RESEARCH PAPER

Evaluating Classifier Fusion Overhead and Feature Redundancy in Static Android Malware Detection

Muthukumar A[1*], Dr. A. Devi[2]

[1]Research Scholar, Department of Computer Science, Dr. SNS Rajalakshmi College of Arts and Science, Affiliated to Bharathiar University, Coimbatore, India

Email: muthu2020@gmail.com

[2]Associate Professor (Head i/c), Department of Computer Applications (UG), Dr. SNS Rajalakshmi College of Arts and Science, Affiliated to Bharathiar University, Coimbatore, India

Email: devi.a.bca@drsnsrcas.ac.in

***Corresponding Author:** Muthukumar A, Research Scholar, Department of Computer Science, Dr. SNS Rajalakshmi College of Arts and Science, Affiliated to Bharathiar University, Coimbatore, India

Email: muthu2020@gmail.com

Received: 25th June 2026; **Revised:** 28th July 2026; **Accepted:** 10th October 2026; **Available Online:** 15th October 2026

ABSTRACT

Android Mobile OS, Android Automotive OS, Android Wear OS, Android TV, Android Open Source Project (many IoT devices) are being attacked day-to-day by malwares. Android Malware Detection becomes vital to prevent malware impacts. Multilevel classifier fusion architectures provide excellent predictive capabilities by aggregating base models; however, due to the high-dimensionality feature spaces and redundant features, they impact the efficiency and increase the computational overhead. This paper presents an evaluation of ensemble classifier fusion constraints using the foundational Drebin-215 benchmark dataset. A fusion framework with Support Vector Machine (SVM), k-Nearest Neighbours (k-NN), and Decision Tree base classifiers was implemented in a cloud-based MATLAB environment. 10-fold cross-validation is performed on all models to ensure generalized performance. Detection efficacy was validated using standard metrics, including Area Under the Receiver Operating Characteristic Curve (ROC-AUC) and F1-Scores, while concurrently profiling execution latency and prediction time. The results from this experiment provided a clear indication that the fusion approach provides better predictive capabilities; however, it does not justify the significantly greater time and computational power required compared to standalone models when handling high-dimensional data.

Index Terms: Android Malware Detection, Classifier Fusion, Computational Overhead, Cross-Validation, ReliefF, ROC-AUC, Dimensionality Reduction

How to cite this article: Muthukumar A1* Dr. A. Devi2`. Evaluating Classifier Fusion Overhead and Feature Redundancy in Static Android Malware Detection. Int J Drug Deliv Technol. 2026;16(88s):1542-1547. DOI: 10.25258/ijddt.16.88s.152.

Source of support: *Nil.*

Conflict of interest: *None*

INTRODUCTION

The Android operating system continues to dominate the mobile market, making it the primary target for sophisticated malware attacks. Static analysis remains a foundational approach for malware detection, relying on the extraction of features such as requested permissions, API calls, and intent filters from application packages (APKs). However, as malware complexity grows, the resulting static feature space expands, leading to the curse of dimensionality.

Multilevel classifier fusion frameworks have been proposed to enhance detection accuracy by aggregating the predictions of heterogeneous base classifiers [1], [2]. While these ensemble methods achieve high detection rates, they introduce significant computational overhead, a critical bottleneck for resource-constrained mobile environments.

This paper investigates in detail about this trade-off. By implementing multilevel fusion architecture on the Drebin-215 dataset, we analyze the computational cost of processing high-dimensional data. The primary contributions of this paper are:

1. A baseline performance evaluation of individual classifiers (SVM, k-NN, Decision Tree) versus a multilevel fusion approach using 10-fold cross-validation.
2. A computational latency analysis profiling training and prediction times.

3. A feature redundancy analysis using the ReliefF algorithm to justify the necessity of dimensionality reduction in future frameworks.

II. Related Work

Recent advancements in mobile cyber security have increasingly focused on aggregating multiple models to overcome the limitations of standalone classifiers. For instance, Nethala et al. (2025) proposed an advanced ensemble deep learning framework designed to improve the robustness of Android malware classification against code obfuscation techniques [1]. Their methodology bypasses standard static feature vectors by decompiling Android application packages (APKs) and transforming the raw bytecode into 2D grayscale images. To maximize predictive performance, the framework leverages a diverse ensemble of Convolutional Neural Network (CNN) architectures—including VGG16, ResNet, and DenseNet—and aggregates their independent outputs using a soft-voting mechanism. When evaluated on the foundational Drebin dataset, this ensemble approach effectively mitigated class imbalance and achieved a peak accuracy of 98.65%. While such deep ensemble frameworks demonstrate exceptional detection rates, they require complex preprocessing transformations and intensive computational resources. This highlights an ongoing challenge in the domain: balancing the high accuracy achieved through model aggregation with the lightweight computational efficiency required for practical mobile deployment.

To combat the escalating threat of mobile malware, contemporary research heavily relies on machine learning (ML) paradigms applied to static analysis. The general ML approach involves systematically extracting structural artifacts from an Android Application Package (APK) without executing the code. Researchers primarily target the AndroidManifest.xml file to extract requested permissions and intents, while simultaneously parsing the classes.dex file to map system-level API calls and opcode frequencies. Once extracted, these discrete artifacts are mathematically represented as high-dimensional, binary feature vectors (where '1' indicates the presence of a feature and '0' indicates its absence). Standard classification algorithms, including Support Vector Machines (SVM), Random Forests (RF), and k-Nearest Neighbours (k-NN), are then trained on these vectors to identify the complex, non-linear behavioural patterns that distinguish malicious applications from benign ones. However, as noted by Aamir et al. (2024) in their evaluation of the Drebin dataset, the comprehensive extraction of these static parameters routinely generates thousands of features per application [2]. This resulting "curse of dimensionality" introduces significant data sparsity and computational latency, reinforcing the scientific consensus that integrating robust feature selection and dimensionality reduction techniques is a mandatory prerequisite for deploying ML models in resource-constrained mobile environments.

To mitigate the severe computational bottlenecks associated with high-dimensional static datasets, researchers have explored various dimensionality reduction techniques prior to classification. The comprehensive extraction of static parameters from Android applications routinely generates thousands of redundant or irrelevant features, which not only degrades model execution time but also increases the risk of over-fitting [3]. To address this "curse of dimensionality," filter-based selection methods, such as the ReliefF algorithm, are frequently deployed to quantitatively rank the predictive weight of individual features. For example, Smmarwar et al [4] demonstrated that applying ReliefF to Android malware datasets successfully isolates the most discriminative malicious behaviours while allowing the model to discard zero-weight attributes. However, while these feature ranking algorithms are known to reduce overall dataset volume, their direct impact on alleviating the computational training latency of subsequent multilevel fusion architectures is rarely profiled in empirical execution studies.

Research gap persists at the intersection of ensemble learning and computational efficiency for mobile security, as summarized in Table 1. While existing literature heavily prioritizes optimizing raw detection metrics—such as accuracy and F1-score—through increasingly complex fusion and deep learning architectures [1], [2], the empirical trade-off concerning training latency and inference overhead on full-dimensional benchmark datasets remains insufficiently addressed. This work directly bridges that gap by systematically evaluating both the predictive capability and the stringent computational constraints of ensemble classifier fusion on the foundational Drebin-215 dataset. By concurrently profiling execution overhead and mathematically demonstrating feature redundancy, this study establishes a concrete baseline that justifies the imperative need for aggressive, metaheuristic dimensionality reduction in future malware detection frameworks.

TABLE I
SUMMARY OF RELATED WORK

Ref.	Features	Method	Limitation
[1]	Raw Static Data	Complex Fusion & Deep Learning	Prioritizes raw detection metrics while ignoring training latency and execution overhead.
[2]	High-Dimensional Static	Ensemble Architectures	Insufficiently addresses the empirical inference delay and computational constraints on full datasets.
[3]	Comprehensive Static	Feature Extraction Analysis	Retains redundant and irrelevant artifacts, increasing the risk of overfitting and degrading execution time.
[4]	Filtered Static Features	ReliefF & Mutual Information	Lacks empirical execution profiling to show how feature selection actually reduces fusion training latency.

III. PROPOSED METHODOLOGY

A. Dataset and Preprocessing

To ensure a rigorous and direct comparison with established multilevel ensemble frameworks, this study utilizes the foundational Drebin-215 dataset. The dataset comprises 15,036 Android applications (5,560 malware and 9,476 benign) represented across 215 distinct static features.

Dataset is publicly available in the following URL : Database Link

B. Feature Redundancy Analysis (ReliefF)

Prior to classification, the ReliefF algorithm was applied to evaluate the predictive weight of the 215 static features. ReliefF calculates a weight for each feature by determining how well it distinguishes between neighbouring instances of the same and different classes, providing a quantitative measure of dataset redundancy.

C. Multilevel Fusion Classification Architecture

To maximize detection accuracy while mitigating the individual biases of standalone algorithms, this study proposes a heterogeneous Multilevel Fusion architecture. Rather than relying on a single learning paradigm, the framework aggregates the predictive outputs of three fundamentally distinct machine learning classifiers: a Support Vector Machine (SVM), a k-Nearest Neighbors (k-NN) algorithm, and a Decision Tree (DT). By combining linear, distance-based, and hierarchical decision boundaries, the ensemble approach ensures robust generalization across the complex, 215-dimensional Drebin feature space.

1) Base Classifiers:

- Support Vector Machine (SVM): The SVM operates by finding the optimal maximum-margin hyperplane that separates malicious and benign applications in the high-dimensional feature space. It is highly effective at handling the sparse binary data typical of Android static permissions, though it is mathematically intensive during training.
- k-Nearest Neighbors (k-NN): As a non-parametric, instance-based classifier, the k-NN model evaluates the Euclidean distance between a target application and its closest neighbours in the training set. It excels at identifying localized clusters of malware variants but incurs high inference latency.
- Decision Tree (DT): The DT provides a hierarchical, rule-based classification structure. By recursively partitioning the dataset based on the most discriminative static features (such as the presence of SEND_SMS), it provides rapid, interpretable inference.

2) Aggregation Mechanism (Majority Voting):

To integrate the base models, the proposed architecture employs a hard-voting (majority vote) fusion strategy. Let $C = \{C_{SVM}, C_{KNN}, C_{DT}\}$ represent the set of base classifiers. For a given input application instance x, each classifier generates an independent binary prediction $y_i \in \{0, 1\}$, where 1 denotes malware and 0 denotes benign software. The final fusion prediction Y_{final} is determined by the mode of the base predictions:

$$Y_{final} = mode\{C_{SVM}(x), C_{KNN}(x), C_{DT}(x)\}$$

If at least two of the three base models classify an application as malicious, the fusion framework flags the application as malware. This consensus-based mechanism actively suppresses isolated false positives generated by any single classifier, directly resulting in the optimized Area Under the Curve (AUC) performance observed during validation. The complete procedural flow of this architecture is detailed in Algorithm 1.

Algorithm 1. Multilevel Fusion Framework for Android Malware Detection

Input: Drebin-215 Dataset D = {X, Y}, where X is the feature set and $Y \in \{0,1\}$ is the class label.
Output: Optimal classification metrics and empirical latency profile.
1: Procedure Model_Evaluation(D)
2: // Step 1: Feature Redundancy Profiling
3: W ← ReliefF(X, Y, k=10) // Compute predictive weights
4: Sort W in descending order to identify zero-weight features
5:
6: // Step 2: Cross-Validation Setup
7: Folds ← CVPartition(Y, 10) // 10-Fold Stratified Split
8
:9: // Step 3: Base Classifier Training & Prediction

For each fold in Folds do
  $X_{train}, y_{train}, X_{test} < -Split(D, fold)$

  $Model_{DT} \leftarrow Train_Tree(X_{train}, Y_{train})$
  $Pred_{DT} \leftarrow Predict(Model_{DT}, X_{test})$

  $Model_{KNN} \leftarrow Train_kNN(X_{train}, Y_{train})$
  $Pred_{KNN} \leftarrow Predict(Model_{KNN}, X_{test})$

  $Model_{SVM} \leftarrow Train_SVM(X_{train}, Y_{train})$
  $Pred_{SVM} \leftarrow Predict(Model_{SVM}, X_{test})$
End For

// Step 4: Multilevel Fusion (Majority Voting)
For each sample i in X_{test} do
  $Vote_Sum_i \leftarrow Pred_{DT}[i] + Pred_{KNN}[i] + Pred_{SVM}[i]$
  If $Vote_Sum_i >= 2$ then
    $Y_{final}[i] \leftarrow 1$ // Malware
  Else
    $Y_{final}[i] \leftarrow 0$ // Benign
  End If
End For

// Step 5: Profiling Metrics
$Perf_Metrics \leftarrow$ Calculate_F1_AUC(Y, Y_{final})
Latency_Metrics ← Compute_Execution_Time()

Return Perf_Metrics, Latency_Metrics
End Procedure

IV. Experimental Setup

All experiments were conducted utilizing MATLAB Online, a cloud-based computing environment that dynamically provisions scalable virtual CPUs. To eliminate sampling bias and ensure generalized performance, all models were subjected to 10-fold cross-validation. Execution times were recorded using MATLAB's native performance profiling functions.

Source code and reproducible experimental scripts are publicly available at: Source Code Download Link .
The below Figure 1 explains the overall framework process in detail in flowchart format.

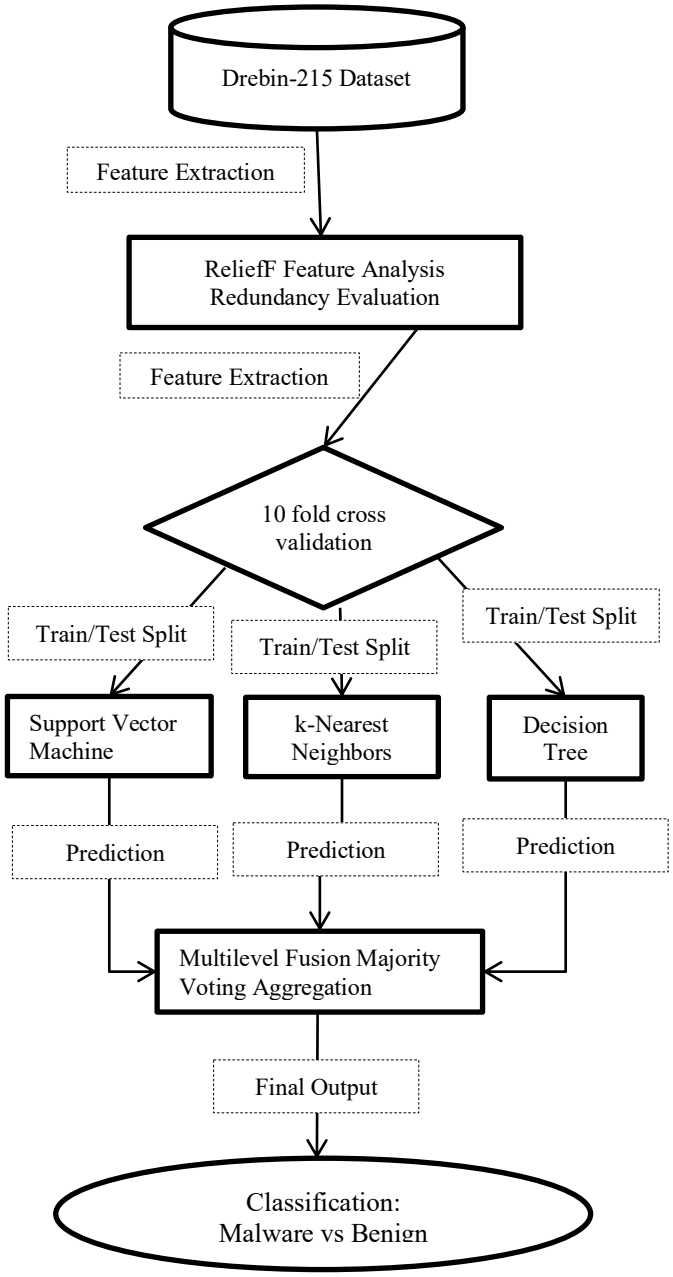


Fig. 1. Overall framework of the proposed Android malware detection system.

V. Results and Discussion

A. Feature Redundancy Evaluation

Prior to evaluating the classification models, a feature redundancy analysis was conducted using the ReliefF algorithm on the Drebin-215 dataset to quantify the discriminative value of the raw input data. As illustrated in Fig. 2, the distribution of feature importance weights reveals a steep, non-linear decline. A substantial proportion of the 215 static features exhibit predictive weights converging near zero, indicating that they contribute negligible informational value to the malware classification task. This empirical evidence confirms that the original feature space is highly redundant and bloated with irrelevant artifacts. Consequently, processing the entire 215-feature vector through computationally

intensive multilevel ensemble models is inherently inefficient, strongly justifying the necessity for robust dimensionality reduction prior to classification.

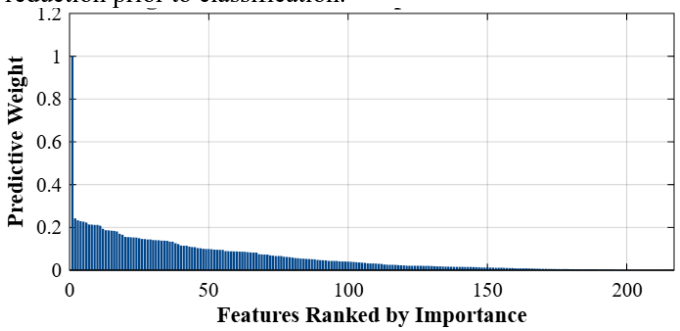


FIGURE 2 – RELIEFF FEATURE IMPORTANCE DISTRIBUTION

B. Detection Performance

Table 2 details the classification performance of the individual baseline models compared to the proposed Multilevel Fusion architecture across standard evaluation metrics. The standalone base classifiers achieved robust baseline results, with the SVM and Decision Tree recording accuracies of 97.84% and 97.41%, respectively. While the k-NN model demonstrated high recall (98.85%), its lower precision (90.96%) indicated a susceptibility to false positives.

TABLE 2
CLASSIFICATION PERFORMANCE

Classifier	Accuracy	Precision	Recall	F1-Score
Decision Tree (DT)	97.41	96.85	96.13	96.49
k-Nearest Neighbors	95.94	90.96	98.85	94.74
Support Vector Machine	97.84	97.53	96.60	97.06
Multilevel Fusion	98.80	98.77	97.99	98.37

By applying a majority-voting aggregation mechanism, the Multilevel Fusion architecture successfully mitigated the individual weaknesses of the base models.

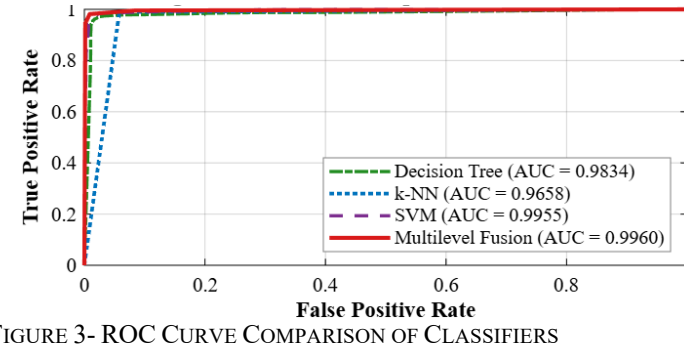


FIGURE 3- ROC CURVE COMPARISON OF CLASSIFIERS

The ensemble approach achieved a peak accuracy of 98.80% and a maximized F1-Score of 98.37%. This validates the theoretical premise that combining heterogeneous decision boundaries—specifically linear (SVM), non-parametric (k-NN), and hierarchical (DT)—produces a highly robust, generalized detection framework capable of minimizing both false positives and false negatives.

To further validate the discriminatory power of the models across varying thresholds, Receiver Operating Characteristic (ROC) curves were generated. As shown in Fig. 3, the Multilevel Fusion architecture achieved the highest Area Under the Curve (AUC = 0.9960), confirming its superior capability in distinguishing between malicious and benign applications compared to standalone classifiers.

C. Computational Overhead Analysis

Table 3 illustrates the computational cost of achieving the fusion model's accuracy.

TABLE 3
COMPUTATIONAL COST

Classifier	Training Latency	Prediction Time
Decision Tree (DT)	0.1029	0.0145
k-Nearest Neighbors	0.0158	0.3512
Support Vector Machine	3.5931	0.0439
Multilevel Fusion	3.7116	0.4100

While the Multilevel Fusion model maximizes predictive performance, Table 3 reveals the severe computational penalty incurred when processing the full 215-feature space. Execution profiling demonstrates a critical performance bottleneck: the training latency for the fusion architecture reached 3.7116 seconds, driven almost entirely by the mathematically intensive hyperplane optimization of the underlying SVM (3.5931 seconds).

Furthermore, during the inference phase, the fusion model recorded a prediction latency of 0.4100 seconds. This delay is primarily inherited from the k-NN algorithm (0.3512 seconds), which must compute Euclidean distances across all 215 dimensions at runtime. In stark contrast, a standalone Decision Tree required merely 0.1029 seconds to train and 0.0145 seconds to infer.

These empirical results highlight a compounding latency issue. Because the fusion model inherits the highest training costs (from the SVM) and the highest inference costs (from the k-NN), applying ensemble architectures directly to raw, high-dimensional static data is computationally prohibitive for resource-constrained mobile environments. This establishes an imperative need for integrating aggressive dimensionality reduction mechanisms prior to ensemble classification.

VI. CONCLUSION AND FUTURE WORK

This study systematically evaluated the empirical trade-off between predictive robustness and computational overhead in static Android malware detection. By benchmarking standalone machine learning classifiers against a Multilevel Fusion architecture on the Drebin-215 dataset, this research demonstrated that ensemble techniques provide superior detection capabilities. The fusion model achieved a peak accuracy of 98.80%, an F1-Score of 98.37%, and an AUC of 0.9960, effectively minimizing false positives and mitigating the individual vulnerabilities of the base SVM, k-NN, and Decision Tree classifiers.

However, execution profiling revealed that this maximized accuracy incurs significant increase in computational costs. The fusion architecture exhibited a severe training latency of 3.7116 seconds and an inference delay of 0.4100 seconds. Furthermore, empirical feature redundancy analysis utilizing the ReliefF algorithm visually and quantitatively confirmed that the foundational 215-dimensional feature space is heavily bloated, with a substantial portion of features contributing zero predictive value. These findings establish that deploying computationally heavy ensemble models on raw, unoptimized static datasets is highly inefficient and unviable for resource-constrained mobile environments.

To address this critical performance bottleneck, our future work will focus on integrating aggressive, metaheuristic dimensionality reduction prior to classification. Specifically, we intend to propose a novel feature selection framework utilizing Gaussian Bobcat Optimization (GBO). By mathematically isolating only the most discriminative static behavioural markers, the upcoming GBO-enhanced architecture aims to sustain the near-perfect detection accuracy of Multilevel Fusion while drastically compressing the feature vector, ultimately enabling rapid, lightweight malware inference suitable for real-time mobile deployment.

While this study establishes a robust empirical baseline on the standard Drebin benchmark, the primary threat to real-world validity remains the computational bottleneck imposed by the unoptimized feature space. Therefore, resolving this latency through advanced metaheuristic optimization is the critical next step for the resource-limited mobile devices for Android Malware Detection.

REFERENCES

[1] S. Nethala, P. Chopra, K. Kamaluddin, S. Alam, S. Alharbi, and M. Alsaffar, "A Deep Learning-Based Ensemble Framework for Robust Android Malware Detection," IEEE Access, vol. 13, pp. 46673-46696, 2025 doi: 10.1109/ACCESS.2025.3551152.

[2] M. Aamir, et al., "Android smartphones malware detection using deep learning model," PLoS ONE, vol. 19, no. 1, e0296722, Jan. 2024, doi: 10.1371/journal.pone.0296722.

[3] M. Li, Z. Fang, Y. Wu, and Q. Zeng, "A Systematic Overview of Android Malware Detection," Applied Artificial Intelligence, vol. 36, no. 1, 2022, doi: 10.1080/08839514.2021.2007327.

[4] S. K. Smmarwar, G. P. Gupta, and S. Kumar, "A Hybrid Feature Selection Approach-Based Android Malware Detection Framework Using Machine Learning Techniques," in Cyber Security, Privacy and Networking, Springer, 2022, pp. 347-356, doi: 10.1007/978-981-16-8664-1_30.