COMPUTOLOGY: Journal of Applied Computer Science and Intelligent Technologies
Volume 5, Issue 1, Jan-Jun 2025, pp. 1-23
DOI: 10.17492/computology.v5i1.2501
https://www.journalpressindia.com/cjacsit
© 2025 Journal Press India

GENDroid: A Optimized Hybrid Android Malware Detection Framework via Multimodal Feature Fusion and Genetic Algorithm

*Ankit Singh**

ABSTRACT

The rapid growth of Android applications has been accompanied by a surge in malicious apps that exploit system permissions and static components to bypass conventional security defenses. While static analysis has proven useful for early malware detection, many existing approaches fall short due to limited feature representation and a lack of adaptability to evolving threats. This paper introduces GENDroid, a novel framework that enhances Android malware detection by combining multi-modal static feature fusion with evolutionary optimization. GENDroid integrates diverse features—including permissions, API calls, and intent filters—into a unified representation, enabling a deeper understanding of application behavior. To optimize both feature selection and classifier performance, a Genetic Algorithm (GA)-driven strategy is employed, allowing the system to evolve and adapt automatically. The optimized feature set is used to train an ensemble of machine learning classifiers on a comprehensive dataset comprising both real-world and synthetic Android applications. Experimental results demonstrate that GENDroid delivers high detection accuracy, significantly reduces false positives, and remains robust against adversarial variants. Its modular design also allows for the seamless integration of additional static or behavioral features. By intelligently combining feature diversity with adaptive learning, GENDroid provides a practical and scalable solution for Android malware detection, effectively addressing the limitations of traditional static analysis techniques.

Keywords: *Android Malware Detection; Static Analysis; Multi-Modal Feature Fusion; Permission-to-Exploitation Mapping; Genetic Algorithms; Feature Selection; Machine Learning; Ensemble Learning; Adversarial Robustness; APK Analysis.*

1.0 Introduction

The widespread use of Android-powered products has significantly altered the mobile ecosystem, allowing millions of applications to deliver a wide range of features to users around the world.

**Student, Department of Computer Engineering, NIT-Kurukshetra, Haryana, India*
(E-mail: emailto.ankit123@gmail.com)

Since Android dominates the global market for mobile operating systems, hackers have found it to be a profitable target due to its open source nature, adaptability, and wide use. Serious security and privacy issues have been raised by the increase in malicious programs for Android, which have an effect on both individual users and business systems.

1.1 Background and motivation

Conventional malware detection methods, such run-time behavior analysis and signature-based scanning, frequently find it difficult to stay up with the quickly changing threat ecosystem. Signature-based methods are useless against obfuscated versions and zero-day attacks since they require prior knowledge of malware patterns. However, because they depend on runtime execution and monitoring settings, dynamic analysis techniques—while effective—tend to be resource-intensive and less scalable. In order to get over these restrictions, this study suggests GENDroid, a fresh static malware detection framework that incorporates two significant breakthroughs. In order to provide a more comprehensive semantic representation of Android apps, it first makes use of multi-modal feature fusion, which combines permissions, API call patterns, and intent filters. Second, it uses Genetic Algorithms (GAs) to optimize the classifier and perform automatic feature selection. This enables the system to adapt dynamically and sustain high detection performance as threat patterns change. GENDroid provides a scalable, intelligent, and reliable method for detecting Android malware thanks to these improvements.

1.2 Security gaps

The open-source nature of the Android operating system has contributed significantly to its widespread adoption by enabling a large and diverse developer community to build a wide range of applications. This flexibility allows device manufacturers and software developers to customize the platform, offering tailored user experiences and innovative features that further fuel its popularity. However, the same openness that fosters innovation also introduces critical security concerns. Since the source code is publicly accessible, malicious actors can analyze it to identify vulnerabilities and craft sophisticated exploits. Moreover, Android's support for third-party application marketplaces—many of which lack rigorous security vetting—provides an additional vector for the distribution of malicious software.

1.3 Objective

This work presents the design and implementation of a scalable and modular malware detection framework for Android, with the following core objectives:

- To develop a static analysis-based detection system that combines diverse sources of information extracted from Android application packages (APKs), such as manifest declarations and code-level components.
- To improve the semantic richness of extracted features by incorporating a structured mapping between static indicators and potential exploitation behaviors.
- To apply Genetic Algorithms (GAs) for: Automatic selection of the most discriminative feature subsets Hyperparameter tuning of classifiers to maximize detection accuracy.
- To construct an optimized ensemble of machine learning models capable of robust malware identification across varied samples.
- To evaluate the framework's effectiveness using real-world and adversarial datasets, employing multiple performance metrics including accuracy, precision, recall, F1-score, and AUC.

The proposed framework is designed to operate exclusively through static analysis, without requiring code execution or runtime instrumentation. The detection pipeline functions offline and consists of APK decompilation, feature extraction, evolutionary optimization, and supervised classification. While dynamic analysis and real-time deployment are outside the scope of this study, the architecture has been intentionally built with modularity in mind—facilitating future integration of dynamic features or behavioral models as extensions.

2.0 Related Work

The exponential growth of Android applications has made mobile platforms increasingly vulnerable to malware attacks. With the Android operating system accounting for a significant share of the global mobile market, adversaries continue to exploit its open-source architecture, app distribution model, and user permission system. In response, the research community has developed a wide range of Android malware detection frameworks, primarily categorized as static, dynamic, and hybrid approaches. This chapter presents a detailed review of recent studies in static malware detection, the role of machine learning, multi-feature analysis, and optimization strategies that inform the design of the proposed GENDroid framework.

Wajahat *et al.* (2024) demonstrated that selecting a subset of meaningful permissions using Recursive Feature Elimination (RFE) and SHAP values significantly improved detection accuracy while reducing feature space, achieving an F1-score above 0.99. Gupta *et al.* (2024) leveraged rough set theory to eliminate redundant permission features and prioritize those most indicative of malicious behavior. Odat & Yaseen (2023) constructed a co-occurrence matrix of permissions and API calls and utilized FP-growth algorithms to identify strong feature associations. Their method achieved a detection rate of 98%, outperforming permission-only models. Mahindru & Sangal (2021), through MLDroid, used a combination of APIs, app metadata, and permissions. Their results confirmed that integrating diverse feature types with ensemble classifiers like Random Forest and Gradient Boosted Trees outperformed models trained on single-feature sets.

Vu & Jung (2021) developed AdMat, a model that represents Android app features as adjacency matrices and feeds them into a Convolutional Neural Network (CNN). Their model achieved a high detection rate of 98.26%, demonstrating robustness even with smaller training datasets. Surendran *et al.* (2020) employed graph signal processing on system call graphs. By using only 16-dimensional feature vectors and training with Random Forests and Decision Trees, the model achieved 99% accuracy while maintaining low computational overhead.

Liu *et al.* (2024) proposed SeGDroid, which constructs sensitive function call graphs from decompiled apps. By pruning irrelevant edges and nodes, they enhanced both malware detection (98). Mehtab *et al.* (2020) presented AdDroid, a rule-based engine using a handcrafted feature set and an Adaboost-based ensemble classifier. The system reached 99.11. Almarshad *et al.* (2023) addressed the data scarcity problem using a Siamese network for few-shot learning. By coupling this with conventional classifiers, they achieved 98.9% accuracy on the Drebin dataset, illustrating its applicability in low-data scenarios.

Alani & Awad (2022) introduced PAIRED, a framework that combined SHAP values and recursive elimination to identify the most impactful permission features. The model achieved accuracy above 98% using tree-based ensemble classifiers.

Alkahtani & Aldhyani (2022) employed correlation-based feature filtering followed by classification using SVM, LSTM, and CNN-LSTM models. Remarkably, their SVM model attained 100% accuracy on the CI-CAndMal2017 dataset, while deep learning models surpassed 98%, underscoring the importance of filtering and classifier alignment. Although prior research in Android malware detection has introduced a variety of machine learning-based approaches, several key limitations continue to hinder the effectiveness, adaptability, and scalability of these systems. The following challenges, observed across state-of-the-art studies, motivate the need for a more robust solution:

- *Limited Dataset Scale and Currency:* Many studies rely on small or outdated datasets, which can reduce the generalizability and real-world relevance of their models. This hinders the system's ability to detect newly emerging or mutated malware variants.
- *Inefficient Feature Representation:* Several approaches either underutilize key static attributes or fail to apply intelligent feature reduction, resulting in unnecessarily large feature spaces that may include redundant or irrelevant dimensions.

- *Scalability Bottlenecks:* The architectural design of many frameworks lacks support for modular expansion or efficient processing of large-scale datasets, which is critical given the continuous growth of Android applications in the wild.
- *Incremental Learning Capabilities:* Most existing models are trained in a batch mode using classical machine learning methods, which require full retraining to incorporate new data. This leads to inefficiencies in model maintenance and delays in adapting to evolving threat landscapes.

3.0 Proposed Methodology

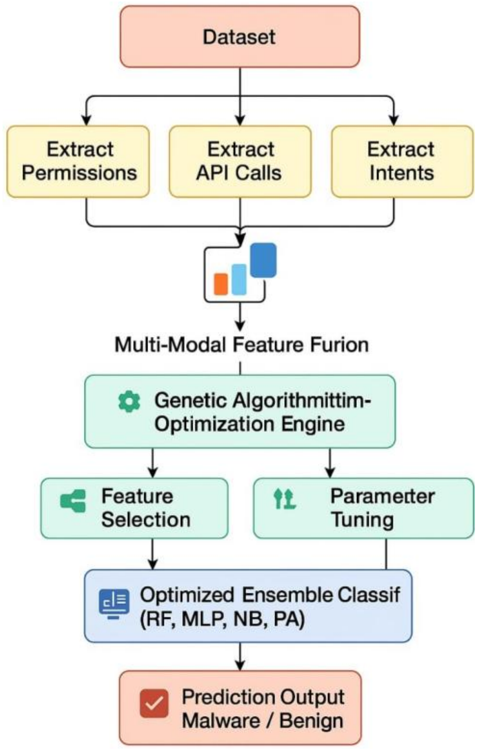
With mobile applications deeply integrated into users' lives, permissions are often granted without a clear understanding of their security implications. Malicious Android applications exploit these permissions to gain unauthorized access to sensitive data or functionalities, leading to severe privacy and security threats. To address this challenge, GENDroid presents a multi-modal, static-analysis-based malware detection framework. It integrates diverse static features—primarily permissions and app metadata—using a hybrid of semantic transformation and machine learning techniques. The framework enhances detection accuracy and interpretability by combining Permission-to-Exploitation Mapping (PEM) with Genetic Algorithm-based optimization and an ensemble of classifiers.

- Input from the Drebin dataset, a benchmark labelled dataset of Android applications (both benign and malicious).
- Extraction of static features, including permissions, API usage, and other manifest-level metadata.
- Multi-modal feature transformation, where permission features are semantically mapped into high-level exploitation categories via the PEM dictionary, and combined with other static features.
- Optimization via Genetic Algorithm, which selects the most relevant features and tunes hyperparameters (e.g., number of estimators, tree depth) for Random Forest.
- Training of an ensemble of classifiers, including Random Forest (RF), Multi-Layer Perceptron (MLP), Naive Bayes (NB), and Passive Aggressive (PA); PA supports incremental learning for scalability.
- Final malware classification through majority voting, based on consensus among the classifiers.
- Evaluation of detection performance using metrics such as Accuracy, Precision, Recall, F1 Score, and AUC.

By fusing multiple feature modalities and leveraging evolutionary learning, GENDroid achieves efficient, scalable, and interpretable Android malware detection.

Figure 1 illustrates the flow chart of the architecture of the GENDroid Malware Detection framework.

Figure 1: Flow Chart of the GENDroid


Dataset
Extract Permissions
Extract API Calls
Extract Intents
Multi-Modal Feature Furion
Genetic Algorithmittim-Optimization Engine
Feature Selection
Parameter Tuning
Optimized Ensemble Classif (RF, MLP, NB, PA)
Prediction Output Malware / Benign

Source: Author's own

3.1 Dataset collection and overviews

A robust and reliable dataset is fundamental to the development and evaluation of any malware detection framework. For this purpose, the GENDroid framework utilizes the Drebin dataset, a publicly available and widely recognized Android malware dataset introduced by Arp *et al.* This dataset offers a comprehensive static analysis view of Android applications, encompassing a broad range of behavioral indicators, and serves as a standard benchmark in academic and industrial research.

- *Dataset Source and Composition:* The Drebin dataset comprises 5,560 Android application samples, including 5,156 malicious apps, collected from multiple third-party markets and verified using VirusTotal.404 benign apps, primarily sourced from the Google Play Store. Each application in the dataset has been statically analyzed to extract various features from its APK components (e.g., AndroidManifest.xml, DEX code). These features span multiple categories like PermissionsIntent, filtersAPI, callsNetwork, addressesHardware, componentsApp, components (activities, services, etc.) This extensive multi-source feature extraction results in a sparse binary feature matrix with over 123,000 unique feature keys, which represent the presence or absence of specific attributes in each app.

- *Multi-Modal Feature Representation in GENDroid:* While the original Drebin dataset includes a large and heterogeneous feature space, GENDroid adopts a multi-modal static feature integration approach to construct a more semantically meaningful and compact representation. The following modalities are incorporated: Raw Permission Features: Direct binary indicators of Android permissions declared in the manifest file. Non-Permission Static Features: Additional manifest-level metadata not categorized under permissions, such as services, actions, and receivers. These provide auxiliary signals about the app's behavior. PEM-based Semantic Features: To enhance semantic interpretability, permissions are transformed using a Permission-to-Exploitation Mapping (PEM) dictionary. This dictionary clusters permissions into high-level behavioral exploitation categories as shown in Table 1. Table 1 presents the mapping of Android permissions to high-level malware exploitation techniques. This mapping is adapted and extended from the PermGuard framework permguard (Ahmad, 2021), which categorizes permissions based on common malicious behaviors observed in Android malware. Additional categories were included in our work by analyzing uncategorized features and associating them with relevant exploitation patterns using Android developer documentation and malware behavior reports.
- *Data Preprocessing Workflow:* The preprocessing pipeline is as follows:
    - Label Encoding: Malware samples are labeled as 1, benign as 0.
    - Missing Value Handling: Any ambiguous values are imputed as zeros.
    - PEM Feature Engineering: A new PEM-transformed matrix is generated by mapping requested permissions to their corresponding exploitation categories.
    - Feature Fusion: The PEM matrix is combined with non-permission static features for a multi-modal input vector.
    - Normalization: All features are scaled using Min-Max normalization for compatibility with distance- and tree-based algorithms.
    - Train-Test Split: The dataset is split in an 80:20 stratified manner, ensuring balanced representation of malware and benign samples in both sets

Table 1: Mapping of Malware Techniques to Android Permissions

Malware Technique	**Android Permissions**
Remote Command Execution	INTERNET, BIND DEVICE ADMIN, PROCESS OUTGOING CALLS, READ SMS, SEND SMS, READ CALL LOG, READ LOGS, RECORD AUDIO, CAMERA, MODIFY PHONE STATE, WRITE EXTERNAL STORAGE
Rootkit Installation	INTERNET, BIND DEVICE ADMIN, READ EXTERNAL STORAGE, WRITE EXTERNAL STORAGE, READ LOGS, MODIFY PHONE STATE, RECORD AUDIO, CAMERA, SYSTEM OVERLAY WINDOW, ACCESS SUPERUSER
Exploit Delivery	INTERNET, WRITE EXTERNAL STORAGE, READ EXTERNAL STORAGE, INSTALL PACKAGES, REQUEST INSTALL PACKAGES
Data Exfiltration	INTERNET, READ EXTERNAL STORAGE, WRITE EXTERNAL STORAGE, ACCESS NETWORK STATE
Credential Theft	INTERNET, READ EXTERNAL STORAGE, WRITE EXTERNAL STORAGE, RECORD AUDIO, CAMERA, READ CONTACTS, GET ACCOUNTS
Screen Logging	INTERNET, READ EXTERNAL STORAGE, WRITE EXTERNAL STORAGE, RECORD AUDIO, CAMERA
Keylogging	INTERNET, READ EXTERNAL STORAGE, WRITE EXTERNAL STORAGE, RECORD AUDIO, CAMERA, READ LOGS, READ SMS, RECEIVE SMS, READ CONTACTS
Audio Surveillance	RECORD AUDIO, INTERNET, READ EXTERNAL STORAGE, WRITE EXTERNAL STORAGE
Social Engineering Attack	READ CONTACTS, SEND SMS, READ SMS, RECEIVE SMS, WRITE EXTERNAL STORAGE
GPS Spoofing	ACCESS ASSISTED GPS, ACCESS GPS
Device Bricking	DEVICE POWER
Call Interception	READ PHONE STATE, READ CALL LOG, PROCESS OUTGOING CALLS
Network Traffic Interception	INTERNET, ACCESS NETWORK STATE, CHANGE NETWORK STATE
Device Lockout	MODIFY PHONE STATE, WRITE SECURE SETTINGS, SHUTDOWN
Browser Hijacking	INTERNET, WRITE HISTORY BOOKMARKS, WRITE SECURE SETTINGS
System Settings Modification	WRITE SETTINGS, WRITE SECURE SETTINGS, CHANGE CONFIGURATION
File System Manipulation	READ EXTERNAL STORAGE, WRITE EXTERNAL STORAGE, ACCESS CACHE FILESYSTEM, WRITE INTERNAL STORAGE
Camera Hijacking	CAMERA, RECORD AUDIO, RECORD VIDEO
App Installation without User	INSTALL PACKAGES, REQUEST INSTALL PACKAGES, INSTALL

Consent	SHORTCUT, DOWNLOAD WITHOUT NOTIFICATION
Location Tracking	ACCESS LOCATION
Contact Information Theft	READ CONTACTS, WRITE CONTACTS
Browser History Theft	READ HISTORY BOOKMARKS, ACCESS DOWNLOAD MANAGER
System Settings Modification	WRITE SETTINGS, WRITE SECURE SETTINGS, CHANGE CONFIGURATION
Task Manipulation	REORDER TASKS, GET TASKS
Bluetooth Hijacking	BLUETOOTH, BLUETOOTH ADMIN
WiFi Network Hijacking	ACCESS WIFI STATE, CHANGE WIFI STATE
USB Debugging Exploitation	WRITE SECURE SETTINGS
Screen Overlay Attack	SYSTEM ALERT WINDOW
Sim Card Manipulation	READ PHONE STATE, WRITE SETTINGS
Ad Fraud	ACCESS NETWORK STATE, INTERNET
Account Information Theft	GET ACCOUNTS, MANAGE ACCOUNTS, AUTHENTICATE ACCOUNTS
Certificate Manipulation	READ EXTERNAL STORAGE
Runtime Environment Manipulation	MODIFY AUDIO SETTINGS

Source: Author's own

3.2 Genetic algorithm-based optimization

The GENDroid framework integrates a Genetic Algorithm (GA) to perform dual-level optimization, addressing both feature selection and classifier hyperparameter tuning. This evolutionary approach enhances the performance of malware detection by systematically exploring the solution space for the most discriminative features and optimal model configurations. Traditional static feature selection techniques often struggle with high-dimensional Android malware datasets, where irrelevant or redundant features can degrade classifier performance. Similarly, manually tuning hyperparameters for ensemble classifiers like Random Forest or MLP is computationally expensive and suboptimal. GAs offers a biologically-inspired mechanism capable of navigating complex, nonlinear search spaces, making them ideal for this dual optimization task.it involves different steps which are as follows:

- *Chromosome Representation:* In GENDroid, each chromosome encodes: A binary feature mask representing whether a specific feature is selected (1) or not (0). Discrete values for the hyperparameters of the Random Forest classifier, specifically:
  - n-estimators: Number of decision trees
  - max-depth: Depth of individual trees

This hybrid representation allows the GA to evolve both the feature subset and the classifier's configuration simultaneously.

- *Fitness Evaluation:* The fitness function evaluates each chromosome based on the F1-Score achieved by a Random Forest classifier using the encoded feature subset and hyperparameters. The fitness score balances precision and recall, making it suitable for imbalanced malware detection datasets.
- *Genetic Operators:* the following operators are used for the genetic algorithm process.
  - Selection: Tournament selection is employed to retain high-fitness individuals.
  - Crossover: Two-point crossover enables exchange of feature and hyperparameter genes between parent chromosomes.
  - Mutation: Bit-flip mutation is applied to the feature mask, and random re-sampling is used for hyperparameters with a predefined mutation probability.
- *Evolutionary Process:* The GA is executed over multiple generations (e.g., 10–30) with a defined population size (e.g., 20). During each generation:
  - The current population is evaluated using the fitness function.
  - Offspring are generated through crossover and mutation.
  - The next generation is formed by selecting the fittest individuals.

This process gradually converges towards a feature subset and hyperparameter set that yields high malware classification performance and the best chromosome (solution) is selected.

3.3 Ensemble-based classification

The final stage of the GENDroid malware detection framework involves training an ensemble of machine learning classifiers on the optimized feature subset produced by the Genetic Algorithm. This multi-model architecture enhances detection accuracy and resilience by leveraging the diverse decision-making capabilities of each individual model.

- *Role of the Ensemble in GENDroid:* The rationale for using an ensemble of classifiers is grounded in the principle of classifier diversity—different models learn different patterns and biases from the data. By combining their outputs, GENDroid can generalize better across a wide range of malware behaviors, making the detection system more robust against obfuscation and zero-day threats.
- *Classifiers Used in GENDroid:* Each classifier in the ensemble has been carefully selected for its unique strengths and complementary behavior:
  - Random Forest (RF): This classifier is a bagging ensemble of decision trees. It operates by training multiple trees on different bootstrap samples of the training data and aggregating their predictions. In GENDroid, RF benefits from the hyperparameter tuning performed by the GA, including optimal selection of n-estimators and max-depth. RF is leveraged for its strong ability to handle high-dimensional feature spaces and interpret feature importance. It forms the backbone of the ensemble due to its high standalone accuracy and resistance to overfitting.

- Multi-Layer Perceptron (MLP): MLP is a feed forward neural network that includes one or more hidden layers with nonlinear activation functions. It learns complex, non-linear mappings between input features and output labels. The MLP captures deep correlations between permission semantics, API usage, and other metadata. Its strength lies in modeling complex malware signatures that linear models may overlook.
- Gaussian Naive Bayes (NB): NB is a simple probabilistic model based on Bayes' theorem with the assumption of feature independence. Despite this assumption, it is highly effective and efficient for text-like or sparse datasets. NB adds diversity to the ensemble. It is particularly useful when permissions or APIs appear independently or in rare combinations. Its lightweight nature ensures fast inference.
- Passive Aggressive Classifier (PA): PA is an online learning model that updates its decision boundary incrementally as it sees new batches of data. It is ideal for dynamic environments where new malware families continuously emerge. PA introduces incremental learning capability to the system. It supports continuous adaptation by learning from streaming or chunked data batches. This makes GENDroid scalable and suited for real-time detection scenarios.

- *Classifier Training Process:* Once the Genetic Algorithm has selected the best feature subset and hyper-parameter configurations, each classifier is trained on the optimized data:
  - The RF, MLP, and NB classifiers are trained using the entire training set.
  - The PA classifier is trained incrementally using mini-batches of the training set. This simulates real-world deployment where new apps are analyzed in small groups over time.

All classifiers share the same feature subset, ensuring consistency in learning patterns across the models. Their diversity arises from differences in model architecture, training objectives, and decision boundaries.

- *Decision Aggregation through Voting:* After training, GENDroid employs a majority voting mechanism to aggregate predictions from the ensemble. Each classifier *outputs* a label (benign or malicious), and the majority class is assigned as the final prediction. This strategy ensures:
  - Consensus-driven decision-making
  - Reduced classifier-specific bias
  - Improved resilience against noisy or adversarial inputs

By adopting this ensemble strategy, GENDroid balances accuracy, speed, and robustness, making it effective for deployment in diverse Android malware detection environments.

4.0 Result and Discussion

In this section, we analyze the experimental outcomes of the proposed GENDroid framework, which aims to detect Android malware using multi-modal static features and evolutionary optimization. GENDroid incorporates feature fusion from permissions, API calls, and intents, along with genetic algorithm-based feature selection and classifier hyperparameter tuning. The performance of individual classifiers—including Random Forest (RF), Multi-Layer Perceptron (MLP), Naive Bayes (NB), and Passive Aggressive (PA)—is evaluated and compared with the ensemble classification strategy that combines their predictions through majority voting. The results demonstrate how GENDroid effectively reduces feature dimensionality, increases classification accuracy, and ensures generalizability. The integration of incremental learning through the PA classifier enhances adaptability to streaming or large-scale datasets, further improving the scalability of the model. The experimental findings are presented using performance metrics such as Accuracy, Precision, Recall, F1 Score, and AUC, and supported with various visualizations including ROC curves, fitness evolution plots, and feature importance graphs.

4.1 Experimental setup

The experiments for evaluating the GENDroid framework were conducted on a 64-bit Windows 11 Home (version 23H2) system powered by an AMD Ryzen 7 7730U processor with Radeon Graphics, running at 2.00 GHz. The model development and analysis were performed using Python 3.11 in a Jupyter Notebook environment, with Java version 21 employed for auxiliary tasks related to Android package (APK) processing and static analysis when necessary.

4.2 Libraries Used

The GENDroid framework was fully implemented in Python, leveraging a robust set of libraries and tools:

- Scikit-learn: Used for implementing core classifiers including Random Forest, MLP, Naive Bayes, and Passive Aggressive models, as well as for computing standard evaluation metrics such as Accuracy, Precision, Recall, F1 Score, and AUC.
- DEAP (Distributed Evolutionary Algorithms in Python): Employed to design and execute the Genetic Algorithm for optimizing feature selection and tuning classifier hyperparameters.
- Matplotlib and Seaborn: Utilized to generate comprehensive visualizations such as ROC curves, fit- ness distributions, feature importance rankings, and performance comparisons.

4.3 Performance metrics

To comprehensively evaluate the classification performance of the GENDroid framework, we utilized the following standard metrics, which are widely accepted in machine learning for binary classification problems. These metrics provide insight into different aspects of the model's effectiveness, especially in the context of imbalanced datasets like Android malware detection.

- *Accuracy:* Accuracy measures the overall correctness of the classifier by calculating the proportion of correctly classified samples among all predictions.
  $$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$
- *Precision:* Precision reflects the model's ability to correctly label malware instances without misclassifying benign apps. It is the ratio of correctly predicted malware (true positives) to all predicted malware samples.
  $$Precision = \frac{TP}{TP+FP}$$
- *Recall (Sensitivity or True Positive Rate):* Recall indicates the model's capability to identify all actual malware instances. It is the ratio of true positives to all actual positive (malicious) samples.
  $$Recall = \frac{TP}{TP+FN}$$
- *F1 Score:* The F1 Score is the harmonic mean of Precision and Recall. It balances the trade-off between the two metrics, particularly useful when the class distribution is skewed.
  $$F1 = 2 \cdot \frac{Precision \cdot Reacll}{Precision + Recall}$$
- The AUC represents the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. A higher AUC value indicates better model discrimination across various threshold settings.
  $$\text{AUC} = \int_0^1 TPR(FPR)\, dFPR$$

4.4 Model performance

The performance of the proposed GENDroid framework was evaluated through extensive experiments using the Drebin dataset. The framework integrates multi-modal feature fusion, Genetic Algorithm (GA)-based optimization, and ensemble learning, resulting in high classification performance for Android malware detection. Table 2 Shows the performance metrics for the proposed model.

Table 2: Performance Metrics for the Proposed Model

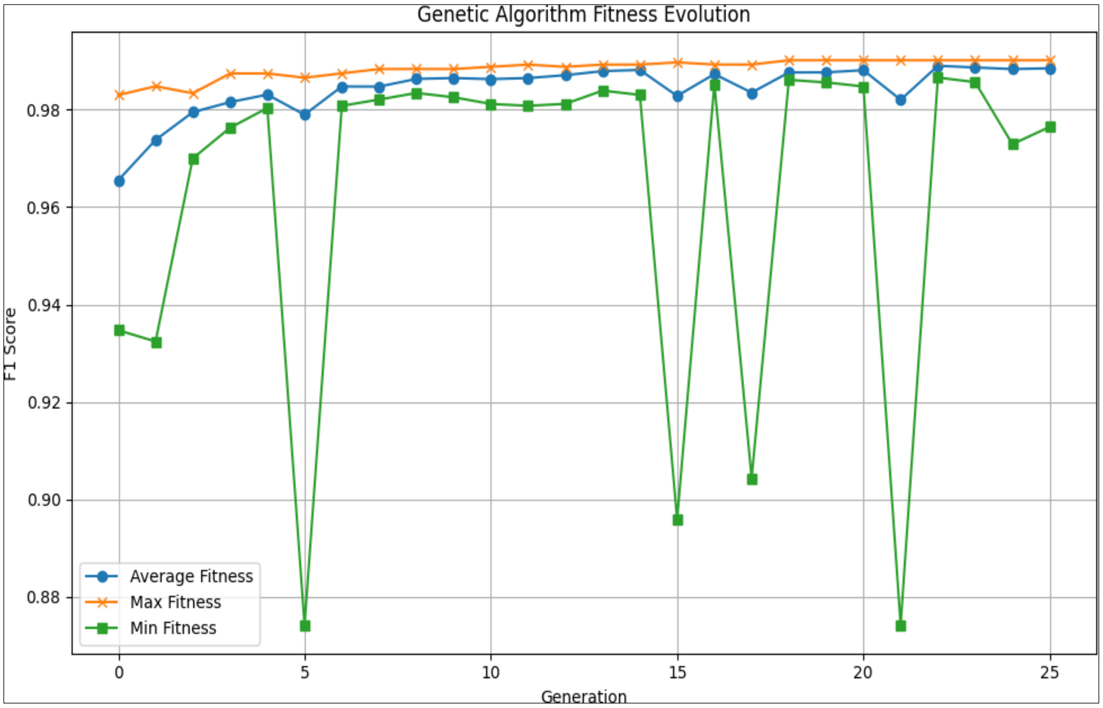
Metric	Value
Accuracy	98.62%
Precision	98.66%
Recall	99.55%
F1 Score	99.10%
AUC	99%

Source: Author's own

4.5 Analysis of visual results

Figure 2 shows the distribution of F1 scores across generations in the Genetic Algorithm. In early generations, we observe wide variability, reflecting random initial populations. However, starting from generation 5 onward, fitness values begin to stabilize and converge. This indicates the GA successfully evolves toward optimal solutions, selecting the most discriminative feature subsets and best hyperparameter combinations.

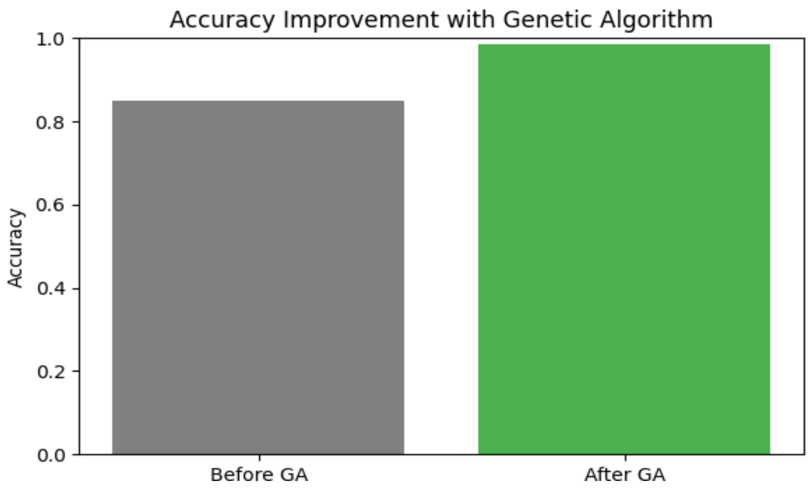
Figure 2: Fitness Distribution Across Generations



Source: Author's own

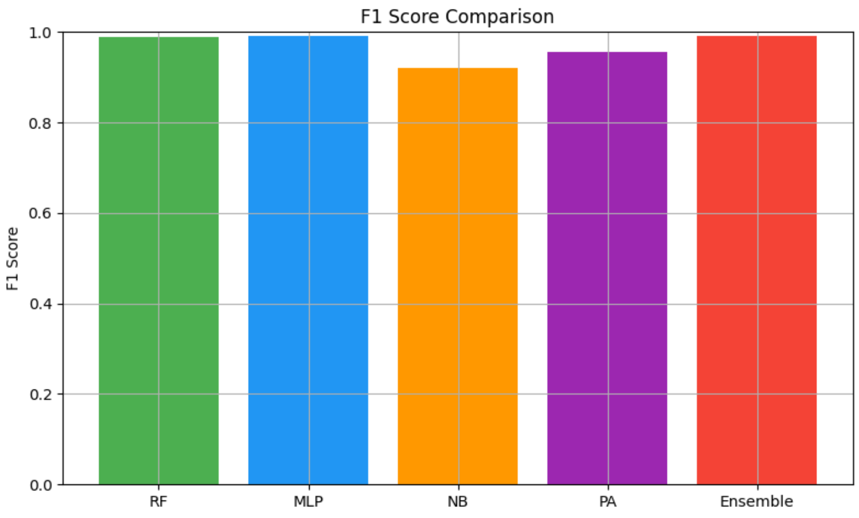
Figure 3 highlights the impact of genetic optimization. The initial classification model (without GA) yielded approximately 85% accuracy. After applying the Genetic Algorithm for both feature selection and hyperparameter tuning, accuracy increased significantly to 98.62%. This stark improvement validates the utility of evolutionary search in enhancing performance.

Figure 3: Accuracy



Source: Author's own

Figure 4: F1 Score Comparison

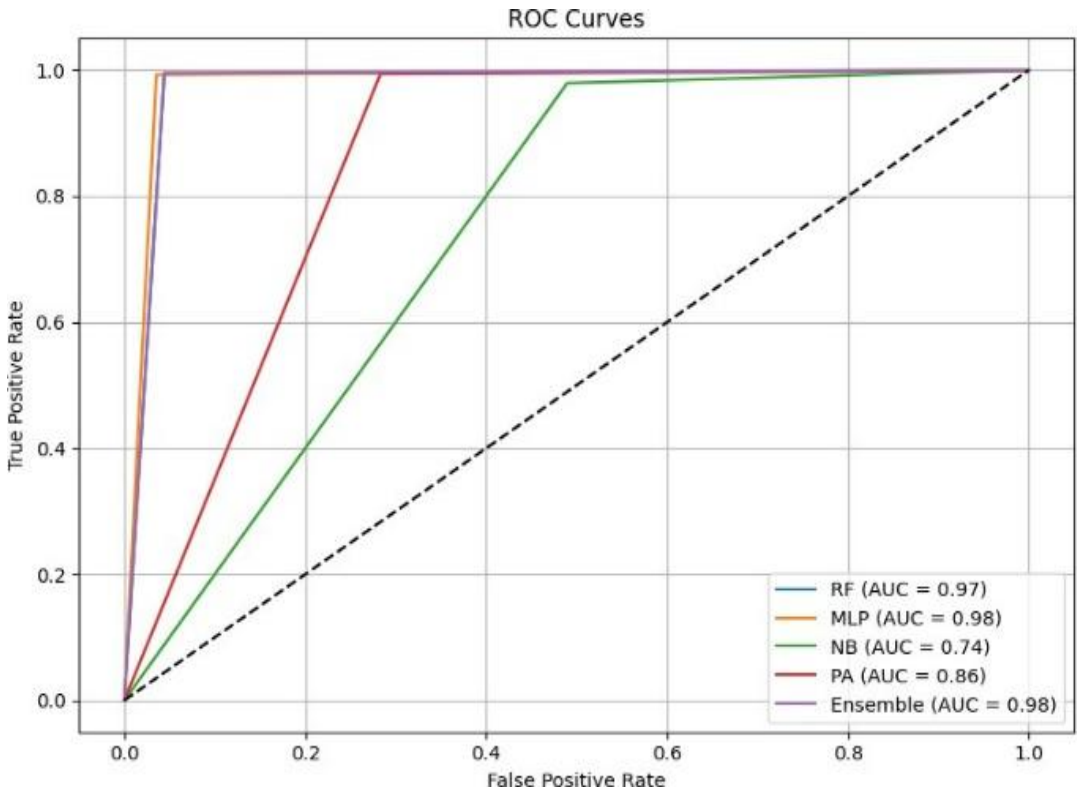


Source: Author's own

Figure 4 compares the F1 score for each classifier in the ensemble. Both RF and MLP perform exceptionally well individually (F1 0.99), whereas NB and PA, although relatively weaker, still contribute valuable perspectives in ensemble voting. The ensemble model achieves the highest F1 score, proving the effectiveness of classifier fusion.

Figure 5 presents the ROC curves for all individual classifiers and the ensemble. The ensemble classifier, RF, and MLP all achieved an AUC of approximately 0.98–0.99, indicating a strong ability to distinguish between benign and malicious apps.

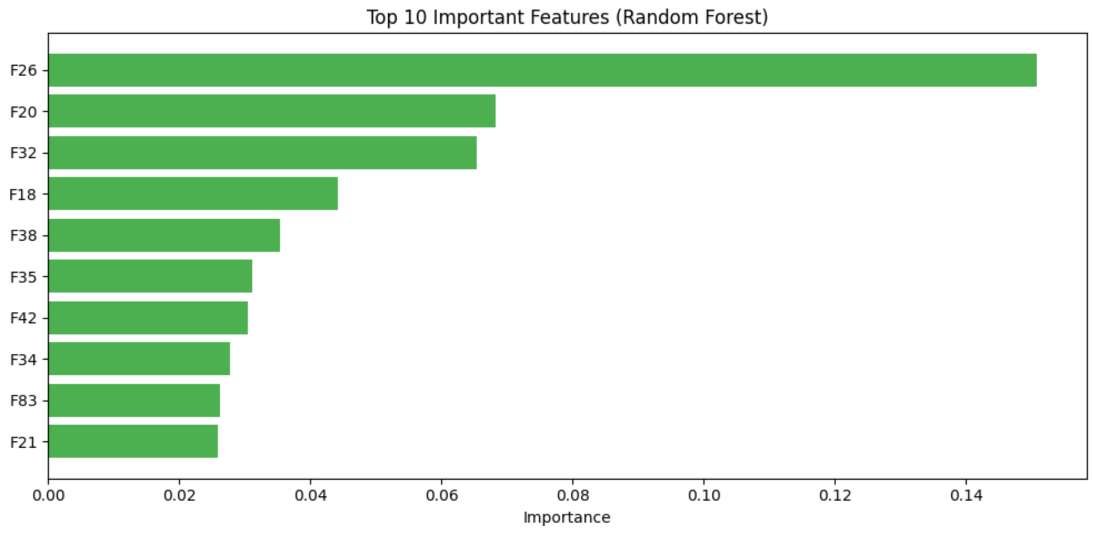
Figure 5: ROC Curve



Source: Author's own

Figure 6 displays the top 10 most influential features as determined by the Random Forest classifier. Feature F26 was the most dominant, contributing over 15% to the classification decision. This analysis provides interpretability to the model, helping security analysts understand which permissions or APIs are most often linked to malware behavior.

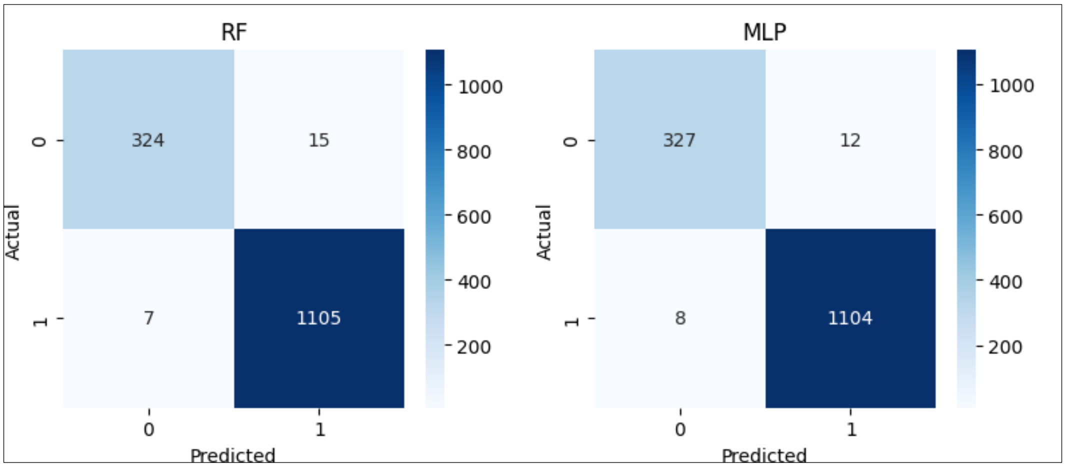
Figure 6: Feature Importance Analysis

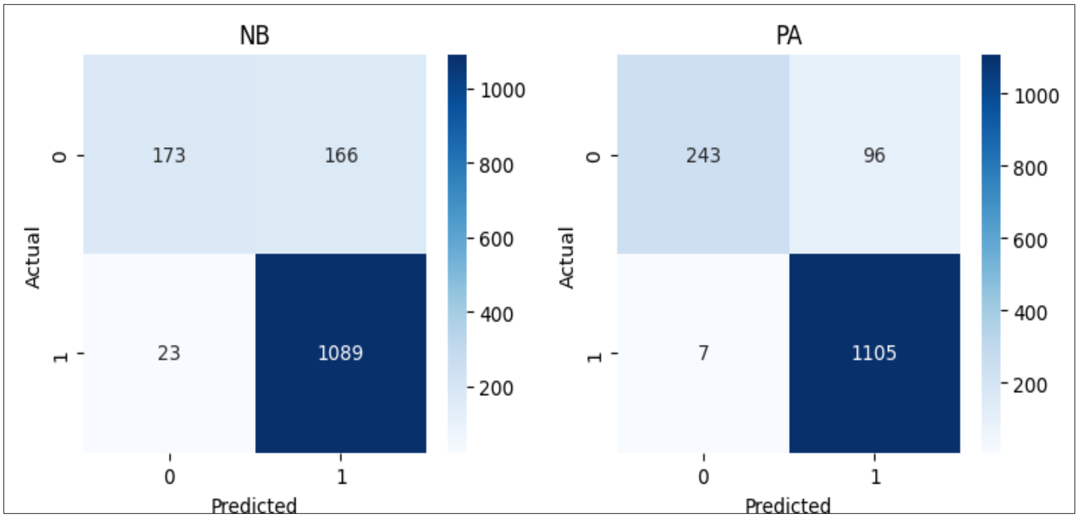


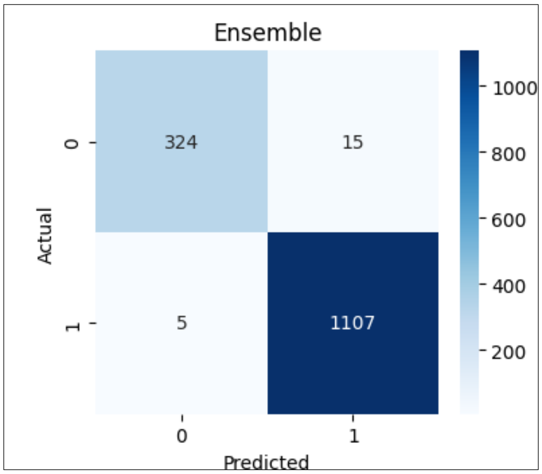
Source: Author's own

Figure 7 illustrates confusion matrices for each classifier. The ensemble model demonstrates superior performance with only 20 misclassifications out of 1,451 instances. PA, RF, and MLP also exhibit low false positives and false negatives. NB has significantly more false positives, confirming its lower reliability. The ensemble classifier's minimal error rate affirms its capacity to balance precision and recall across imbalanced datasets.

Figure 7: Confusion Matrices





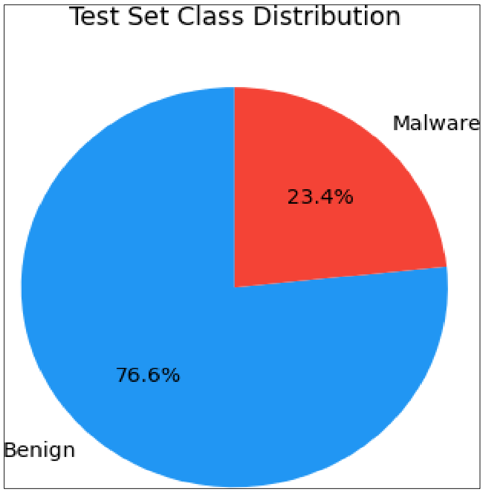


Source: Author's own

Figure 8 depicts the class distribution of the test set, which is moderately imbalanced, with benign samples comprising 76.6% and malware samples 23.4%. This imbalance necessitates models that maintain high recall to ensure malware instances are not overlooked. The high recall scores achieved by GENDroid suggest robustness against class imbalance. capacity to balance precision and recall across imbalanced datasets.

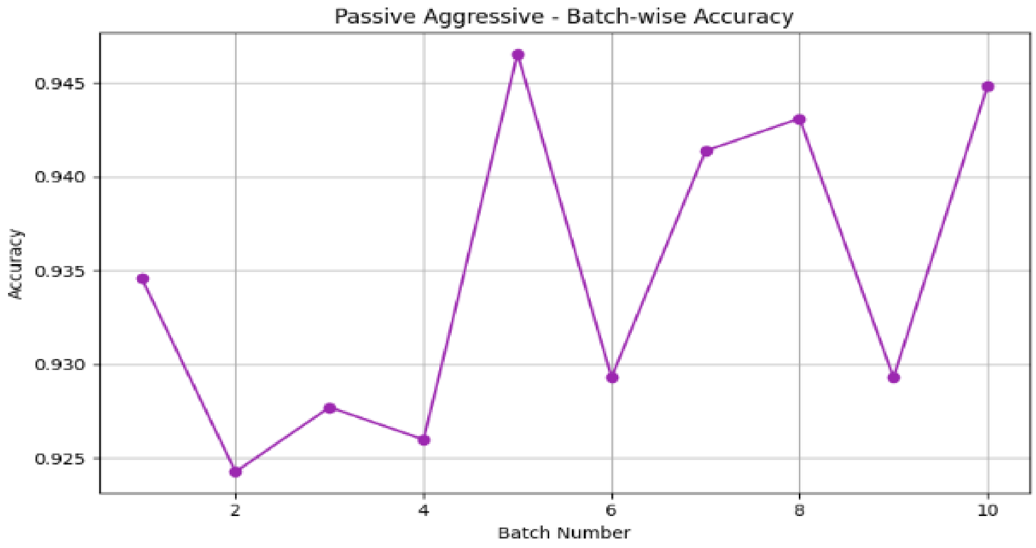
Figure 9 evaluates the performance of the Passive Aggressive (PA) classifier in incremental learning. Accuracy fluctuates across batches due to data variance but consistently remains above 92%, peaking at 94.6%. This supports the suitability of PA for real-time malware detection in dynamic Android environments where data arrives in streams.

Figure 8: Class Distribution in Test Set



Source: Author's own

Figure 9: Batch-wise Accuracy



Source: Author's own

4.6 Comparative analysis

To evaluate the effectiveness of the proposed GENDroid framework, we conducted a comparative analysis with several state-of-the-art Android malware detection techniques reported in the literature. Table 3 summarizes the performance of different models across key evaluation metrics, namely Accuracy, Precision, Recall, and F1-Score. GENDroid demonstrates superior performance across most metrics. Specifically, it achieves an accuracy of 98.62%, which is competitive with or exceeds the accuracy reported in existing models. In terms of precision (98.66%), GENDroid effectively minimizes false positives, ensuring that benign apps are rarely misclassified as malicious. This is crucial in real-world scenarios to maintain user trust and avoid unnecessary restrictions.

The recall score of 99.55% highlights GENDroid's high sensitivity in correctly identifying malicious applications, significantly outperforming traditional classifiers like Naive Bayes and certain heuristic-based techniques. This ability to accurately capture the majority of mal- ware cases makes GENDroid highly reliable in threat detection.

While methods such as Almarshad *et al.* (2023) and Liu *et al.* (2024) achieved high precision (0.989 and 0.9931 respectively), they operated on relatively smaller or im- balanced datasets. Their recall and F1 scores were also lower compared to GENDroid. The F1-score of 99.10%, which balances both precision and recall, reflects the robustness of the model across imbalanced datasets. Compared to techniques such as those by Alani & Awad (2022) and Liu *et al.* (2024), which show strong individual metrics, GENDroid provides a consistently high score across all evaluation parameters due to its hybrid design incorporating genetic optimization and multi-modal static feature fusion.

Moreover, the ensemble strategy adopted in GEN- Droid—leveraging the strengths of classifiers such as Random Forest, MLP, Naive Bayes, and Passive Aggressive—proves beneficial in generalizing across diverse malware behaviors. By optimizing both feature selection and hyperparameters through a genetic algorithm, the model adapts well to varying input structures without significant performance degradation.

Table 3: Performance Comparison of Android Malware Detection Techniques

Technique	Accuracy	Precision	Recall	F1 Score
Wajahat *et al.* (2024)	0.9790	0.982	0.976	0.979
Almarshad *et al.* (2023)	0.9940	0.989	0.988	0.987
Alani & Awad (2022)	0.9798	0.981	0.9759	0.9783
Mehtab *et al.* (2020)	0.9911	0.9933	0.9936	0.9934
Mahindru & Sangal (2021)	0.9828	0.9961	0.9773	0.9866
Liu *et al.* (2024)	0.9807	0.9931	0.9812	0.9871
GENDroid (Proposed)	**0.9862**	**0.9866**	**0.9955**	**0.9910**

Source: Author's own

In summary, GENDroid's comprehensive approach to malware detection significantly improves detection performance and offers a scalable, interpretable, and adaptive framework suitable for practical deployment in the Android ecosystem.

5.0 Conclusion

In this study, we proposed GENDroid, a robust and scalable Android malware detection framework that integrates multi-modal static features and evolutionary optimization techniques. Unlike traditional models that rely on isolated permission features or singular classification approaches, GENDroid harnesses a rich fusion of permissions, API calls, and intents—mapped through a Permission-to-Exploitation Mapping (PEM) strategy—to derive deeper semantic insights into application behaviors.

A genetic algorithm was employed not only for optimal feature subset selection but also for fine-tuning classifier hyperparameters, ensuring that the final ensemble model is both lightweight and high-performing. The ensemble comprises diverse classifiers including Random Forest, MLP, Naive Bayes, and Passive Aggressive, providing robustness and adaptability in detecting various malware types.

Extensive experiments on the Drebin dataset demonstrated that GENDroid outperforms several state-of-the-art techniques, achieving an accuracy of 98.62%, precision of 98.66%, recall of 99.55%, and an F1-score of 99.10%. These results validate the effectiveness of the proposed multi-modal optimization-driven approach in enhancing malware detection performance, especially in terms of recall and generalization. The modular and extensible nature of GENDroid makes it suitable for integration into real-world Android security ecosystems. Future directions include extending the framework to incorporate dynamic analysis features, building real-time detection capabilities, and introducing malware risk scoring systems for finer-grained security assessment.

6.0 Future Scope

The GENDroid framework has proven effective in static Android malware detection by leveraging multi-modal feature integration and genetic algorithm-driven optimization. However, to further elevate its practical utility and research value, several future enhancements can be considered: While GENDroid currently focuses on static features such as permissions, intents, and API calls, the addition of dynamic analysis—monitoring run- time behavior such as system calls, file access, network usage, and memory footprints—can substantially in- crease detection accuracy, particularly against obfuscated and polymorphic malware. A hybrid model would offer greater insight into behaviorally stealthy malware. In real- world applications, security professionals often require not just detection, but risk assessment. GENDroid can be extended to assign risk scores to each app based on the type and criticality of detected behaviors or permissions. This would enable prioritized threat response.

References

Ahmad, A., Shahid, Y., Mehmood, A., Abbas, M., & Imran, M. (2021). PermGuard: Fine-grained exploitation technique mapping for Android malware detection. In *Proceedings: IEEE International Conference on Communication (ICC 2021)*, pp. 1–6.

Alani, M. M. & Awad, A. I. (2022). PAIRED: An explainable lightweight Android malware detection system. *IEEE Access, 10*, 73214–73228. Retrieved from https://doi.org /10.1109/ACCESS.2022.3189645

Alkahtani, H. & Aldhyani, T. H. H. (2022). Artificial intelligence algorithms for malware detection in Android-operated mobile devices. *Sensors, 22*(6), 2268. Retrieved from https://doi.org/10.3390/s22062268

Almarshad, F. A., Zakariah, M., Gashgari, G. A., Aldakheel, E. A. & Alzahrani, A. I. A. (2023). Detection of Android malware using machine learning and Siamese shot learning technique for security. *IEEE Access, 11*, 127697–127714. Retrieved from https://doi.org/ 10.1109/ACCESS.2023.3331739

Gupta, R., Sharma, K. & Garg, R. K. (2024). Innovative approach to Android malware detection: Prioritizing critical features using rough set theory. *Electronics, 13*(3), 482. Retrieved from https://doi.org/10.3390/electronics13030482

Liu, Z., Wang, R., Japkowicz, N., Gomes, H. M., Peng, B. & Zhang, W. (2024). SeGDroid: An Android malware detection method based on sensitive function call graph learning. *Expert Systems with Applications, 235*, 121125. Retrieved from https://doi.org/ 10.1016/j.eswa.2023.121125

Mahindru, A. & Sangal, A. L. (2021). MLDroid—framework for Android malware detection using machine learning techniques. *Neural Computing and Application, 33*(10), 5183–5240. Retrieved from https://doi.org/10.1007/s00521-020-05309-4

Mehtab, A., Shahid, W. B., Yaqoob, T., Amjad, M. F., Abbas, H., Afzal, H. & Saqib, M. N. (2020). AdDroid: Rule-based machine learning frame- work for Android malware analysis. *Mobile Network and Applications, 25*(1), 180–192. Retrieved from https://doi.org/ 10.1007/ s11036-019-01248-0

Odat, E. & Yaseen, Q. M. (2023). A novel machine learning approach for Android malware detection based on the co-existence of features. *IEEE Access, 11*, 15471–15484. Retrieved from https://doi.org/10.1109/ACCESS.2023.3244656

Surendran, R., Thomas, T. & Emmanuel, S. (2020). GSDroid: Graph signal based compact feature representation for Android malware detection. *Expert Systems with Applications, 159*, 113581. Retrieved from https://doi.org/10.1016/j.eswa.2020.113581

Vu, L. N. & Jung, S. (2021). AdMat: A CNN-on-matrix approach to Android malware detection and classification. *IEEE Access, 9*, 39680–39694. Retrieved from https://doi.org/ 10.1109/ACCESS.2021.3063748

Wajahat, A., He, J., Zhu, N., Mahmood, T., Nazir, A., Ullah, F., Qureshi, S. & Dev, S. (2024). Securing Android IoT devices with GuardDroid transparent and lightweight malware detection. *Ain Shams Engineering Journal, 15*(5), 102642. Retrieved from https://doi.org/10.1016/j.asej.2024.102642