Ensemble Learning for Multi-Class Android Malware Detection: A Robust Framework for Family Level Classification

Mana Saleh Al Reshan
Department of Information System-College of Computer Science and Information Systems, Najran University,
Najran 61441, Saudi Arabia
Emerging Trends and Technologies Lab (ETRL)-College of Computer Science and Information Systems, Najran University,
Najran 61441, Saudi Arabia

Abstract—**The widespread popularity of Android devices has made them a prime target for sophisticated and evolving malware threats. Traditional malware detection techniques rely on binary classification (malicious vs. benign), which fails to capture the nuanced behavioral differences between malware families, critical for threat intelligence and incident response. To address this limitation, we propose a robust multi-class classification approach for Android malware family detection, leveraging ensemble learning and advanced feature selection methods. Our system uses a hybrid feature extraction strategy that combines Chi-Squared and Mutual Information techniques to eliminate low-utility features and retain the most discriminative attributes. These include flow-based metrics, inter-arrival time (IAT), and session duration, key indicators of malicious behavior. We evaluated five baseline classifiers (Random Forest, Gradient Boosting, XGBoost, Extra Trees, and Decision Trees) across three ensemble strategies (bagging, voting, and stacking). Among these, the Stacking ensemble achieved the highest overall performance, with 83% across all evaluation metrics, accuracy, precision, recall, and F1-score, and a True Negative Rate (TNR) of 93.34%. The framework also improves the detection of minority malware families in imbalanced datasets. These findings highlight the advantages of ensemble learning for building scalable and reliable Android systems suitable for real-world deployment.**

Keywords—Malware detection; cyber threat; ML models; feature selection; ensemble methods

I. Introduction

The rapid proliferation of intelligent mobile devices has established Android as the dominant smartphone computing platform. In 2020, global smartphone production reached approximately 1.24 billion units, with Android accounting for around 78.45% of the market share [1]. This dominance can be attributed to its open-source architecture, affordability, and extensive connectivity options, including Wi-Fi, Bluetooth, and GPS. Additionally, Android's support for third-party application installation significantly enhances device functionality.

However, this openness introduces critical security vulnerabilities. Unlike Apple's tightly controlled ecosystem, Android's decentralized app distribution model facilitates the spread of malicious applications beyond the official Google Play Store. Despite Google's integrated malware scanning systems, many harmful apps still manage to evade detection [2]. The fragmented nature of the Android ecosystem further compounds security challenges, increasing exposure to threats such as data breaches and system-level attacks. In the third quarter of 2021, the Kaspersky Security Network reported over 9.5 million blocked incidents involving mobile malware, adware, and riskware. Common infection vectors include malicious MMS messages, phishing emails, and compromised applications. Users are particularly at risk when installing unverified apps that embed malicious code or exploit system vulnerabilities [3]. With Android powering approximately 70% of global mobile devices, it remains a prime target for various cyberattacks, including malware, ransomware, and spyware. Frequently used apps like WhatsApp, Facebook, mobile banking tools, and games are often exploited, placing user data such as photos, contacts, messages, and financial information at significant risk. This underscores the urgent need for intelligent and reliable malware detection systems to safeguard user privacy and device security.

Machine learning (ML), particularly ensemble learning techniques, has shown great promise in malware classification [4]. These methods enhance detection accuracy and offer robustness against evasion techniques. Ensemble techniques such as bagging, voting, and stacking combine the predictive power of multiple classifiers, mitigating the weaknesses of individual models and improving overall performance [5].

Unlike traditional single-classifier methods, such as Decision Trees or Gradient Boosting, which are prone to overfitting and poor generalization on large and varied malware datasets, our ensemble learning methodology provides tangible enhancements. For example, in a stacking ensemble research on the AndMal 2020 dataset, the authors obtained 83% accuracy and a TNR of 93.4% for multi-class classification, which beat traditional models hands down. Similarly, recent deep-learning-based ensembles combining CNN variants (ResNet, SENet, SEResNet) in "MFEMDroid" have also yielded superior detection performance owing to their complementary strengths. These results further substantiate that ensemble methods, specifically stacking, bagging, and voting, not only enhance detection precision and reduce false positives but also are more robust against evasive attacks. Further, in contrast to most existing research centering on binary classification only, our framework contributes more by allowing multi-family detection (Adware, Spyware, SMS_Malware, Benign), enabling richer threat identification and improved real-world applicability.

This research proposes and evaluates an Android malware detection framework that employs various ensemble learning methods to classify malware samples into families. The primary objective is to enhance detection reliability, reduce false positives, and bolster mobile cybersecurity.

Given the widespread use of Android smartphones, this study addresses a critical challenge in mobile security. By applying ensemble learning to classify malware into families, the proposed framework facilitates early threat identification and contributes to the development of intelligent, automated defense mechanisms. The findings are valuable to researchers, cybersecurity professionals, and mobile developers aiming to enhance the security of mobile environments.

Key Contributions:

- We obtained an Android malware dataset from the Kaggle repository. We utilized the Android Malware Dataset from Kaggle, consisting of 355,630 records with 85 features. Categorized into four families: Spyware, Adware, SMS_Malware, and Bengin.
- The dataset was preprocessed by handling null values, removing duplicates and irrelevant data, and applying label encoding to categorical features.
- Feature selection was performed using Chi-square and Mutual Information (MI) techniques to retain the most relevant features.
- We trained and evaluated five baseline machine learning classifiers: Random Forest (RF), Gradient Boosting (GB), Extreme Gradient Boosting (XGB), Extra Trees (ET), and Decision Tree (DT). In addition, we implemented and assessed three ensemble methods: stacking, bagging, and voting.
- This study presents a reproducible framework that can serve as a foundation for future research in mobile cybersecurity and intelligent malware detection.

The remainder of the study is structured as follows: Section II reviews related work on Android malware and Machine learning approach. Section III describes the dataset, preprocessing steps, and feature selection used in this study. Section IV presents the experimental results and provides comparative studies. Section V concludes the study and outlines the future direction.

II. Related Work

An algorithm for malware detection was proposed in this work [6]. Two Android malware datasets, including CICAndMal2017 and Drebin, were selected for conducting research. To detect malware attacks, the researchers implemented support vector machine (SVM), K-nearest neighbors (KNN), Linear Discriminant Analysis (LDA), Long Short Term Memory (LSTM), and Conventional Neural Network-Long Short Term Memory (CNN-LSTM). The highest accuracy rate, which is 100% was obtained by the SVM model using the CICAndMal2017 dataset.

This research conducted a systematic literature review to examine deep learning for malware attack detection. Their research spans 2014 to 2021, and they identified a total of 123 studies. They found 40.1% (53) of studies were deep learning-based. To fulfill their goal, some research questions were established [7].

Malicious pattern analysis can help reverse the misuse of online applications, such as banking and various social media apps. In study [8], an IOS malware categorization model was proposed, which is capable of detecting 30 patterns of phylogenetic. Out of 150mobile apps, only 7 apps were identified by the classifier.

Android Malware grayscale image representation visualization methods were proposed [9]. The authors utilize both global and local features for better performance. It was observed that local features such as manifest, dex-arc, produce poor accuracies between 65.16% and 93.56%. However, orb features take low computational time for feature selection, training extraction model training, and malware sample detection.

With the growing popularity of Android smartphones, malware threats are receiving increased attention. This study [10] introduces MLDroid, a web-based framework for detecting Android malware through dynamic analysis. MLDroid identifies relevant features using feature selection techniques and trains models using various machine learning approaches. The system was evaluated on over 500,000 real-world apps. Results show that integrating four techniques, such as deep learning, Farthest First Clustering, Y-MLP, and a nonlinear ensemble decision tree forest, along with rough set-based feature selection, achieved a high malware detection rate of 98.8%.

The growing use of Android devices has led to the emergence of sophisticated malware that employs complex obfuscation techniques, making static analysis alone inadequate for effective detection. Although dynamic analysis can uncover evasive behaviors, it is resource-intensive and limited in execution path coverage. To overcome these limitations, a hybrid detection approach combining static and dynamic analysis was proposed. A publicly available dataset containing 352 static and 323 dynamic features was developed to support both binary and multiclass classification. Irrelevant features were removed using the Information Gain algorithm. Experimental results demonstrated that the hybrid method significantly outperformed individual analyses, with the Random Forest classifier achieving 98.53% accuracy for binary classification and 90.1% for multiclass classification [11].

Android malware is a persistent and growing threat, affecting industries like healthcare, banking, transportation, government, and e-commerce. It became challenging to classify malicious apps. This study [12] proposed two ML-enabled dynamic analysis methods: one to categorize malware by category and the other by family. With the CCCS-CIC-

AndMal2020 dataset that contains 14 malware categories and 180 families, the proposed models were more than 96% accurate in detecting categories and more than 99% accurate in detecting families.

As Android devices gain popularity, malware detection and family categorization have become the most important research topics. Although static and dynamic analysis are most commonly used, they are based on complicated processes. In this work [13], the researchers present a hybrid approach optimized for multi-feature data to detect and categorize Android malware. Static analysis makes use of permissions and intents, three feature selection techniques being tested, with the best results from Chi-Square, yielding a 95.04% detection rate using a Random Forest classifier. Main static features were also tested for more information. In dynamic analysis, as opposed to previous methods involving one-way HTTP or transport-layer traffic, the researchers maintained complete session-level protocol layers. The Res7LSTM model was subsequently used to further classify uncertain samples. Experimental results indicate that the proposed method enhances malware family classification accuracy from 71.48% (previous work) to 99% by taking advantage of session-based, multi-layer traffic analysis with high accuracy using fewer static features.

Malware family classification categorizes similar malware instances into families to determine their behavioral patterns and facilitate recovery. A permission-based classification technique with both native and native/custom Android permissions is proposed [14]. Experiments were performed on the DREBIN dataset with various classifiers and tested using accuracy, macro F1-score, balanced accuracy (BAC), and Matthews correlation coefficient (MCC), the latter two measures being appropriate for imbalanced datasets. The results indicated that: i) tailor-made permissions enhanced classification performance, ii) accuracy and BAC varied up to 3.67% despite having the same model, and iii) LightGBM and AdaBoost performed better than other classifiers.

The detection and categorization of Android malware families must be automated to reduce the workload of security analysts. While various machine learning and deep learning techniques exist, the rapid growth of mobile applications necessitates faster and more accurate solutions. This study proposes a scalable method that significantly improves both detection accuracy and analysis speed. Evaluated on large-scale datasets, the proposed approach achieves an F-measure of 99.71%, a low false positive rate of 0.37%, and processes 300,000 samples in just 3.3 hours. For malware family classification across 28 families, it achieves an F-measure of 97.5%, a precision of 96.55%, and a recall of 98.64%. Comparative results demonstrate that our approach outperforms previous methods in both efficiency and effectiveness [15].

III. Research Methodology

This section presents a detailed description of the entire pipeline employed for Android Malware families' classification, comprising dataset preprocessing, feature selection, data splitting, training and testing of baseline classifiers, and the ensemble methods, as shown in Fig. 1.

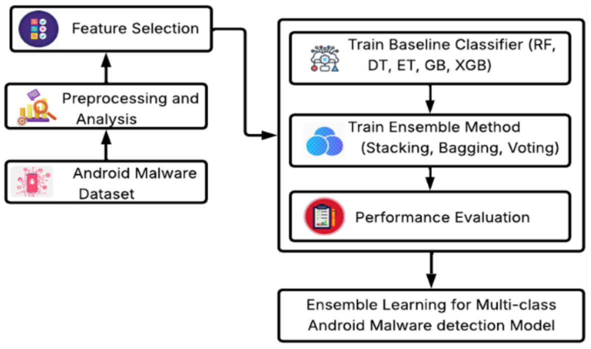
Feature Selection
Preprocessing and Analysis
Android Malware Dataset
Train Baseline Classifier (RF, DT, ET, GB, XGB)
Train Ensemble Method (Stacking, Bagging, Voting)
Performance Evaluation
Ensemble Learning for Multi-class Android Malware detection Model

Fig. 1. Proposed research methodology.

A. Dataset

We downloaded the Android Malware dataset from Kaggle [16]. The dataset was comprised of (355630 x 85) records. The dataset contains four families, such as Adware, Spyware, SMS_malware, and Bengin, as depicted in Fig. 2.

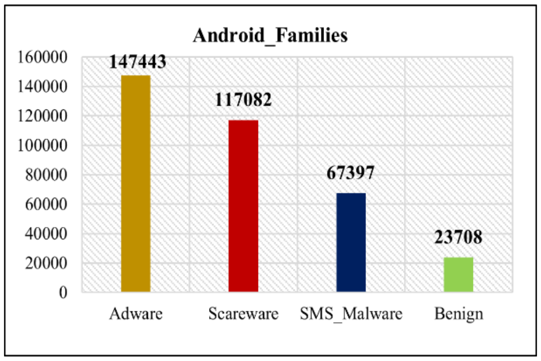

Fig. 2. Android families.

B. Preprocessing and Analysis

A non-informative timestamp column was removed, as it did not contribute any meaningful value to the training process. To prepare the dataset for machine learning algorithms, categorical features were encoded using the Label Encoding technique. Each categorical column was converted into a numerical format by assigning a unique integer to every distinct category. Missing values were either filled or removed, as appropriate, to maintain consistency and data integrity. After encoding, the dataset was separated into feature variables (X) and target labels (y). A final check for missing values in the feature set was conducted to ensure data completeness before model training and feature selection. These preprocessing steps ensured that the data was clean, fully numeric, and suitable for both statistical analysis and machine learning applications [17].

C. Feature Selection

To enhance model performance and reduce computational complexity, feature selection was performed using two statistical techniques: the Chi-Squared Test and Mutual Information.

1) Chi-Square test: The Chi-Squared test is a statistical hypothesis test used to assess the independence between categorical features (attributes) and the target class. In the context of feature selection, it assesses whether the actual frequency distribution of a feature significantly differs from the expected distribution assuming that the feature is statistically independent of the target variable [18]. Mathematically, the Chi-Squared statistic for a feature f is defined, as shown in Eq. (1):

$$Z^2 = \sum_{i=1}^{t} \sum_{k=1}^{d} \frac{(M_{ik} - P_{ik})^2}{P_{ik}} \tag{1}$$

where,

M_{ik}: Signifies the observed frequency of category i of features f with class labels k

P_{ik} characterizes the expected count of frequency assuming independence.

$\boldsymbol{k}$: Depicts the number of feature categories

$\boldsymbol{d}$: Shows the number of class labels

Fig. 3 demonstrates the selected features by their Chi-Squared scores, highlighting their relevance for classification. Top features such as Flow Bytes/s, Flow IAT Std, and various Inter-Arrival Time (IAT) metrics are essential for distinguishing normal traffic from anomalies or attacks. The prominence of IAT-related features (e.g., Fwd IAT Total, Bwd IAT Mean) underscores the importance of packet timing in detecting abnormal patterns linked to malicious activity. Metrics like Flow Bytes/s and Packet Length Variance help identify unusual data rates and packet sizes, often seen in network intrusions. Active and Idle time features further reveal session-level behavior, aiding in the detection of stealthy threats. The high Chi-Squared values (up to 10th) reflect the strong discriminatory power of these features. These statistically selected features provide a solid foundation for enhancing intrusion detection and traffic classification through machine learning.

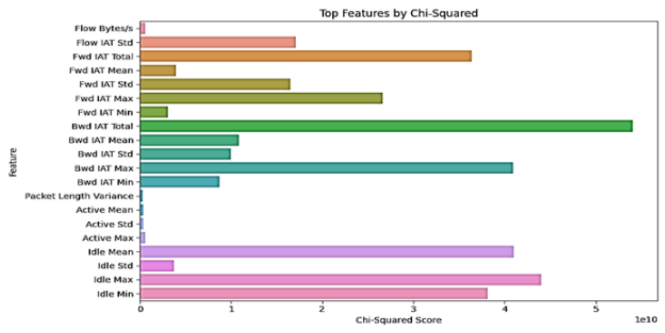

Fig. 3. Selected features using the Chi-Square method.

To enhance feature selection beyond the limitations of Chi-Squared analysis, Mutual Information (MI) was employed to capture both linear and non-linear dependencies between features and the target variable. This approach helps optimize the feature set by removing less informative attributes, particularly in scenarios involving complex network traffic patterns.

2) Mutual information feature selection: Mutual Information (MI) was used to quantify the amount of shared information between each feature and the target variable. Unlike the Chi-Squared test, MI can capture both linear and non-linear relationships. MI scores were computed for all features, and the top 20 with the highest scores were selected [19].

Fig. 4 presents a hierarchical listing of network traffic features sorted by their Mutual Information scores, indicating their predictive relevance to the target variable (e.g., intrusion detection). The top-ranked feature is Flow ID, followed by key metrics such as Fwd Avg Bytes/Bulk, Packet Length Mean, and Avg Bwd Segment Size, each with related sub-features indented beneath. These attributes represent various aspects of network behavior, including flow identifiers, packet size distributions, data transfer characteristics, and traffic dynamics. A supporting (though currently blank) tabular structure appears to outline threshold-based MI scores, likely intended to guide the selection of highly informative features. This step plays a critical role in building more accurate and robust machine learning models for network traffic classification and security analysis.

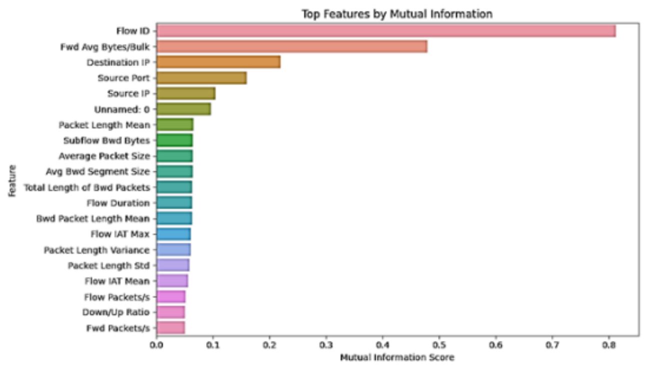

Fig. 4. Selected features using the MI method.

D. Model Development

We trained and evaluated four baseline classifiers, such as RF, ET, GB, XGB, and DT. To enhance predictive accuracy and robustness, we also implemented ensemble methods including stacking, bagging, and voting. Algorithm 1 demonstrates the training and testing mechanism of baseline and ensemble methods.

Algorithm-1: Training and Evolution of Baseline Classifiers

```
Input:    Selected features set
Output: evaluation metrics for each baseline classifier
Begin:
    1.    Split the dataset into features (X) and Target label (Y)
    2.    Apply train_test_split on X and Y with stratification
    3.    Define baseline classifiers
              •    RF
              •    DT
              •    ET
              •    GB
              •    XGB
    4.    For each baseline classifier:
              •    Train model
              •    Compute evaluation metrics:
                       o    Accuracy, precision, recall, F2 score, TNR
                       o    Confusion matrix
    5.    Train ensemble methods on trained baseline classifiers
    6.    Repeat Step 4 for each ensemble method
end
```

IV. Results and Discussion

The experiments in this study were performed on Windows 10 using Kaggle Notebook. The dataset was divided into training (70%) and test (30%) sets, with stratification to maintain the original label distribution, as depicted in Table I below.

TABLE I DIVISION OF SAMPLES

Training Samples	Testing samples	Total Samples
248938	106688	355626

TABLE II PERFORMANCE EVALUATION OF ML MODELS

ML models	Accuracy	Precision	Recall	F1 score	TNR
RF	81%	81%	81%	80%	92.37%
GB	70%	70%	70%	66%	87.95%
ET	79%	79%	79%	79%	91.72%
DT	79%	79%	79%	79%	91.92%
XGB	77%	77%	77%	77%	91.05%
Stacking	83%	82%	80%	8%	93.34%
Bagging	80%	89%	80%	80%	92.13%
Voting	82%	82%	82%	82%	92.95%

Table II presents the comparative performance of various machine learning models evaluated on a multi-class classification problem. Among the evaluated models, the Stacking classifier demonstrates the best overall performance, achieving 83% across accuracy, precision, recall, and F1 score, along with the highest true negative rate (TNR) of 93.34%. This reflects its strong generalization and balanced class discrimination, likely due to its ability to aggregate diverse base learners and optimize their combined predictions. Voting (82%) and RF (81%) also perform well across all metrics, benefiting from ensemble techniques that reduce variance and enhance stability. Bagging, while showing slightly lower accuracy (80%), achieves the highest precision (89%), indicating confident positive predictions, albeit with a trade-off in recall. ET (79%), DT (79%), and XGBoost (77%) yield moderate, balanced results but are outperformed by more advanced ensemble methods. GB exhibits the weakest performance, with 70% accuracy and the lowest F1 score (66%), possibly due to overfitting or suboptimal parameter tuning. Overall, ensemble approaches, especially stacking and voting, offer superior predictive performance and robustness, while single models and simpler boosting methods lag in capturing complex class relationships.

TABLE III CROSS-VALIDATION

ML models	Accuracy (CV =5)	Mean Std
RF	87%	± 0.0008
GB	88%	± 0.0011
ET	86%	± 0.0007
DT	83%	± 0.0007
XGB	84%	± 0.0009
Stacking	89%	± 0.0003
Bagging	84%	± 0.0009
Voting	85%	± 0.0012

To ensure the robustness of the ML model, 5-fold stratified cross-validation was performed. Table III describes the performance of ML classifiers and ensemble methods. The analysis illustrated that ensemble methods outperformed individual learners consistently. Among the baseline classifiers, GB (88%), RF (87%), and ET (86%) demonstrated robust predictive accuracy, while the lone DT ranked lower at 83% because of its greater overfitting tendency. XGBoost (84%) performed competitively, but could not surpass GB within this dataset. Ensemble methods improved results even more, with the highest accuracy achieved by the Stacking Classifier (89%) and lowest variance (±0.0003), performing superior generalization through diverse base learners via a meta-model. Bagging (84%) and Voting (85%) produced incremental improvements over single classifiers, but were inferior to stacking. The uniformly low standard deviations between models reflect consistently stable and dependable performance, and stacking proves to be the most stable and dependable means for deployment.

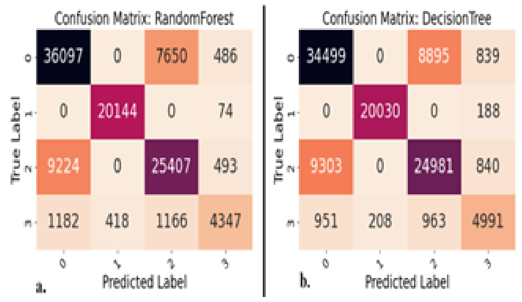

Fig. 5. Confusion matrix for ML models.

Fig. 5 represents the performance of the RF and DT models over unseen test data using confusion matrices. In Fig. 5(a), the RF model correctly classified 85995 out of the total samples (106688) with 20693 misclassifications, demonstrating high predictive accuracy. Similarly, Fig. 5(b) shows the DT model's confusion matrix, with 84501 correct predictions and 22187 misclassifications. The superior performance of the RF model compared to the DT model is expected, given RF's ensemble nature. By combining the predictions of multiple decision trees through bootstrap aggregation (bagging) and random feature selection, RF reduces variance and improves generalization to unseen data. In contrast, a single DT is more prone to overfitting, often capturing noise in the training set. The confusion matrix results further demonstrate that the RF model more effectively handles class boundaries, resulting in fewer misclassifications than the DT model.

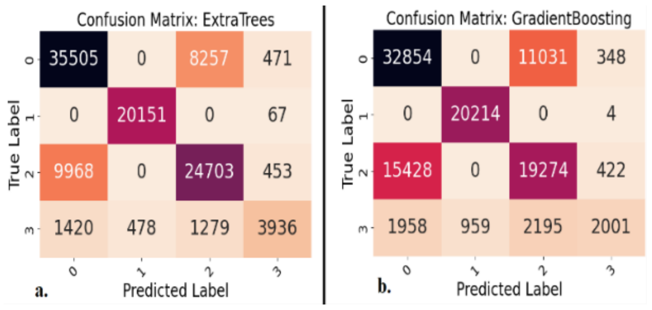

Fig. 6. Performance evaluation of classifiers.

Fig. 6(a) presents the confusion matrix for the ET classifier. The model demonstrated high predictive accuracy for class 1 and class 2, with 20,151 and 24,703 correct predictions, respectively. However, it showed significant misclassification between class 0 and class 2: 8,257 class 0 instances were incorrectly predicted as class 2, and 9,968 class 2 instances were misclassified as class 0, indicating notable overlap between these two classes. Fig. 6(b) shows the confusion matrix for the GB model. It correctly identified 20,214 class 1 and 19,274 class 2 samples, but exhibited greater confusion between class 0 and class 2 than Extra Trees, misclassifying 15,428 class 2 instances as class 0 and 11,031 class 0 instances as class 2. Both models performed poorly on class 3, with substantial dispersion across incorrect classes, indicating limited discriminative power for this class. Both models consistently underperform on class 3, with predictions for this class dispersed across the other categories. This suggests that class 3 may be underrepresented in the training data or lacks sufficiently distinctive features, making it difficult for the models to learn clear decision boundaries. The consistently poor classification performance for class 3 underscores a fundamental limitation in the dataset, either in terms of class imbalance or inadequate feature representation for this class.

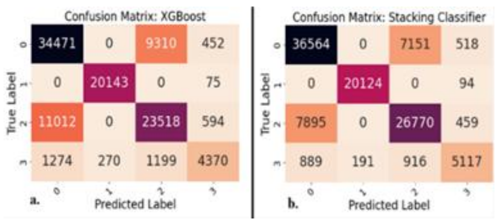

Fig. 7. Performance evaluation of ML models over unseen data.

Fig. 7(a) and Fig. 7(b) compare the performance of XGBoost and the Stacking classifier. XGBoost performs well on classes 1 and 2, with 20,143 and 23,518 correct predictions, respectively, as depicted in Fig. 7(a). However, in Fig. 7(b), it shows significant confusion between class 0 and class 2, misclassifying 9,310 class 0 instances as class 2 and 11,012 class 2 instances as class 0. This indicates overlapping feature space or weak decision boundaries between these classes. In contrast, the Stacking classifier demonstrates stronger overall performance. It substantially increases correct predictions for class 0 (36,564) and class 2 (26,770), while reducing misclassifications between them. Additionally, it improves classification for class 3, correctly identifying 5,117 instances, suggesting better generalization and improved handling of minority classes through ensemble learning. Overall, the Stacking model achieves more balanced classification and effectively addresses the limitations observed in XGBoost.

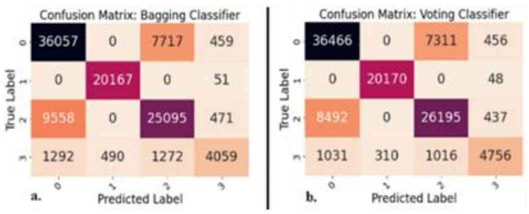

Fig. 8. Confusion matrix for ensemble methods.

The confusion matrices compare the bagging and voting classifiers. Fig. 8(a) shows the confusion matrix for the Bagging Classifier, where the classifier is showing high accuracy for class 0 (36,057 correct predictions) with fewer overall misclassifications. The Voting Classifier slightly improves correct predictions for class 0 (36,466) but introduces greater confusion, notably misclassifying 7,311 class 0 instances as class 1, as represented in Fig. 8(b). Diagonal entries indicate correct classifications, while off-diagonal entries represent errors. Bagging uses bootstrap aggregation to reduce variance and produce stable predictions. In contrast, voting combines outputs from diverse models, enhancing generalization but sometimes increasing misclassification between closely related classes. These results illustrate the trade-off between model diversity and predictive stability in ensemble methods.

The baseline classifier comparison provides key insights into performance across standard evaluation metrics, as represented in Fig. 9. Among the five models, RF, GB, ET, DT, and XGB. The XGBoost demonstrates the strongest performance, achieving the highest accuracy (92.37%) along with solid precision (81%), recall (81%), and F1 score (80%). This is largely due to its advanced regularization, gradient-based optimization, and ability to capture complex patterns while mitigating overfitting. All ensemble methods (RF, GB, ET, and XGB) outperform the single Decision Tree, which shows the lowest accuracy (87.95%) and reduced precision, recall, and F1 scores (all around 70%). This highlights the benefit of ensemble learning in enhancing generalization by combining multiple learners. ET (91.92%) and Random Forest (91.05%) also perform well, leveraging random feature selection and ensemble averaging to improve robustness. GB (91.72%), while slightly behind XGBoost, still delivers strong performance but lacks some of XGBoost's advanced optimization features, which likely explains the gap. In summary, XGBoost emerges as the most effective classifier, though other ensemble methods remain competitive. While the Decision Tree model is simple and interpretable, it lacks the predictive power needed for applications requiring high accuracy.

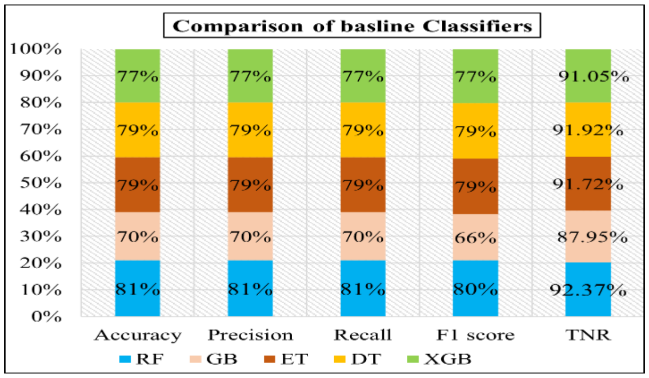

Fig. 9. Comparison of ML classifiers.

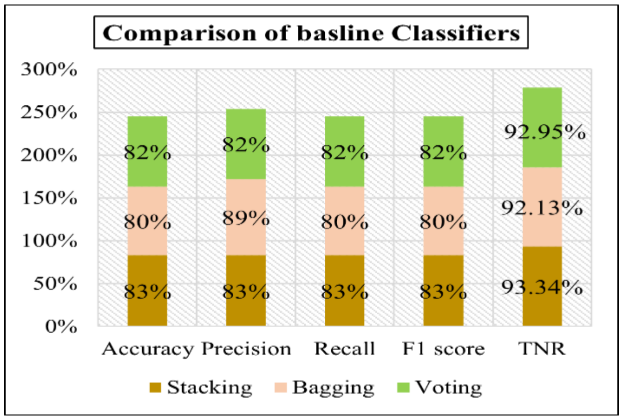

Fig. 10. Comparison of ensemble methods.

Fig. 10 depicts the comparison of three ensemble methods to determine the most effective approach. Key metrics, including accuracy, precision, recall, F1 score, and true negative rate (TNR), are used to evaluate performance, with values such as 92.95%, 92.13%, and 93.34% indicating strong results in individual configurations. Ensemble techniques, namely stacking, bagging, and voting, are also assessed for their ability to enhance model robustness by combining multiple classifiers. The inclusion of varying percentages (e.g., 30%, 25%, 20%) suggests an investigation into data split ratios or classifier weighting, aimed at further optimizing performance. Consistent precision values, such as 82%, 80%, and 83% across different ensemble methods, highlight the stability and reliability of these approaches. Overall, the analysis underscores the importance of selecting appropriate evaluation metrics and ensemble strategies to maximize classifier accuracy and predictive performance.

A. *Discussion*

The increased popularity of Android phones made them a favorite target among malicious applications, thus rapidly increasing malware on Android. Traditional detection systems typically perform binary classification (malicious vs. benign), overlooking the nuanced distinctions between malware families and differences that are crucial for threat intelligence, incident response, and forensic analysis [20] [21]. Addressing this gap, our study introduces a multi-class classification framework to differentiate between Android malware families using ML and ensemble-based models. As indicated in Table II, ensemble methods, specifically stacking, consistently outperformed baseline standalone models. The Stacking classifier obtained 83% in all the evaluation metrics (accuracy, precision, recall, F1-score) and a maximum TNR of 93.34%, validating its better generalization ability. Confusion matrix analyses also reveal the virtues and vices of separate classifiers. Models such as Random Forest and Extra Trees performed well with high accuracy for dominant families (e.g., Class 1 and Class 2), but struggled with minority classes (e.g., Class 3), revealing an ongoing class imbalance problem. Interestingly, Stacking demonstrated better performance with underrepresented classes, correctly classifying 5,117 examples of Class 3 and illustrating that it was capable of detecting intricate, non-linear relationships. Feature selection had an essential impact on model performance. Employing a hybrid technique incorporating Chi-Squared and Mutual Information algorithms, we found discriminative features like Flow Bytes/s, inter-arrival time (IAT) metrics, and session-based duration corresponding to behavioral patterns of various malware families. The dual-method strategy allowed dimensionality reduction to be done in an efficient manner without losing interpretability and enhancing model accuracy. Decision Trees were the worst-performing classifiers among all because of overfitting and the poor learning capacity of complex data patterns. Conversely, Voting classifiers and XGBoost yielded close-to-optimal performance, further supporting the efficacy of aggregation and boosting on high-variance, imbalanced datasets, such as Android malware. Ensemble learning supplemented with careful feature selection is an effective approach to steady multi-class Android malware detection, especially in real-world environments where data imbalance and intricate decision boundaries are prevalent.

V. Conclusion

This work presents an efficient and scalable machine learning architecture for multi-class Android malware family classification, addressing the limitations of traditional binary detection approaches. By leveraging ensemble learning, specifically the stacking method and a hybrid feature selection strategy combining Chi-Squared and Mutual Information techniques, the system achieves high accuracy, strong generalizability, and robustness to data imbalance. Empirical results demonstrate that the Stacking classifier outperforms both individual models and other ensemble techniques, achieving 83% across all evaluation metrics and the highest TNR of 93.34%. Notably, it significantly improves the detection of minority malware families, making it well-suited for real-world threat environments characterized by imbalanced class distributions and complex behavioral patterns. These findings validate the effectiveness of ensemble-based approaches and principled feature engineering in malware detection and provide a solid foundation for advancing automated and intelligent mobile security systems. Future work will focus on expanding the dataset, incorporating dynamic analysis features, and integrating deep learning models to further enhance detection accuracy and threat response capabilities.

VI. Future Work

While the present study demonstrates the efficacy of ensemble learning for Android malware detection, various promising research directions remain unexplored. To begin with, expanding the dataset to include newer and varied malware samples, such as zero-day attacks, would make the model more robust against changing threats. Second, the incorporation of dynamic analysis capabilities like system calls, network traffic, and runtime activity might supplement the static capabilities employed in this research to give a better understanding of malware activity. Third, combining ensemble techniques with deep learning frameworks (like CNNs, RNNs, or combination models) might further enhance detection rates, particularly in dealing with intricate temporal and behavioral patterns. Additionally, future efforts need to tackle adversarial robustness, as malware developers more and more use evasion and obfuscation mechanisms to defeat machine learning-based detectors. Research on federated and distributed learning methods is also a promising direction, enabling collaborative malware detection on devices without compromising user privacy. Finally, using explainable AI (XAI) techniques would add more transparency and trust in detection outcomes, which is extremely essential for cybersecurity professionals when deploying tools in critical real-world settings.

References

[1] M. Huang, “Press Center - Yearly 5G Smartphone Production Projected to Exceed 200 Million Units Thanks to Smartphone Brands’ Proactive Push in 2H20, Says TrendForce | TrendForce - Market research, price trend of DRAM, NAND Flash, LEDs, TFT-LCD and green energy, PV,” TrendForce. Accessed: Jul. 02, 2025. [Online]. Available: https://www.trendforce.com/presscenter/news/20200722-10398.html

[2] O. N. Elayan and A. M. Mustafa, “Android Malware Detection Using Deep Learning,” Procedia Computer Science, vol. 184, pp. 847–852, Jan. 2021, doi: 10.1016/j.procs.2021.03.106.

[3] “IT threat evolution in Q3 2021. Mobile statistics.” Accessed: Jul. 02, 2025. [Online]. Available: https://securelist.com/it-threat-evolution-in-q3-2021-mobile-statistics/105020/

[4] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, “A Review of Android Malware Detection Approaches Based on Machine Learning,” IEEE Access, vol. 8, pp. 124579–124607, 2020, doi: 10.1109/ACCESS.2020.3006143.

[5] D. Gupta and R. Rani, “Improving malware detection using big data and ensemble learning,” Computers & Electrical Engineering, vol. 86, p. 106729, Sep. 2020, doi: 10.1016/j.compeleceng.2020.106729.

[6] H. Alkahtani and T. H. H. Aldhyani, “Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices,” Sensors, vol. 22, no. 6, Art. no. 6, Jan. 2022, doi: 10.3390/s22062268.

[7] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, “Deep Learning for Android Malware Defenses: A Systematic Literature Review,” ACM Comput. Surv., vol. 55, no. 8, p. 153:1-153:36, Dec. 2022, doi: 10.1145/3544968.

[8] M. Mohd Saudi, M. A. Husainiamer, A. Ahmad, and M. Y. I. Idris, “iOS mobile malware analysis: a state-of-the-art,” J Comput Virol Hack Tech, vol. 20, no. 4, pp. 533–562, Nov. 2024, doi: 10.1007/s11416-023-00477-y.

[9] H. M. Ünver and K. Bakour, “Android malware detection based on image-based features and machine learning techniques,” SN Appl. Sci., vol. 2, no. 7, p. 1299, Jun. 2020, doi: 10.1007/s42452-020-3132-2.

[10] A. Mahindru and A. L. Sangal, “MLDroid—framework for Android malware detection using machine learning techniques,” Neural Comput & Applic, vol. 33, no. 10, pp. 5183–5240, May 2021, doi: 10.1007/s00521-020-05309-4.

[11] M. Dhalaria and E. Gandotra, "A hybrid approach for Android malware detection and family classification," IJIMAI, vol. 6, no. 6, pp. 174–188, 2021.

[12] A. H. E. Fiky, A. E. Shenawy, and M. A. Madkour, “Android Malware Category and Family Detection and Identification using Machine Learning,” Jul. 05, 2021, arXiv: arXiv:2107.01927. doi: 10.48550/arXiv.2107.01927.

[13] C. Ding, N. Luktarhan, B. Lu, and W. Zhang, “A Hybrid Analysis-Based Approach to Android Malware Family Classification,” Entropy, vol. 23, no. 8, Art. no. 8, Aug. 2021, doi: 10.3390/e23081009.

[14] M. Kim, D. Kim, C. Hwang, S. Cho, S. Han, and M. Park, “Machine-Learning-Based Android Malware Family Classification Using Built-In and Custom Permissions,” Applied Sciences, vol. 11, no. 21, Art. no. 21, Jan. 2021, doi: 10.3390/app112110244.

[15] B. Sun, T. Takahashi, T. Ban, and D. Inoue, “Detecting Android Malware and Classifying Its Families in Large-scale Datasets,” ACM Trans. Manage. Inf. Syst., vol. 13, no. 2, p. 12:1-12:21, Oct. 2021, doi: 10.1145/3464323.

[16] “Android Malware Detection.” Accessed: Jul. 02, 2025. [Online]. Available: https://www.kaggle.com/datasets/subhajournal/android-malware-detection

[17] A. Parashar, A. Parashar, W. Ding, M. Shabaz, and I. Rida, “Data preprocessing and feature selection techniques in gait recognition: A comparative study of machine learning and deep learning approaches,” Pattern Recognition Letters, vol. 172, pp. 65–73, Aug. 2023, doi: 10.1016/j.patrec.2023.05.021.

[18] I. Sumaiya Thaseen and C. Aswani Kumar, “Intrusion detection model using fusion of chi-square feature selection and multi class SVM,” Journal of King Saud University - Computer and Information Sciences, vol. 29, no. 4, pp. 462–472, Oct. 2017, doi: 10.1016/j.jksuci.2015.12.004.

[19] L. Hu, L. Gao, Y. Li, P. Zhang, and W. Gao, “Feature-specific mutual information variation for multi-label feature selection,” Information Sciences, vol. 593, pp. 449–471, May 2022, doi: 10.1016/j.ins.2022.02.024.

[20] A. Alhogail and R. A. Alharbi, “Effective ML-Based Android Malware Detection and Categorization,” Electronics, vol. 14, no. 8, Art. no. 8, Jan. 2025, doi: 10.3390/electronics14081486.

[21] A. Muzaffar, H. Ragab Hassen, M. A. Lones, and H. Zantout, “An in-depth review of machine learning based Android malware detection,” Computers & Security, vol. 121, p. 102833, Oct. 2022, doi: 10.1016/j.cose.2022.102833.