Enhanced Android Malware Detection Using Deep Learning and Ensemble Techniques

Abdul Museeb[1]*, Yaman Hamed[2], Rajalingam Sokkalingam[3], Anis Amazigh Hamza[4], Atta Ullah[5], Iliyas Karim Khan[6]
Centre for Intelligent Asset Reliability-Institute of Emerging Digital Technologies-Department of Applied Sciences,
Universiti Teknologi Petronas, Seri Iskandar, Perak, Malaysia[1, 2, 3]
XLIM, UMR CNRS 7252, SRI (Smart Systems and Network), University of Limoges,
16 Rue Atlantis, 87068 Limoges Cedex, France[4]
Fundamental & Applied Sciences Department, Universiti Teknologi PETRONAS,
Seri Iskandar 32610, Perak, Malaysia[5, 6]

***Abstract*—Android malware continues to pose significant security threats, with evolving tactics that often bypass traditional detection systems. Existing detection mechanisms remain ineffective against obfuscated or novel malware variants, necessitating the development of more robust detection techniques. This study introduces a comprehensive machine learning framework for Android malware detection that leverages a systematic comparison between a deep Neural Network and diverse ensemble methods, including Voting Ensemble, Stacking Ensemble, XGBoost, and Random Forest. Unlike prior studies that often focus on individual approaches, this work provides an empirical benchmark that demonstrates how practical ensemble configurations can achieve superior performance while maintaining computational efficiency. The model is trained using the CIC-AndMal2017 dataset, incorporating a comprehensive set of static features, including API calls, permissions, services, receivers, and activities. Feature selection was performed to optimize model performance, reducing redundancy and improving detection accuracy. The models were evaluated on multiple classification metrics, including accuracy, F1-score, and confusion matrices, with the Voting Ensemble model achieving an accuracy of 94.14%, outperforming all other approaches, including the deep neural network. This study contributes to the field by demonstrating that a carefully constructed ensemble of diverse classifiers can not only improve detection accuracy but also offer a more scalable, lightweight solution compared to complex deep learning models. The research provides a significant advancement in practical Android malware detection by identifying optimal strategies that balance performance with computational efficiency.**

***Keywords*—Android malware detection; machine learning; API calls; permissions; android security; malware classification**

I. Introduction

The rise of mobile technology has made smartphones integral to daily life, often outpacing desktop systems in user interaction and internet engagement [1]. With vast amounts of sensitive data stored on mobile devices, including login credentials and banking information, they have become prime targets for cyber threats [2]. Android, with an 88% market share, leads the mobile ecosystem and powers billions of devices worldwide [3]. Its open-source nature, while enabling widespread adoption, also increases security vulnerabilities [4]. Its open-source architecture fosters broad adoption due to flexibility, cost-efficiency, and a vast application library. However, this openness also exposes it to heightened security risks, enabling malicious actors to exploit inherent system vulnerabilities and bypass inadequate security checks [5]. Research in Android malware detection has focused on static and dynamic analysis, as well as hybrid approaches [6]. Static analysis is scalable but ineffective against obfuscated malware, while dynamic analysis uncovers runtime threats at the cost of high computational demand. Hybrid methods aim to balance these strengths [7]. Static analysis inspects application code without execution, offering scalability but often failing against obfuscated or polymorphic malware. Conversely, dynamic analysis monitors real-time application behavior in controlled environments, effectively uncovering runtime threats but demanding significant computational resources. Hybrid methods strive to merge the strengths of both, achieving a trade-off between performance and threat coverage [8]. Traditional network intrusion detection systems (NIDS), divided into signature-based, anomaly-based, and hybrid models, also enhance security but face challenges in detecting novel threats [9]. Therefore, the need for intelligent, adaptive, and lightweight malware detection frameworks remains a critical focus in the broader context of Android security.

This work addresses a specific need in the current landscape of Android malware detection research. While previous studies have explored hybrid models that integrate deep learning with complex meta-heuristic optimizations [10, 11], there is a noticeable gap for a clear benchmark comparing a standard deep neural network against more straightforward ensemble constructions like Stacking and Voting. The relative performance of these high-accuracy ensemble methods against a deep learning baseline remains underexplored. Therefore, this study provides a crucial empirical comparison focusing on practical ensemble designs, offering valuable insights into the trade-offs between model complexity and detection performance for real-world applications.

To overcome these challenges, our study proposes an enhanced static analysis framework that uses deep learning and ensemble classification models. This approach leverages static features from the CICAndMal2017 dataset, offering a lightweight, scalable solution that can detect both known and unseen malware with high accuracy and minimal overhead. The main contributions of this study are as follows:

*Corresponding author.

- We propose an enhanced Android malware detection framework that integrates deep learning (neural network) and ensemble techniques (stacking and voting) to improve detection accuracy using static features, without the need for runtime analysis.
- Extensive experimentation is performed using the publicly available CICAndMal2017 dataset, ensuring a realistic and diverse malware landscape. This dataset enables the extraction of meaningful static features that are representative of modern Android threats.
- Our results indicate that ensemble models, particularly the Voting Ensemble, outperform both traditional machine learning classifiers and standalone deep learning models, demonstrating improved detection performance and robustness.
- We provide a comparative analysis with baseline models, including Random Forest, XGBoost, and Neural Networks, highlighting the advantages of ensemble approaches in terms of accuracy, precision, and generalization for Android malware detection.

This study is organized in the following way: Section II looks at the current research on finding Android malware. It examines the evolution of methodologies from conventional machine learning to contemporary deep learning and hybrid ensembles. Section III goes into detail about the proposed framework. It includes a description of the CICAndMal2017 dataset, the steps for static analysis and feature extraction, and the setup of the learning algorithms that were tested. Section IV provides an extensive performance evaluation of the tested classifiers, including Neural Networks, Random Forest, XGBoost, and a Voting Ensemble, and examines the results that illustrate the ensemble's superiority. In Section V, the study comes to a close by listing the main contributions and suggesting some possible directions for future work on making malware detection technologies that are both effective and flexible.

II. Related Work

The detection of Android malware has become a critical research focus due to the evolving complexity of malicious applications and their capacity to bypass conventional security mechanisms. To address these challenges, researchers have explored a range of approaches, including traditional machine learning, which involves selecting suitable classifiers for malware detection; feature engineering, which focuses on extracting and optimizing informative attributes; and ensemble learning, which integrates multiple models to enhance predictive accuracy and resilience. These techniques are applied within static, dynamic, or hybrid analysis frameworks and are generally categorized as signature-based or anomaly-based, depending on whether they rely on known attack patterns or behavioral deviations. Table I provides a consolidated summary of representative studies, outlining their methodologies, datasets, detection types, performance levels, and associated drawbacks.

TABLE I. Summary of Literature Review

Ref.	Approach	Feature Type	Detection Type	Methodology	Dataset Used	Reported Accuracy	Drawbacks
[11]	ML (DT, RF)	Permissions	Static	Decision Tree, Random Forest	Custom datasets	90%	Weak against zero-day; limited feature diversity
[20]	ML	Permissions	Static	Permission-based filtering	Custom dataset	81%	Insufficient as standalone; poor detection precision
[21]	ML (kNN)	Permissions, API, Intents	Static	DroidMat with kNN	Not specified	Not reported	Scalability issues; weak base learner
[22]	API Monitoring	API Calls	Dynamic	Logcat instrumentation	Custom dataset	Not reported	Resource-intensive; not scalable for large datasets
[23]	ML (Permissions + API)	API + Permissions	Static	Feature fusion with ML classifiers	Not specified	Not reported	Limited generalization; lacks robustness
[12]	ML (SVM)	Permissions	Static	Reduced feature SVM	Custom dataset	93.62%	Vulnerable to obfuscation; limited feature scope
[24]	ML (Opcode Transformation)	Dalvik Opcodes	Static	Grayscale opcode images + ML	Custom dataset	91% (RF)	High dimensionality; scalability issues
[13]	ML (Dynamic Features)	CPU, Memory, Network	Dynamic	SVM on Drebin resource metrics	Drebin dataset	94.2%*	Computationally expensive; impractical for real-time use
[17]	Ensemble + Optimization	Static	Static	OEL-AMD (BGWO + Ensemble)	Not specified	92.4%	High complexity; feature selection overhead
[25]	Graph-Based ML	API Dependency Graphs	Static	DroidSIFT – graph similarity	Genome	93%	Heavy preprocessing; computationally expensive
[21]	Dynamic Analysis Tools	API, Sys Calls, NetFlow	Dynamic	DroidBox, CopperDroid, AMAT	Sandbox datasets	Not reported	Susceptible to anti-emulation evasion

Machine learning has arisen as a potent method to address the shortcomings of conventional signature-based detection, facilitating the recognition of new and obscured malware. Various classifiers and feature sets have been employed, each offering unique strengths but facing specific challenges. Permission-based detection is one of the earliest lightweight methods. Aung et al. [12] used Decision Trees and Random Forest on permissions (~90% accuracy) but with weak zero-day resistance, while Huang et al. [13] found permissions effective as a quick filter (81%) though insufficient as a standalone solution. To improve robustness, DroidMat [14] combined permissions, API calls, and intents using kNN, though

scalability issues remained. Nishimoto et al. [15] proposed logcat-based API monitoring to track sensitive invocations, which detected obfuscation but required heavy resources. Chan and Song [16] showed that permissions combined with selected API calls improved accuracy over single features. Similarly, Li et al. [17] applied SVM on reduced permission sets with 93.62% accuracy, but a limited feature scope reduced robustness. More advanced representations were explored by Anderson et al. [18] , who transformed Dalvik op-codes into grayscale images for classification (91% with RF), though dimensionality hindered scalability. Massarelli et al. [19] employed dynamic resource usage metrics with SVM (94.2%), but such runtime analysis introduced computational overhead, limiting real-time use.

Ensemble learning and hybrid models aim to improve detection rates and reduce false positives by combining the strengths of multiple classifiers or methodologies. Ensemble learning aggregates predictions from several base models to produce a more accurate and robust result, while hybrid models integrate static and dynamic detection techniques or metaheuristic algorithms with machine learning. Optimized Ensemble Learning for Android Malware Detection OEL-AMD, proposed by Sharma et al.[10], OEL-AMD, which used Binary Grey Wolf Optimization (BGWO) for feature selection and ensemble classification, achieved 92.4% accuracy with fewer false positives. Sharma and Agrawal et al. [11] developed a hybrid model combining the Intelligent Water Drop (IWD) algorithm with deep learning, reaching 94.5% accuracy while preserving key features. Droid SIFT [20] introduced weighted contextual API dependency graphs for classifying unknown apps, obtaining ~93% accuracy on the Genome dataset, though at a high computational cost. Tools like Droid Box, Copper Droid, and AMAT [21] used sandbox-based behavioral analysis, but their effectiveness is limited due to anti-emulation tactics, such as Telephony Manager checks, exploited by Pincer malware [22].

Feature engineering and selection methods have been widely used to enhance classification efficiency. Li et al. [23] introduced Significant Permission Identification (SigPID) to prune redundant permissions, improving performance. Bhagwat and Gupta [24] applied PCA and mutual information to reduce dimensionality while retaining informative features, achieving 92.8%. Advanced optimization techniques like Harris Hawks Optimization (HHO) and Genetic Algorithms (GA) [25] further refined feature subsets for classification. Beyond malware-specific studies, clustering-based methods also contribute to feature optimization. The Enhanced Gap Statistic (EGS) [26] improved optimal cluster determination by standardizing reference data. The Kernelized Rank Order Distance (KROD) method. In [27], the authors transformed non-spherical data into spherical form, enhancing clustering accuracy. An improved EGS variant [28] incorporated Gaussian standardization, outperforming classical clustering methods on large datasets. A modified cubic B-spline method [29] offered better error estimates in curve fitting, while a combined approach [30] integrated winsorization, KROMD, and enhanced gap statistic to improve K-means clustering stability. A comprehensive evaluation [31] of the Gap Statistic revealed efficiency on simple datasets but weaknesses on complex, high-dimensional data. Together, these studies emphasize the importance of clustering and optimization for improving feature selection and dimensionality reduction in malware detection pipelines.

Deep learning models have gained increasing attention due to their ability to automatically learn complex feature relationships and uncover hidden patterns in large-scale malware datasets. Unlike traditional ML methods that rely heavily on handcrafted features, deep learning can extract hierarchical representations directly from raw or transformed data, making it well-suited for detecting obfuscation and polymorphic malware. Kim et al. [32] applied a CNN on API call graphs, achieving 96.2% accuracy and demonstrating effectiveness in detecting behavioral patterns. Fallah and Bidgoly [33] employed LSTMs to classify malware families based on temporal network traffic data, reaching 94% accuracy. While deep learning improves adaptability and accuracy, challenges remain regarding computational cost and interpretability, limiting deployment on resource-constrained devices. More recently, a survey by Kouliaridis and Kambourakis et al. [34] confirmed the effectiveness of ensemble models for Android malware detection, particularly with modern datasets. This supports our methodological choice of employing ensemble techniques on the CIC-AndMal2017 dataset. Furthermore, a comprehensive review by Liu et al. [35] synthesizes the landscape of machine learning-based detection, underscoring the persistent challenge of balancing model performance with computational efficiency, a key gap our research addresses by demonstrating the efficacy of practical ensemble methods.

In summary, prior studies demonstrate the evolution of Android malware detection from lightweight ML classifiers to advanced ensembles, feature engineering, and deep learning methods. However, existing approaches face trade-offs between scalability, accuracy, and robustness. These gaps highlight the need for an enhanced framework that leverages deep learning combined with ensemble techniques to achieve reliable, efficient, and scalable Android malware detection.

III. METHODOLOGY

The experimental framework established in this study is designed to construct a robust Android malware detection system. The procedure commences with dataset curation and preprocessing, advances through a critical feature selection process, and proceeds to the training phase of ensemble models (Voting and Stacking) and Neural Networks. The concluding phase entails a thorough performance assessment to verify the detecting capabilities. Fig 1 presents an illustrated overview of an end-to-end process.

A. Dataset Collection and Preprocessing

The efficacy of any machine learning model depends on the quality and diversity of the training dataset. In this study, we utilized the CIC-AndMal2017 dataset, a robust and comprehensive collection of labeled Android applications. This dataset serves as the cornerstone of our Android malware detection system and provides the foundation for building and evaluating the models. The dataset is rich with real-world samples, including both benign and malicious applications, which is essential for developing a model that can generalize

well to new, unseen data. The process began with collecting and preparing the data. Unlike other datasets, which often consist of a limited set of features, the CIC-AndMal2017 dataset includes a wide variety of attributes. These features cover different aspects of Android application behavior and functionality, making it a comprehensive source for malware detection.

The CIC-AndMal2017 [36] dataset serves as the cornerstone of our Android malware detection system. It includes thousands of labeled APKs categorized into benign, adware, scareware, and SMS malware. These categories encompass a broad spectrum of practical applications and malware, guaranteeing varied and comprehensive training and testing data. Benign apps act as a baseline, representing non-malicious behavior, while adware apps disrupt user experience with intrusive ads. Scareware tricks users into harmful actions, and SMS malware exploits SMS functionality to send unauthorized messages or steal personal data. Together, these categories provide the foundation for an effective and comprehensive malware detection model, as summarized in Table II.

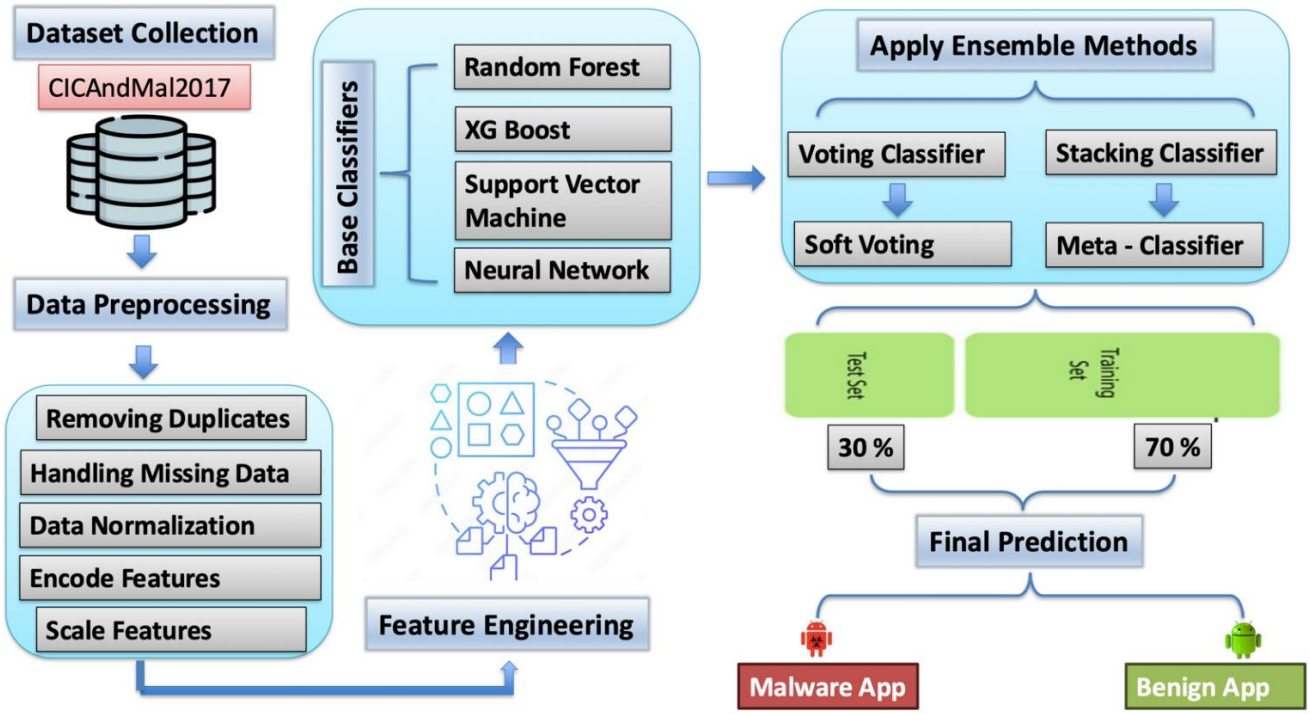

Fig. 1. Proposed methodology framework.

TABLE II. DESCRIPTION OF THE CATEGORIES IN THE CIC-ANDMAL2017 ANDROID MALWARE DATASET

Category	Description
Benign Applications	Non-malicious apps that serve as a baseline for comparison.
Adware	Apps designed to display intrusive advertisements, often affecting user experience and consuming resources.
Scareware	Apps that use deceptive tactics to manipulate users, such as creating false alerts or urging unnecessary purchases.
SMS Malware	Apps that exploit SMS functionality for malicious purposes, like sending unauthorized messages or stealing data.

The dataset is well-balanced, with the number of benign and malicious apps being approximately equal, ensuring that the model is trained on a balanced representation of both classes. The dataset provides 900+ samples, which are ideal for training machine learning models that need to generalize well on real-world data. To prepare the data for machine learning model training, several preprocessing steps were undertaken to ensure that the features were clean, standardized, and ready for use. These steps included:

1) Label Encoding: All categorical variables, such as feature labels and app categories, were encoded into numeric values using Label Encoder from scikit-learn. This transformation was essential to make the data compatible with machine learning algorithms.

2) Feature scaling: Numerical features were standardized using StandardScaler to ensure that all features were on the same scale. This is crucial to avoid biases in model training, especially when certain features may have larger magnitudes than others.

3) Handling missing data: Any missing or incomplete entries were handled through imputation or removal, ensuring no gaps in the dataset that could affect the model's performance.

The models were trained on a 70% subset of the data, with the remaining 30% held out as a test set. This hold-out method is crucial for validating predictive performance and ensuring the models can generalize beyond the data they were trained on.

B. Feature Extraction

The Feature extraction is essential for transforming raw APK data into structured input for machine learning models. In this study, we extracted key features from the CIC-AndMal2017 dataset, including API calls, permissions, services, receivers, and activities. These features play a critical role in distinguishing between benign and malicious applications.

- API Calls: Methods invoked by the application during runtime, reflecting interactions with the system. Categorized into android_api, com_api, and java_api sets.
- Permissions: Requested access rights by the app, such as accessing the location or sending SMS messages, which may indicate potential malicious activity.
- Services: Background tasks run by the app, often indicating persistent behavior associated with malicious activity.
- Receivers: Components that listen for system or app-level events, often exploited by malware for covert operations.
- Activities: UI components that define user interactions, providing insight into the app's intent.

These extracted features were stored in JSON format, where each APK was represented as a file containing metadata and features. The dataset was organized into several configurations for model training:

- API + Permissions + Services data: A comprehensive dataset integrating all three feature types.
- API + Permissions data: Focused on API behavior and requested permissions.
- API + Services data: Examining API interactions alongside background services
- Permissions + Services data: Exploring the link between permissions and background behavior.
- API data alone: Isolated API calls to assess their standalone predictive power.

These datasets were stored as CSV files for efficient processing. The extracted features are summarized in Table III, while their distribution across different application types is shown in Table IV. These tables offer clarity on the features used for model training and their relevance for Android malware detection.

TABLE III. SUMMARY OF EXTRACTED FEATURES

Feature	Definition	Example
Permissions	Privileges requested by the app to access system resources	ACCESS_FINE_LOCATION
API Calls	Methods invoked during runtime	java.net. HttpURLConnection
Services	Processes running in the background	android.app.Service
Receivers	Components listening for broadcasts	android.content.BroadcastReceiver
Activities	Screens/interfaces of the application	android.app.Activity

TABLE IV. FEATURE COUNTS FOR DIFFERENT APK TYPES

APK Type	Permissions	APIs	Services	Receivers	Activities
Benign	636	202820	1837	1514	11063
Adware	320	14560	920	800	2300
Scareware	200	10234	700	400	1200
SMS Malware	150	9000	500	300	1000

C. Feature Selection Using Correlation Analysis

Feature selection is a critical step in enhancing the performance of machine learning models by reducing the dataset's complexity while retaining the most relevant features. In this study, we selected features from the CIC-AndMal2017 dataset that would most effectively help distinguish between benign and malicious applications. The selection process focused on API calls, permissions, services, receivers, and activities, which together represent the application's behaviour and intent. To identify the most relevant features, we used the Pearson correlation coefficient to measure the linear relationship between each feature and the target variable (malicious = 1, benign = 0). The Pearson correlation coefficient is calculated using the formula:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_{ij}-\overline{x_j})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_{ij}-\overline{x_j})^2 \sum_{i=1}^{n}(y_i-\bar{y})^2}} \quad (1)$$

where,

- x_{ij} is the value of feature x_j for sample i,
- y_i is the target label for sample i,
- $\overline{x_j}$ and $\bar{y}$ are the means of feature x_j and target label y.

This approach allowed us to measure the correlation between each characteristic and the target variable, helping us determine which features were most strongly associated with detecting malicious behavior.

Features with a correlation of $|r_{xy}| < 0.3$, indicating weak correlation with the target label, were excluded from further analysis. These features, such as services and receivers, showed minimal ability to distinguish between benign and malicious applications, and therefore did not contribute significantly to the classification task. On the other hand, features with stronger correlations, primarily API calls and permissions, were retained for training the model. These features demonstrated a high correlation with the target label, providing critical information to accurately classify applications as either benign or malicious. By focusing on these highly relevant features, we ensured the model was trained on the most meaningful data, enhancing its ability to make accurate predictions.

D. Model Training and Evaluation

This study utilized various machine learning models to categorize Android applications as benign or harmful. The models used include Voting Ensemble, Stacking Ensemble, and Neural Networks, which were all trained and evaluated using the extracted features from the CIC-AndMal2017 dataset. Each model was selected for its capacity to manage high-dimensional datasets and its resilience in practical application contexts. The initial phase of model construction involved partitioning the dataset into training and testing subsets. The dataset was divided into 70% for training and 30% for testing, ensuring that the models were trained on a significant volume of data while maintaining a distinct set for impartial performance assessment. We employed a scaled dataset for model training, applying StandardScaler to both the training and test sets to standardize all characteristics. The models were trained using the following approach:

1) Neural network model: The Neural Network was constructed with a sequence of fully connected (Dense) layers, utilizing ReLU activation functions. To enhance training stability and prevent overfitting, batch normalization and dropout layers were strategically incorporated. A final output layer with a sigmoid activation function was employed for the binary classification task (benign versus malicious).

2) Voting ensemble: The Voting Ensemble integrates predictions from several heterogeneous base estimators, including Random Forest, XGBoost, and Linear Models. It operates on a "soft voting" principle, where the final prediction is derived by averaging the class probabilities output by each constituent model, thereby leveraging the collective wisdom of the ensemble.

3) Stacking ensemble: A Stacking Ensemble was implemented to synthesize the capabilities of diverse base classifiers. The predictions from these base models serve as input features for a meta-classifier, which was configured as a Logistic Regression model in this work. This two-tiered architecture aims to learn how to best combine the base models' outputs to achieve a more accurate and robust final prediction than any single model could provide.

Each model was trained with a fixed number of epochs for the Neural Network model (50 epochs), while the ensemble models used the default settings for their classifiers.

E. Hyperparameter Tuning and Evaluation Metrics

To ensure optimal performance, hyperparameters were tuned for each model. In particular, the Random Forest model within the ensemble classifiers was configured with:

Number of trees ($n_{estimators}$): 100.

Maximum depth: No limit to tree depth.

Minimum samples per split: 2.

Minimum samples per leaf: 1.

These settings were selected to avoid overfitting, while maintaining model complexity to capture meaningful patterns from the data. The XGBoost model was also fine-tuned with a learning rate of 0.1 and a max depth of 6, aiming to balance model complexity and generalization.

Each trained model's performance was assessed using a set of classification measures, including the Confusion Matrix, Accuracy, Recall, Precision, and F1-Score. These measures evaluate each model's capacity to distinguish between benign and malicious applications, with a particular focus on the F1-Score due to its resilience in addressing class imbalances.

Accuracy: Measures the overall correctness of the model by evaluating the proportion of correctly classified instances out of the total samples. It is calculated as:

Accuracy:

$$\frac{TP+TN}{TP+TN+FP+FN} \tag{2}$$

Precision: Evaluates how many of the predicted malicious applications are malicious. A higher precision score indicates fewer false positives. It is computed as:

Precision:

$$\frac{TP}{TP+FP} \tag{3}$$

Recall (Sensitivity): Determines the model's ability to correctly identify malware by measuring how many actual malicious applications were detected. It is given by:

Recall:

$$\frac{TP}{TP+FN} \tag{4}$$

F1-Score: A mean of precision and recall that balances false positives and false negatives. It is particularly advantageous when the dataset exhibits inequality. The formula is:

F1-Score:

$$2 \times \frac{precion \times sensitivity}{precion + sensitivity} \tag{5}$$

This methodology utilizes a Voting Ensemble, Stacking Ensemble, and Neural Networks to detect Android malware, with feature selection based on all available features, including API calls, permissions, services, receivers, and activities. Unlike previous approaches that focused on a subset of features, this

study leverages the full spectrum of extracted features to train the models. By incorporating all relevant attributes, the approach ensures a comprehensive analysis of Android application behavior, leading to enhanced model performance and accuracy. The models effectively handle high-dimensional data, improving classification results while minimizing computational complexity.

IV. Results and Discussion

In this section, we present the findings of the models assessed for malicious application detection, along by a comprehensive explanation of their performance. The models, comprising Voting Ensemble, Stacking Ensemble, XGBoost, Neural Networks, and Random Forest, were assessed utilizing conventional classification metrics. A comparison analysis was conducted to elucidate the advantages and drawbacks of each model.

A. Experimental Setup

The studies utilized the CIC-AndMal2017 dataset, comprising labeled Android applications classified as benign or malicious, including adware, scareware, and SMS malware. The dataset was pre-processed and divided into training (70%) and testing (30%) subsets. The features for classification included API calls, permissions, services, receivers, and activities, which were extracted from the dataset and encoded for use in machine learning models. The models were assessed using standard classification metrics to determine their capability to differentiate between benign and malicious apps.

B. Model Training and Evaluation

We trained five different models that we exercised during the training process. The selection of these models was predicated on their shown efficacy in managing complicated, high-dimensional datasets. Accuracy, precision, recall, and F1-score were the metrics that were utilized to assess the performance of each model. The results from these examinations are described in Table V.

TABLE V. Evaluation Metrics for All Models

Model	Accuracy	Precision (Benign)	Precision (Malware)	Recall (Benign)	Recall (Malware)	F1-Score (Benign)	F1-Score (Malware)
Random Forest	91.21%	0.92	0.89	0.94	0.85	0.93	0.87
Neural Network	92.67%	0.94	0.89	0.94	0.89	0.94	0.89
XGBoost	93.41%	0.94	0.92	0.96	0.88	0.93	0.90
Stacking Ensemble	93.77%	0.94	0.92	0.94	0.92	0.94	0.91
Voting Ensemble	94.14%	0.95	0.93	0.97	0.89	0.96	0.91

As shown in Table V, Voting Ensemble achieved the highest accuracy (94.14%), closely followed by the Stacking Ensemble (93.77%) and XGBoost (93.41%). The Neural Network model achieved a slightly lower accuracy of 92.67%, but it showed promising results in terms of recall for both classes, indicating its good generalization ability for classifying both safe and harmful applications. The Random Forest model, while robust, lagged slightly behind the other models in terms of overall accuracy and F1-score.

C. Model Performance and Discussion

Voting Ensemble demonstrated superior performance, achieving the highest accuracy and F1-score. This model combined multiple classifiers, which likely contributed to its robust performance across both benign and malicious classifications. Its high recall for benign applications (0.97) indicates that it was highly effective at correctly identifying non-malicious apps, minimizing false negatives.

Stacking Ensemble performed similarly to the Voting Ensemble, with only a slight reduction in accuracy and F1-score. This model combines base classifiers to improve prediction accuracy, and its balanced performance across both precision and recall suggests that it effectively handled the trade-off between false positives and false negatives.

XGBoost, recognized for its efficacy and performance, achieved 93.41% accuracy. While it had comparable results to the Random Forest model, its lower precision for malware (0.92) compared to Voting Ensemble indicates that XGBoost may have struggled more with correctly classifying some malicious apps.

Neural Networks were a strong performer in terms of recall, particularly for malware detection (0.89). However, their lower precision and accuracy in comparison to ensemble methods suggest that they may not be as reliable for this task without further hyperparameter tuning and training.

Random Forest, despite being a widely used model for classification tasks, achieved the lowest accuracy in this experiment (91.21%). While it performed well on benign applications (precision = 0.92 and recall = 0.94), it faced challenges in classifying malware, particularly in reducing false negatives.

D. Confusion Matrix

A comprehensive assessment of model performance was conducted utilizing confusion matrices. These matrices facilitated the evaluation of false positives and false negatives in the predictions of each model. Below is a confusion matrix for the Voting Ensemble model, Fig. 3 offering a visual picture of the model's efficacy in differentiating between benign and malicious applications. The Voting Ensemble model effectively reduced both false positives and false negatives, demonstrating its durability and calibration. To further demonstrate the performance of the models, Fig. 4 presents confusion matrices for the individual models, including Neural Network, Stacking Ensemble, Random Forest, and XGBoost.

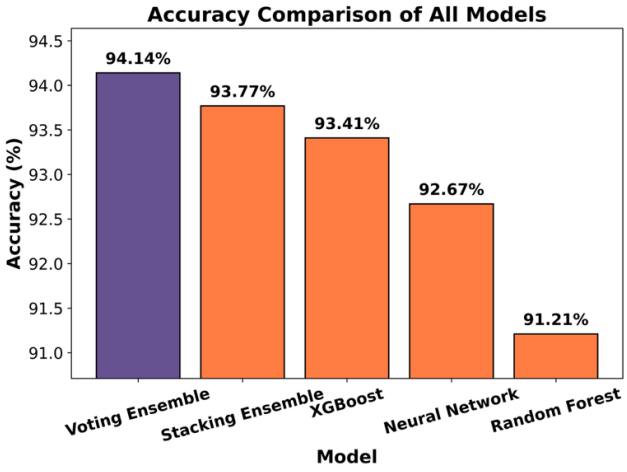

Fig. 2. Model accuracy comparison.

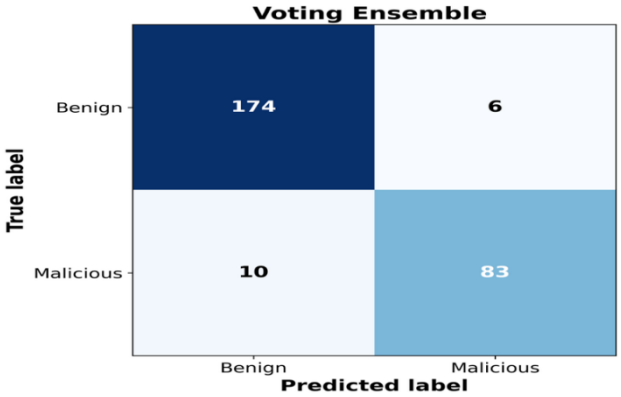

Fig. 3. Confusion matrix for voting ensemble.

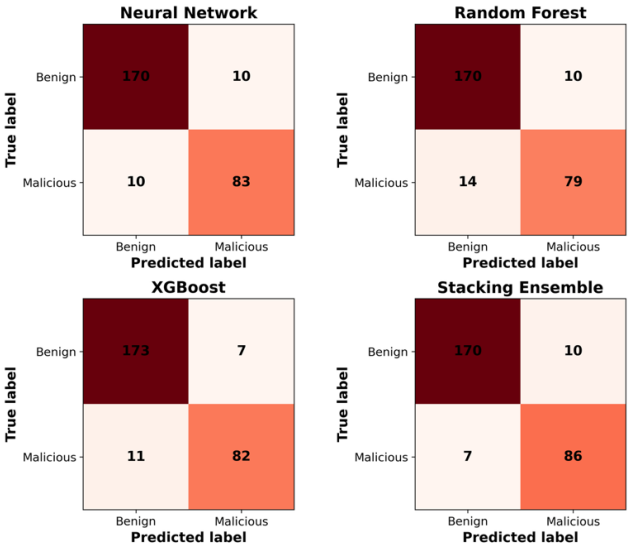

Fig. 4. Comparison of confusion matrices for different models.

E. Precision and Recall Curves

Fig. 5 illustrates the ROC (Receiver Operating Characteristic) Curves for all the models evaluated in this study. The ROC curve plots the True Positive Rate against the False Positive Rate across various thresholds, helping to visualize the trade-offs between correctly identifying benign and malicious applications. The Area Under the Curve (AUC), derived from the ROC, quantifies the model's ability to distinguish between the two classes; a higher AUC indicates better discriminatory power. The curves allow for a deeper understanding of each model's performance beyond simple accuracy by demonstrating how well the model can maintain a balance between sensitivity (correctly identifying malicious apps) and specificity (minimizing false positives) under different conditions. Models with higher AUC scores are generally more robust and better at generalizing across varying thresholds, as shown in the figure.

Additionally, Fig. 6 presents the Precision-Recall Curves for each model, highlighting how well the models balance precision and recall, particularly for the malware class. These curves further emphasize the strengths and weaknesses of each model in distinguishing between benign and malicious applications.

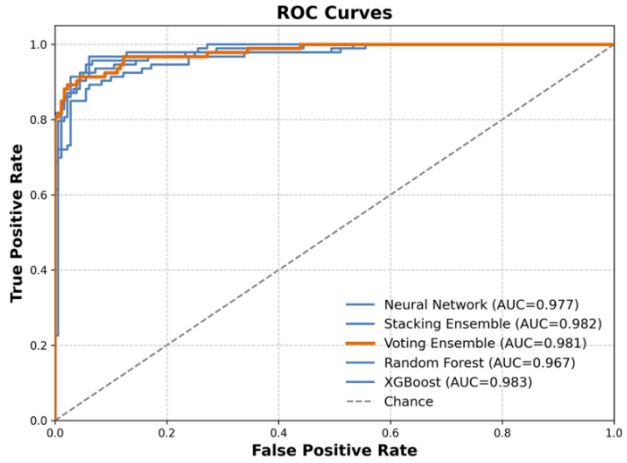

Fig. 5. ROC curves of all the models.

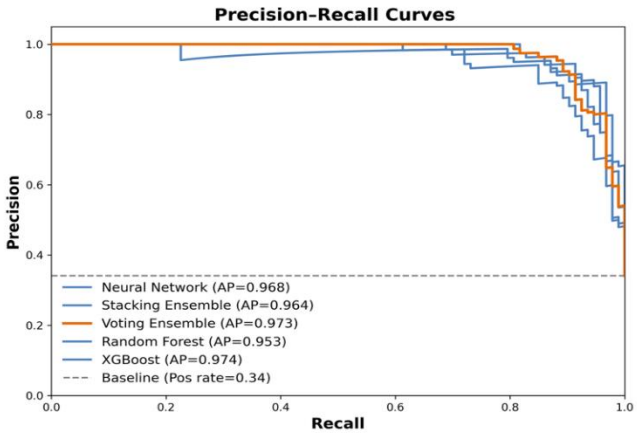

Fig. 6. Precision-recall curves.

F. Comparison with Previous Work

To further validate the effectiveness of our approach, we compared the performance of our models with recent studies in the field of Android malware detection. The following Table VI summarizes the accuracy and features used in several recent papers. While Fig. 2 presents a visual comparison of the detection accuracy across all models, highlighting the superior performance of the proposed ensemble-based approach.

TABLE VI. COMPARISON WITH EXISTING METHODS

Model	Accuracy	Dataset	Features Used	Year
Voting Ensemble (Proposed)	94.14%	CIC-AndMal2017	API Calls, Permissions, Services, Receivers, Activities	2025
Random Forest [37]	93.0%	Drebin	Permissions, Intents	2020
AdaBoost, k-NN, LR, NB [38]	91.7%	AndroZoo	Permissions, Intents,	2021
RF, KNN and DT [39]	84.14%	Custom dataset (300 benign, 183 malware)	Dalvik op-codes as grayscale images	2018
SVM[40]	91.7%	AndroZoo	Permissions, Intents	2021
StormDroid [41]	93.80%	Google Play, Contagio	Permissions, API Calls, sequences	2016

The comparative evaluation summarized in Table VI positions the proposed method within the broader context of existing research, highlighting its relevance and contribution to the field. Although direct accuracy comparisons are limited by the diversity of datasets and evaluation benchmarks across studies, the value of this work lies not solely in its high accuracy but in the methodological framework that enabled it. Unlike several high-performing approaches that depend on complex sequential architectures [41], meta-heuristic feature selection strategies [10], or hybrid deep learning frameworks [11], the proposed Voting Ensemble demonstrates that state-of-the-art performance can be achieved through a simpler and more reproducible methodology. This finding highlights an important insight: a carefully constructed ensemble of standard classifiers can serve as an efficient and practical alternative to more intricate and computationally demanding solutions, offering an optimal balance between performance, complexity, and practicality.

V. CONCLUSION

This study proposed a scalable framework for Android malware detection using static analysis of application features, including API calls, permissions, services, receivers, and activities. Through a comprehensive comparative analysis, we demonstrated that practical ensemble methods, particularly the Voting Ensemble, can achieve state-of-the-art accuracy (94.14%) while outperforming a more complex deep neural network. This finding is a key contribution, as it challenges the assumption that increasingly complex models are always necessary for high performance, offering a more lightweight and interpretable alternative. However, due to its reliance on static analysis, this approach is limited against advanced threats employing dynamic code loading or runtime evasion techniques. This limitation precisely defines the research gap our future work will address. The practical significance of this work lies in providing a highly effective and deployable solution for many real-world scenarios where computational resources are constrained. Future research will focus on integrating dynamic behavioral features, such as runtime API sequences and network traffic analysis, to create a hybrid detection framework. Exploring other ensemble strategies and adapting the model to detect zero-day malware through continual learning will also be critical. By building upon this foundation, we aim to develop even more resilient and adaptive security solutions capable of countering the evolving mobile threat landscape.

REFERENCES

[1] B. D. Deebak and S. O. Hwang, "Healthcare applications using blockchain with a cloud-assisted decentralized privacy-preserving framework," IEEE Transactions on Mobile Computing, 2023.

[2] M. Chaieb, M. A. Ghorab, and M. A. Saied, "Detecting Android Malware: From Neural Embeddings to Hands-On Validation with BERTroid," arXiv preprint arXiv:2405.03620, 2024.

[3] M. Gohari, S. Hashemi, and L. Abdi, "Android malware detection and classification based on network traffic using deep learning," in 2021 7th International Conference on Web Research (ICWR), 2021: IEEE, pp. 71-77.

[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.

[5] P. Faruki et al., "Android security: a survey of issues, malware penetration, and defenses," IEEE communications surveys & tutorials, vol. 17, no. 2, pp. 998-1022, 2014.

[6] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of android malware," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 26, no. 3, pp. 1-29, 2018.

[7] P. Panagiotou, N. Mengidis, T. Tsikrika, S. Vrochidis, and I. Kompatsiaris, "Host-based intrusion detection using signature-based and ai-driven anomaly detection methods," Information & Security: An International Journal, vol. 50, no. 1, pp. 37-48, 2021.

[8] I. F. Darwin, Android Cookbook: Problems and Solutions for Android Developers. " O'Reilly Media, Inc.", 2017.

[9] R. Mayrhofer, J. V. Stoep, C. Brubaker, and N. Kralevich, "The android platform security model," ACM Transactions on Privacy and Security (TOPS), vol. 24, no. 3, pp. 1-35, 2021.

[10] S. K. Smmarwar, G. P. Gupta, S. Kumar, and P. Kumar, "An optimized and efficient android malware detection framework for future sustainable computing," Sustainable Energy Technologies and Assessments, vol. 54, p. 102852, 2022.

[11] R. M. Sharma and C. P. Agrawal, "MH-DLdroid: A Meta-Heuristic and Deep Learning-Based Hybrid Approach for Android Malware Detection," Int. J. Intell. Eng. Syst, vol. 15, pp. 425-435, 2022.

[12] W. Z. Zarni Aung, "Permission-based android malware detection," International Journal of Scientific & Technology Research, vol. 2, no. 3, pp. 228-234, 2013.

[13] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance evaluation on permission-based detection for android malware," in Advances in Intelligent Systems and Applications-Volume 2: Proceedings of the International Computer Symposium ICS 2012 Held at Hualien, Taiwan, December 12–14, 2012, 2013: Springer, pp. 111-120.

[14] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in 2012 Seventh Asia joint conference on information security, 2012: IEEE, pp. 62-69.

[15] Y. Nishimoto, N. Kajiwara, S. Matsumoto, Y. Hori, and K. Sakurai, "Detection of android api call using logging mechanism within android framework," in International Conference on Security and Privacy in Communication Systems, 2013: Springer, pp. 393-404.

[16] P. P. Chan and W.-K. Song, "Static detection of Android malware by using permissions and API calls," in 2014 International Conference on Machine Learning and Cybernetics, 2014, vol. 1: IEEE, pp. 82-87.

[17] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," IEEE Transactions on Information Forensics and Security, vol. 9, no. 11, pp. 1869-1882, 2014.

[18] H. S. Anderson and P. Roth, "Ember: an open dataset for training static pe malware machine learning models," arXiv preprint arXiv:1804.04637, 2018.

[19] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in 2017 12th International Conference on Malicious and Unwanted Software (MALWARE), 2017: IEEE, pp. 31-38.

[20] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," in Proceedings of the 2014 ACM SIGSAC conference on computer and communications security, 2014, pp. 1105-1116.

[21] V. Regard, "Studying the effectiveness of dynamic analysis for fingerprinting Android malware behavior," ed, 2019.

[22] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection," in Proceedings of the 9th ACM symposium on Information, computer and communications security, 2014, pp. 447-458.

[23] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216-3225, 2018.

[24] S. Bhagwat and G. P. Gupta, "Android malware detection using hybrid meta-heuristic feature selection and ensemble learning techniques," in International conference on advances in computing and data sciences, 2022: Springer, pp. 145-156.

[25] O. A. Alzubi, J. A. Alzubi, A. M. Al-Zoubi, M. A. Hassonah, and U. Kose, "An efficient malware detection approach with feature weighting based on Harris Hawks optimization," Cluster Computing, pp. 1-19, 2022.

[26] I. K. Khan, H. Daud, N. Zainuddin, and R. Sokkalingam, "Standardizing reference data in gap statistic for selection optimal number of cluster in K-means algorithm," Alexandria Engineering Journal, vol. 118, pp. 246-260, 2025.

[27] I. K. Khan et al., "Numerical solution by kernelized rank order distance (KROD) for non-spherical data conversion to spherical data," in AIP Conference Proceedings, 2024, vol. 3123, no. 1: AIP Publishing LLC, p. 020011.

[28] I. K. Khan et al., "Standardization of expected value in gap statistic using Gaussian distribution for optimal number of clusters selection in K-means," Egyptian Informatics Journal, vol. 30, p. 100701, 2025.

[29] M. Iqbal et al., "A modified basis of cubic B-spline with free parameter for linear second order boundary value problems: Application to engineering problems," Journal of King Saud University-Science, vol. 36, no. 9, p. 103397, 2024.

[30] A. M. Abdussamad and A. Inayat, "Addressing limitations of the K-means clustering algorithm: Outliers, non-spherical data, and optimal cluster selection," AIMS Math, vol. 9, no. 9, pp. 25070-25097, 2024.

[31] I. K. Khan et al., "Optimal Cluster Determination in K-Means Using Gap Statistic Analysis Across Diverse Datasets," European Journal of Statistics, vol. 5, pp. 8-8, 2025.

[32] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, "MAPAS: a practical deep learning-based android malware detection system," International Journal of Information Security, vol. 21, no. 4, pp. 725-738, 2022.

[33] S. Fallah and A. J. Bidgoly, "Android malware detection using network traffic based on sequential deep learning models," Software: Practice and Experience, vol. 52, no. 9, pp. 1987-2004, 2022.

[34] V. Kouliaridis and G. Kambourakis, "A comprehensive survey on machine learning techniques for android malware detection," Information, vol. 12, no. 5, p. 185, 2021.

[35] M. Dhalaria and E. Gandotra, "Android malware detection techniques: A literature review," Recent Patents on Engineering, vol. 15, no. 2, pp. 225-245, 2021.

[36] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in 2018 International Carnahan conference on security technology (ICCST), 2018: IEEE, pp. 1-7.

[37] E. Odat and Q. M. Yaseen, "A novel machine learning approach for android malware detection based on the co-existence of features," IEEE Access, vol. 11, pp. 15471-15484, 2023.

[38] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in Proceedings of the 13th international conference on mining software repositories, 2016, pp. 468-471.

[39] F. M. Darus, N. A. A. Salleh, and A. F. M. Ariffin, "Android malware detection using machine learning on image patterns," in 2018 Cyber Resilience Conference (CRC), 2018: IEEE, pp. 1-2.

[40] S. K. Dash et al., "Droidscribe: Classifying android malware based on runtime behavior," in 2016 IEEE Security and Privacy Workshops (SPW), 2016: IEEE, pp. 252-261.

[41] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "Stormdroid: A streaminglized machine learning-based system for detecting android malware," in Proceedings of the 11th ACM on Asia conference on computer and communications security, 2016, pp. 377-388.