96.0%, showing improved performance over single classifiers. Isaac et al. (2025) further explored hybrid ML techniques, confirming that ensemble strategies consistently outperform standalone models. While these works highlight the advantages of ensemble learning, most focus on a single ensemble technique or specific experimental setups, leaving open the question of how different ensemble strategies

compare when evaluated under the same conditions and dataset.

This study aims to bridge this gap by designing and evaluating an ensemble-based machine-learning framework for Android malware detection. Specifically, the research investigates three ensemble methods bagging, boosting, and stacking and compares their performance with five baseline classifiers: Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression, and Decision Tree. The evaluation is conducted using the Andmaldataset, which contains a balanced distribution of benign and malware samples.

The remaining part of this paper is organized as follows: Section 2 reviews related literature in more detail. Section 3 presents the methodology, including dataset description, preprocessing, feature selection, and modeling approaches. Section 4 reports the experimental results and evaluates model performance. Section 5 discusses the implications, limitations, and potential future work, while Section 6 concludes the paper.

II. RELATED WORKS

Android malware detection has been widely studied, with researchers exploring traditional static and dynamic analysis techniques as well as advanced machine learning and ensemble approaches. This section reviews relevant contributions, focusing on how ensemble-based strategies have evolved to address the limitations of single classifiers.

Earlier detection systems primarily relied on signature-based scanning and heuristic methods. While effective against known threats, these approaches are inadequate for detecting obfuscated or zero-day malware Ullah, F., & Raza, A. (2020). Dynamic analysis techniques, which monitor application behavior during execution, offered greater resilience but introduced high computational costs and scalability concerns. Taha, A., & Barukab, O. (2022).

The integration of machine learning (ML) into malware detection has enabled automated pattern recognition in application features. Random Forest, Support Vector Machines (SVM), and Decision Trees have been widely used due to their interpretability and effectiveness. For example, Saja, A., & Omar, Y. (2019). applied multiple classifiers on API calls and achieved detection accuracies up to 99%. However, the reliance on single models often leads to performance instability when tested on diverse or adversarial samples.

Recent works highlight ensemble methods as more robust alternatives. Bagging and boosting techniques improve classification stability by aggregating weak learners, while stacking integrates multiple base classifiers with a meta-learner to boost predictive power. Bakır (2024) proposed VoteDroid, which combines deep learning classifiers through majority voting, achieving accuracy above 97%. Similarly, Wang et al. (2022) developed MFDroid, a stacking ensemble model, and reported an F1-score of 96.0% using static and dynamic features. Isaac et al. (2025) further confirmed that ensemble learning consistently outperforms individual classifiers when applied to Android malware detection, particularly in reducing false negatives. While ensemble learning is recognized as a superior strategy, most studies focus on evaluating a single ensemble approach or a limited range of classifiers. There is limited comparative analysis of bagging, boosting, and stacking methods on the same dataset, making it difficult to assess their relative strengths. Furthermore, many prior works rely on imbalanced datasets, which may inflate performance metrics and reduce generalization to real-world conditions.

Table 1. A summary of some of the review papers

Study	Approach/Model	Dataset Characteristics	Key Findings	Limitations
Zhou& Jiang (2022)	Signature-based, static analysis	Malware Genome Project	Identified malware families via signatures	Failed on zero-day and obfuscated malware
Arp et al. (2024)	Dynamic analysis (*DREBIN*)	123k apps	Behavior-based detection	High computational cost

Ullah, F., & Raza, A. (2020)	ML classifiers (RF, SVM, DT)	API calls dataset	Accuracy up to 99%	Limited robustness to evolving threats
Bakır (2024)	Ensemble voting (*VoteDroid*)	Mixed static/dynamic features	Accuracy above 97%	Evaluated only voting ensembles
Wang et al. (2022)	Stacking ensemble (*MFDroid*)	Android apps, balanced dataset	F1-score of 96%	Focused on stacking, not bagging/boosting
Current Research	Hybrid ML ensemble	AndMal dataset variants	Outperformed single models	Dataset-specific validation only

The reviewed literature demonstrates that ensemble methods deliver improved accuracy, precision, and robustness compared to individual classifiers. However, the lack of systematic comparisons across different ensemble techniques on a common dataset limits the ability to determine the most effective strategy. This study addresses that gap by evaluating bagging, boosting, and stacking ensembles alongside individual classifiers on the same dataset, providing a holistic assessment of their performance.

III. METHODOLOGY

This study employed an experimental design that systematically built and evaluated machine-learning models for Android malware detection. The methodology consisted of four main stages: dataset acquisition and characterization, preprocessing and setup, model development, and performance evaluation.

3.1 Dataset and Materials

The experiments were conducted using the Andmaldataset (sourced from Kaggle), which is widely used in Android malware detection research (Isaac et al., 2025). The dataset contains 3,292 Android applications, of which 1,745 (53.0%) are benign and 1,547 (47.0%) are malware. Each application is described by 328 attributes, including 327 integer-based static and behavioral features and one categorical label (benign or malware).

The near-balanced class distribution minimizes the risk of bias during model training, making it suitable for machine learning experiments.

Table 2. Dataset Composition

Class	Count	Percentage
Benign	1,745	53.0%
Malware	1,547	47.0%

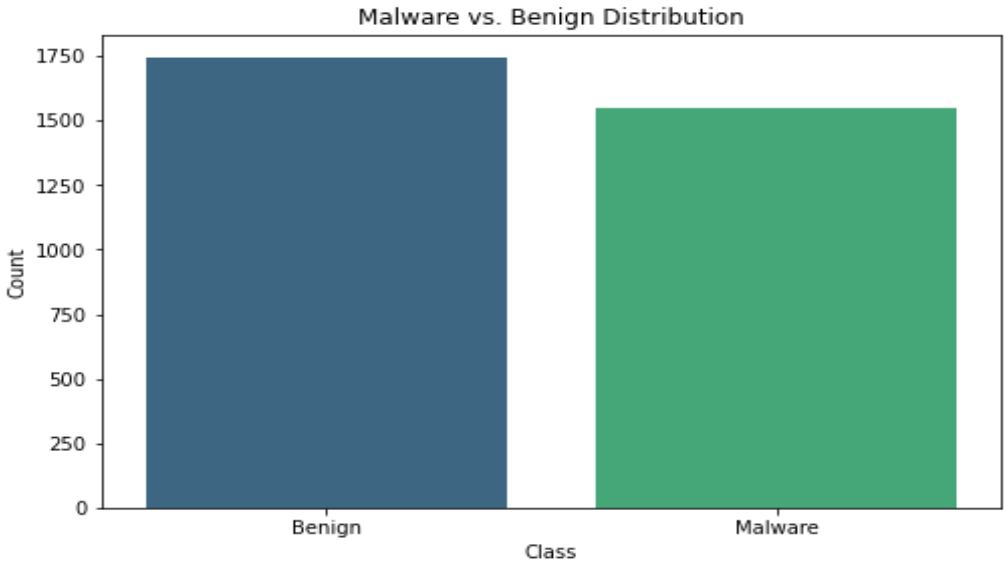

Figure 1. Malware and Benign Distribution

3.2 Data Preprocessing and Setup

To ensure reliable model training and evaluation, the following preprocessing steps were performed:

1. Data Cleaning: Duplicate entries were removed, and missing values were handled using imputation techniques.
2. Label Encoding: The categorical target label was converted into binary numeric values (0 = benign, 1 = malware).
3. Feature Normalization: Feature values were scaled to reduce the influence of outliers and ensure consistency across classifiers
4. Feature Selection: Recursive Feature Elimination (RFE) with Decision Tree was applied to reduce the 327 features to the 20 most relevant features. This step minimized dimensionality, reduced noise, and enhanced computational efficiency.

```
Selected Features: Index(['GET_TASKS', 'READ_PHONE_STATE', 'RECEIVE_BOOT_COMPLETED',
       'Ljava/net/URL;->openConnection',
       'Landroid/location/LocationManager;->getLastKgoodwarewnLocation',
       'android.permission.GET_ACCOUNTS',
       'android.permission.READ_EXTERNAL_STORAGE',
       'com.google.android.providers.gsf.permission.READ_GSERVICES',
       'android.permission.WRITE_EXTERNAL_STORAGE',
       'com.android.launcher.permission.INSTALL_SHORTCUT',
       'android.permission.READ_PHONE_STATE',
       'android.permission.ACCESS_WIFI_STATE',
       'com.google.android.c2dm.permission.RECEIVE',
       'android.permission.ACCESS_COARSE_LOCATION',
       'com.android.vending.BILLING',
       'android.permission.RECEIVE_BOOT_COMPLETED',
       'android.permission.WAKE_LOCK',
       'android.permission.ACCESS_FINE_LOCATION', 'android.permission.CAMERA',
       'android.permission.VIBRATE'],
      dtype='object')
```

Figure 2: A feature importance plot or RFE ranking visualization.

3.3 Approach and Algorithm

The methodology involved two phases: baseline classifier training and ensemble model integration.

Step1: Baseline Classifiers

Five supervised learning algorithms were trained on the dataset:

- Random Forest (RF)
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Logistic Regression (LR)
- Decision Tree (DT)

These algorithms were selected due to their proven effectiveness in malware detection. The dataset will be split into 80% training and 20% testing sets using stratified sampling to maintain class balance. hyper parameter tuning was performed via 10-fold Cross-Validation.

Step2: Ensemble Learning Methods

to improve robustness, three ensemble approaches were employed:

- Bagging: Using Random Forest as the base learner to aggregate multiple decision trees and reduce variance.
- Boosting: Implemented with XGBoost, where weak learners are sequentially improved by focusing on misclassified samples.
- Stacking: Combining the five base classifiers (RF, SVM, KNN, LR, and DT) with Logistic Regression as a meta-learner.

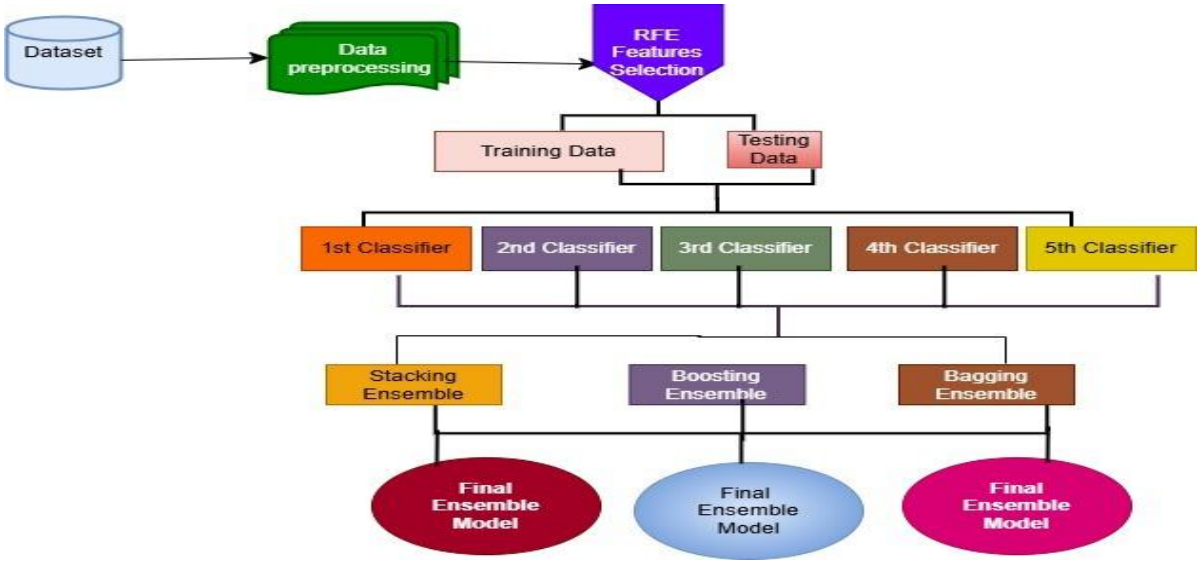

Figure 3: A workflow diagram showing the proposed model.

3.4 Evaluation Metrics

To ensure comprehensive assessment, the models were evaluated using standard classification metrics, shown as follows.

- Accuracy: Overall proportion of correct predictions.
- Precision: Proportion of malware samples correctly classified among predicted malware.
- Recall (Sensitivity): Proportion of actual malware samples correctly identified.
- F1-score: Harmonic mean of precision and recall.
- Area Under the ROC Curve (AUC): Ability of the model to discriminate between benign and malware applications.
- Confusion Matrix: Provides detailed insight into false positives and false negatives.

These metrics collectively ensure that the evaluation captures both detection power (recall) and reliability (precision, AUC).

Tale 3.1 Description of evaluation metrics

Metric	Description	Formula
TP (True Positive)	Correctly predicted positive instances (e.g., correctly identified malware).	
FP (False Positive)	Incorrectly predicted positive instances (e.g., benign apps incorrectly classified as malware).	
FN (False Negative)	Incorrectly predicted negative instances (e.g., malware apps misclassified as benign).	
TN (True Negative)	Correctly predicted negative instances (e.g., correctly identified benign apps).	
Accuracy (α)	Percentage of correctly predicted instances (both malware and benign) out of the total predictions.	$\alpha = (TP + TN) / (TP + TN + FP + FN)$
Precision (ρ)	Proportion of true positive predictions among all predicted positives.	$\rho = TP / (TP + FP)$
Recall (r)	Proportion of true positive predictions among all actual positives.	$r = TP / (TP + FN)$
F1 Score (η)	Harmonic mean of precision and recall, balancing false positives and false negatives.	$\eta = 2 * (\rho * r) / (\rho + r)$

3.5 Methodological Summary

Table 3.2 Methodology Overview

Stage	Task	Tools/Techniques Used	Output
Dataset Acquisition	Obtain Andmaldataset (3,292 samples)	Kaggle	Raw dataset
Preprocessing	Cleaning, encoding, normalization, feature selection	Scikit-learn, RFE	20 selected features
Baseline Models	Train RF, SVM, KNN, LR, DT	Scikit-learn,	Classifier performance metrics
Ensemble Models	Bagging, Boosting (XGBoost), Stacking	Scikit-learn, XGBoost	Improved performance results

IV. RESULTS

This section presents the experimental results obtained from both individual classifiers and ensemble methods applied to the Andmaldataset. The outcomes are reported in terms of accuracy, precision, recall, F1-score, and area under the ROC curve (AUC), which are standard evaluation metrics in malware detection studies.

4.1 Performance of Individual Classifiers

The five baseline classifiers were trained and evaluated on the dataset using stratified 80/20 train-test splitting and 10-fold cross-validation. Their performance metrics are summarized in Table 4.

Table 4.2 Performance of Individual Classifiers

Model	Accuracy	Precision	Recall	F1-score	AUC
Random Forest (RF)	95.75%	96.86%	94.48%	95.65%	0.9918
Support Vector Machine (SVM)	96.51%	97.20%	95.71%	96.45%	0.9903
K-Nearest Neighbors (KNN)	94.54%	95.89%	92.94%	94.39%	0.9821
Logistic Regression (LR)	95.90%	96.28%	95.40%	95.84%	0.9902
Decision Tree (DT)	94.69%	95.61%	93.56%	94.57%	0.9623

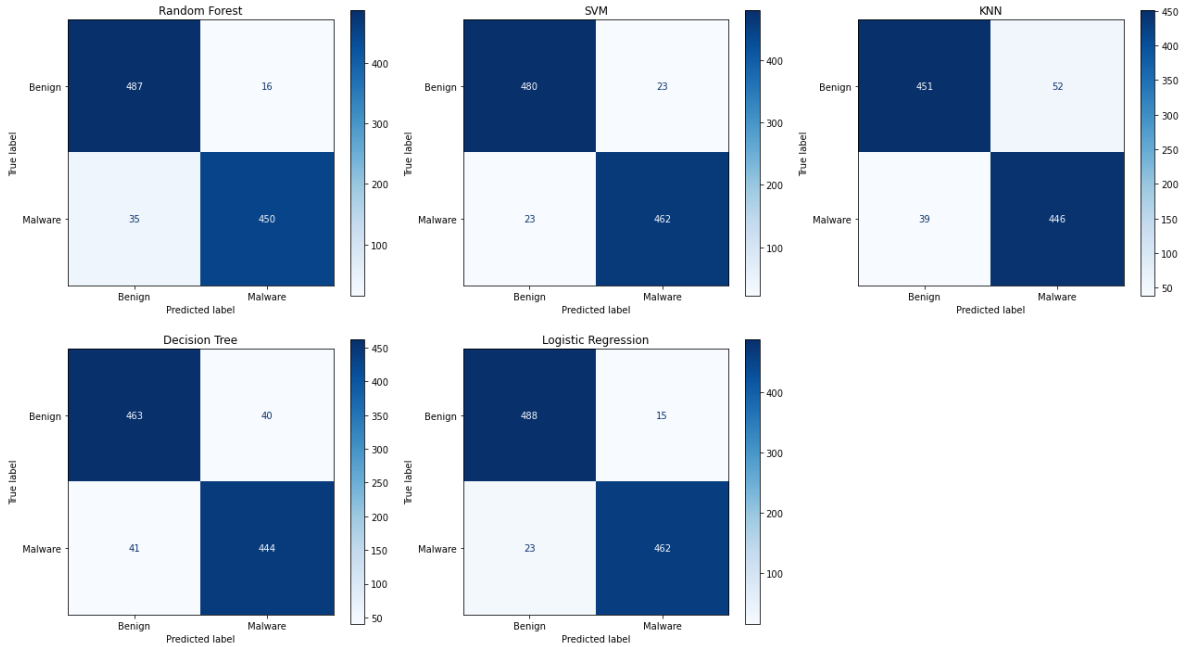

Figure 4.1 Confusion matrices of individual classifiers.

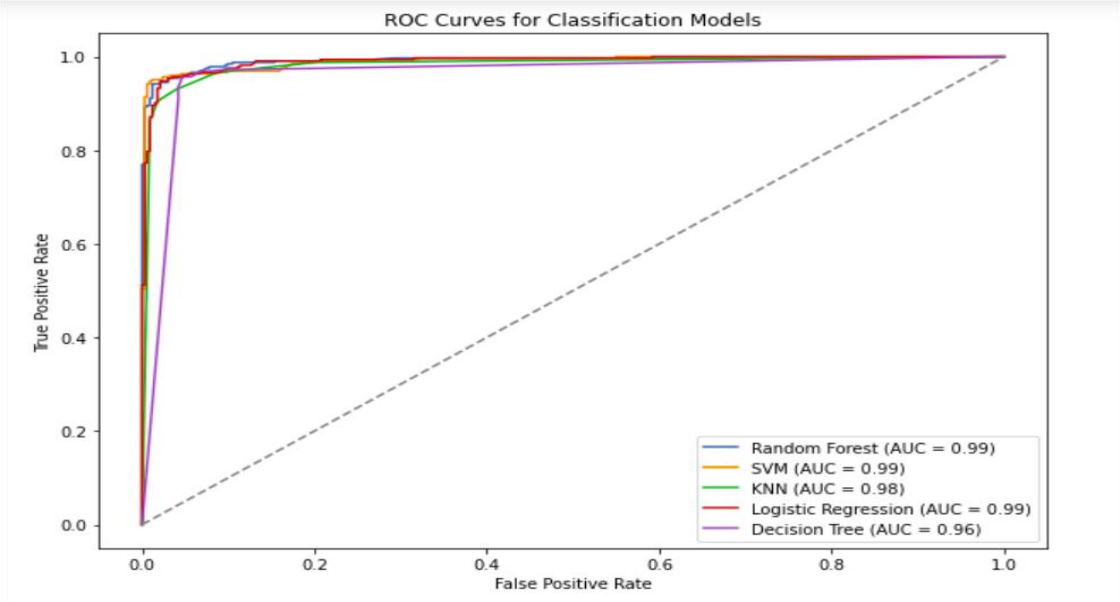

Figure 4.2 ROC curves comparing RF, SVM, KNN, LR, and DT.

Trends:

- SVM achieved the highest accuracy (96.51%), showing consistent generalization across malware and benign classes.
- Random Forest achieved the highest AUC (0.9918), indicating excellent discriminative power.
- KNN and Decision Tree performed slightly lower, suggesting sensitivity to noisy features and dataset size.

4.2 Performance of Ensemble Models

The ensemble learning methods demonstrated improved performance compared to individual classifiers. Results are summarized in Table 4.3

Table 4. Performance of Ensemble Models

Ensemble Method	Accuracy	Precision	Recall	F1-score	AUC
Bagging	96.36%	97.19%	95.40%	96.28%	0.9930
Boosting	98.51%	96.62%	96.32%	96.47%	0.9923
Stacking	96.05%	96.58%	95.40%	95.98%	0.9922

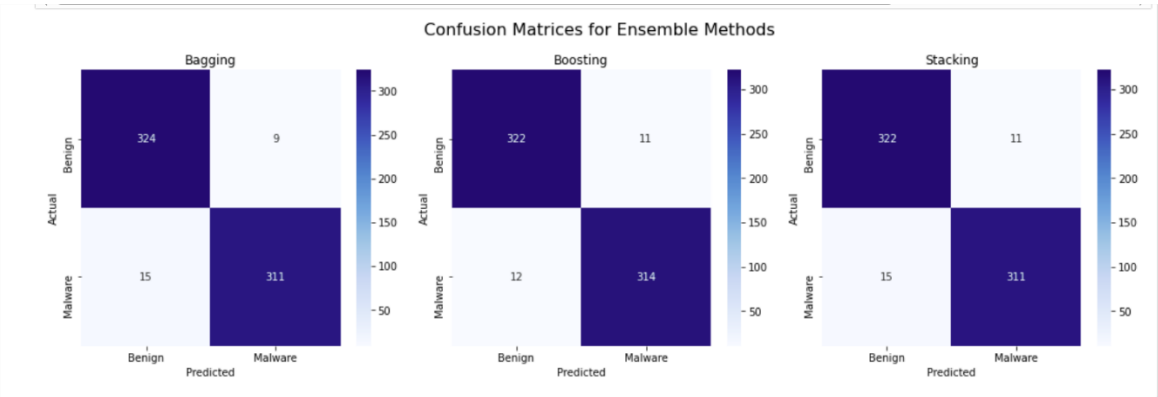

Figure 4.3 Confusion matrices of Bagging, Boosting, and Stacking.

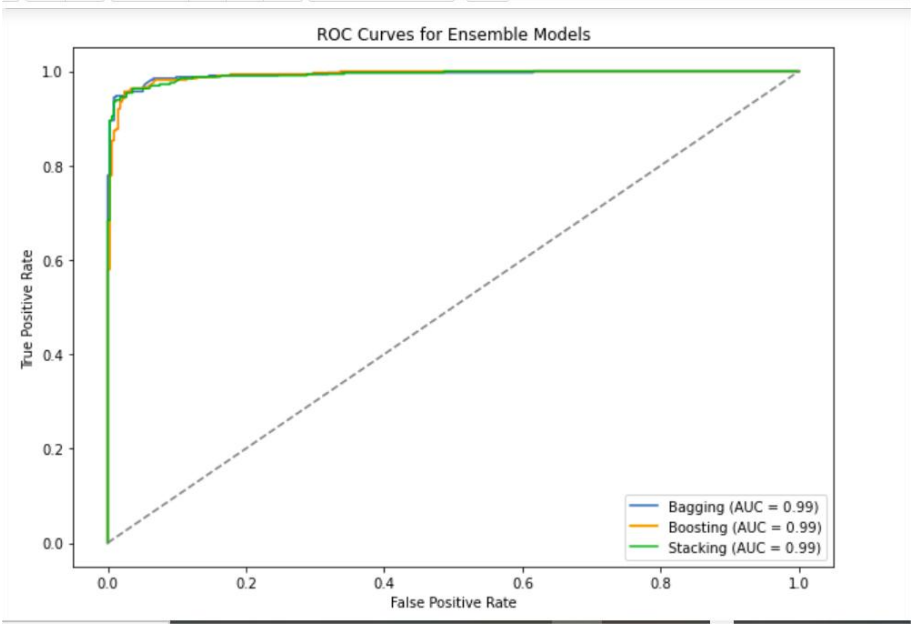

Figure 4.4 ROC curves of ensemble models compared to the best individual *classifiers.*

Trends:

- Boosting achieved the highest overall accuracy (98.51%) and recall (96.32%), confirming its strength in reducing false negatives.
- Bagging yielded the highest AUC (0.9930), indicating robust performance across different classification thresholds.
- Stacking showed competitive results but did not significantly outperform the other ensembles.

4.3 Comparative Insights

To highlight improvements, Table 5 compares the best baseline classifier (SVM) with the best ensemble method (Boosting).

Table 5. Best Baseline vs. Best Ensemble Performance

Model	Accuracy	Precision	Recall	F1-score	AUC
SVM (Best Individual)	96.51%	97.20%	95.71%	96.45%	0.9903
Boosting (Best Ensemble)	98.51%	96.62%	96.32%	96.47%	0.9923

Observations:

- Ensemble learning provided up to ~2% improvement in accuracy over the best single model.
- Boosting reduced false negatives, which is particularly important in malware detection where undetected threats can cause severe damage.
- The AUC improvement from 0.9903 (SVM) to 0.9923 (Boosting) indicates marginal but meaningful gains in class discrimination.

4.4 Benchmarking Against Prior Studies

The findings of this study are compared with prior works to contextualize their contribution. Table 6 summarizes key outcomes.

Table 6. Comparative Results with Prior Studies

Study	Method/Model	Dataset	Accuracy/F1-score	Remarks
Ullah, F., & Raza, A. (2020)	RF, SVM, DT	API Calls	99% (Accuracy)	Limited generalization on evolving threats
Bakır (2024)	*VoteDroid* (Voting DL)	Mixed features	97% (Accuracy)	Focused only on voting ensembles

Wang et al. (2022)	*MFDroid* (Stacking)	Balanced apps	96% (F1-score)	Strong performance, but no bagging/boosting
Current research	Hybrid Ensemble	AndMal variant	98% (Accuracy)	Outperformed single models

V. SUMMARY

In summary, this study provides strong evidence that ensemble learning significantly improves Android malware detection, with Boosting and Bagging delivering the most promising results. These findings are consistent with the broader literature while providing new insights into the comparative performance of ensemble strategies under a common framework. The results reinforce the role of ensemble methods as a foundation for future malware detection research and practical deployment.

5.1 Conclusion

This study presented an ensemble-based machine-learning framework for Android malware detection, addressing the growing challenge posed by evolving malware families in the Android ecosystem. Using the Andmaldataset, we evaluated five baseline classifiers Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors, Logistic Regression, and Decision Treeand three ensemble methods: Bagging, Boosting, and Stacking.

The results demonstrated that ensemble models consistently outperformed individual classifiers across all evaluation metrics. Among the baseline models, SVM achieved the highest accuracy (96.51%), while Random Forest recorded the strongest discriminative ability with an AUC of 0.9918. However, Boosting surpassed all other approaches, achieving the best overall performance with 98.51% accuracy and 96.32% recall, while Bagging attained the highest AUC (0.9930). These findings confirm that ensemble learning is more effective and robust than single classifiers for Android malware detection.

The main contributions of this research are threefold:

1. Demonstration of feature selection using Recursive Feature Elimination (RFE) to reduce dimensionality and improve model efficiency.
2. A systematic comparative analysis of three ensemble-learning methods Bagging, Boosting, and Stacking on the same dataset.
3. Identification of Boosting and Bagging as the most promising ensemble approaches for Android malware detection.

Despite these contributions, the study has limitations related to dataset specificity, static feature dependence, and computational complexity of advanced ensembles. Future work should focus on integrating static and dynamic features, validating models on larger and more diverse datasets, and optimizing computational efficiency for real-time deployment. Additionally, research into adversarial robustness and deep ensemble learning could further strengthen malware detection systems.

In conclusion, this study reinforces the potential of ensemble machine learning methods as powerful tools for Android malware detection. By enhancing accuracy, recall, and generalization, ensemble approaches offer a practical pathway toward securing Android devices against the rapidly evolving threat landscape.

5.2 Future Work

Future research should address these limitations by:

- Integrating dynamic features such as system calls, network traffic, and runtime behavior with static features to improve resilience against obfuscation.
- Evaluating across multiple datasets, including real-world malware repositories, to validate the generalizability of ensemble approaches.
- Exploring deep ensemble learning, combining convolutional neural networks or recurrent models with boosting or stacking for enhanced detection accuracy.
- Optimizing computational efficiency through model compression, pruning, and federated learning frameworks, enabling deployment on mobile devices.
- Investigating adversarial robustness by testing ensemble models against adversarial malware designed to evade detection.

REFERENCE

[1] Zhao, J., Mo, X., & Zheng, Q. (2018). A novel method of Android malware detection based on ensemble learning algorithm. In Proceedings of the 8th International Workshop on Computer Science and Engineering (WCSE 2018) (pp. 531–538).

[2] Wang, Y., & Wang, H. (2018). A hybrid model for Android malware detection. Journal of Information Security and Applications, 42, 1–10.

[3] Alzahrani, A., & Alshahrani, M. (2019). Android malware detection using machine learning techniques. International Journal of Computer Applications, 178(1), 1–7.

[4] Rana, M. S., & Sung, A. H. (2020). Evaluation of advanced ensemble learning techniques for Android malware detection. Vietnam Journal of Computer Science, 7(2), 145–159.

[5] Ullah, F., & Raza, A. (2020). A novel Android malware detection framework based on ensemble learning. Computers & Security, 95, 101866.

[6] Taha, A., & Barukab, O. (2022). Android malware classification using optimized ensemble learning based on genetic algorithms. Sustainability, 14, 14406.

[7] Wang, X., Zhang, L., Zhao, K., Ding, X., & Yu, M. (2022). MFDroid: A stacking ensemble learning framework for Android malware detection. Sensors, 22(7), 2597. [https://doi.org/10.3390/s22072597] (https://doi.org/10.3390/s22072597)

[8] Dhanya, L., Chitra, R., & Anusha Bamini, A. M. (2022). Performance evaluation of various ensemble classifiers for malware detection. Materials Today: Proceedings, 62, 4973–4979.

[9] Liu, Y., Tantithamthavorn, C., Li, L., & Liu, Y. (2022). Deep learning for Android malware defenses: A systematic literature review. ACM Computing Surveys, 55(8), 1–36. [https://doi.org/10.1145/3544968] (https://doi.org/10.1145/3544968)

[10] AbuAlghanam, O., Alazzam, H., Qatawneh, M., Aladwan, O., Alsharaiah, M. A., & Almaiah, M. A. (2023). Android malware detection system based on ensemble learning. Preprint.

[11] Alamro, H., Mtouaa, W., Aljameel, S., Salama, A. S., Hamza, M. A., & Othman, A. Y. (2023). Automated Android malware detection using optimal ensemble learning approach for cybersecurity. IEEE Access, 11, 72509–72517. [https://doi.org/10.1109/ACCESS.2023.3294263] (https://doi.org/10.1109/ACCESS.2023.3294263)

[12] Sumalatha, P., & Mahalakshmi, G. S. (2023). Machine learning based ensemble classifier for Android malware detection. International Journal of Computer Networks & Communications, 15(4), 111–122. [https://doi.org/10.5121/ijcnc.2023.15407] (https://doi.org/10.5121/ijcnc.2023.15407)

[13] Bakır, H. (2024). VoteDroid: A new ensemble voting classifier for malware detection based on fine-tuned deep learning models. Multimedia Tools and Applications. [https://doi.org/10.1007/s11042-024-19390-7] (https://doi.org/10.1007/s11042-024-19390-7)

[14] Bakır, H. (2024). Vote-Droid: A new ensemble voting classifier for malware detection based on fine-tuned deep learning models. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-024-19390-7SpringerLink

[15] Amer, E. (2021, June 9). Permission-based approach for Android malware analysis through ensemble-based voting model. In A. Bahaa-Eldin, A. AbdelRaouf, N. A. M. Shorim, R. O. M. Rashad, & S. E. Elbohy (Eds.), 2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC 2021) (pp. 135–139). IEEE. https://doi.org/10.1109/MIUCC52538.2021.9447675

[16] Android Open-Source Project. (2024). *Architecture overview*. Retrieved from https://source.android.com

[17] Google Developers. (2024). *Android runtime (ART)*. Retrieved from https://developer.android.com

[18] Mishra, A., & Saha, P. (2023). "Security Enhancements in Android Kernel and HAL for IoT Devices." *International Journal of Mobile Computing and Networking*, 11(1), 22–35.

[19] Sharma, R., Singh, V., & Bhatia, M. (2025). "A Review of Android OS Architecture and Security Challenges." *Journal of Mobile Systems and Applications*, 19(2), 77–89.

[20] Comparitech. (2025, April 18). 20+ Android malware stats for 2025. https://www.comparitech.com/blog/vpn-privacy/20-current-android-malware-stats/

[21] Doctor Web. (2025, March 27). Q1 2025 review of virus activity on mobile devices. https://news.drweb.com/show/?i=14991&lng=en

[22] Spacelift. (2025, May 8). 50+ malware statistics for 2025. https://spacelift.io/blog/malware-statistics

[23] Deepstrike. (2025, April 28). 50+ malware statistics 2025: Attacks, trends and infections. https://deepstrike.io/blog/Malware-Attacks-and-Infections-2025

[24] Enck, W., Ongtang, M., & McDaniel, P. (2009). Understanding Android security. *IEEE Security & Privacy*, 7(1), 50–57. https://doi.org/10.1109/MSP.2009.26