**Review of Computer Engineering Research**
*2026 Vol. 13, No. 4, pp. 1-28*
*ISSN(e): 2410-9142*
*ISSN(p): 2412-4281*
*DOI: 10.18488/76.v13i4.5152*


# A lightweight static-analysis framework with optimized feature selection for android malware detection and classification

**Vinay Kumar Dwivedi**[1+]
**Akhilesh A. Waoo**[2]

[1,2]*Department of Computer Science and Engineering, AKS University, Satna, Madhya Pradesh, India.*
[1]*Email: vinaykumardwivedi14@gmail.com*
[2]*Email: akhilesh_e1735@aksuniversity.com*

*(+ Corresponding author)*

**Article History**
Received: 26 January 2026
Revised: 24 April 2026
Accepted: 15 May 2026
Published: 7 September 2026

**Keywords**
Android malware detection
Lightweight deep learning
Malware family classification
PCA–ANOVA feature optimization
Static malware analysis.

## ABSTRACT

Android malware is growing rapidly, making detection more challenging and necessitating systems that are both accurate and efficient. This study aims to design a lightweight and reliable framework for Android malware detection and classification that reduces computational costs while maintaining strong performance. The proposed method employs static analysis and combines feature reduction, statistical feature selection, and a deep learning model. Initially, Principal Component Analysis (PCA) is used to reduce and simplify the original feature set. Subsequently, ANOVA-based ranking identifies the most important features with minimal computational effort. These selected features are then input into a CNN–BiGRU model, which learns both local patterns and sequential relationships within the data. The framework is evaluated using five-fold cross-validation on the Drebin and KronoDroid benchmark datasets for both binary and multiclass classification. Results demonstrate that the model achieves 98% accuracy on the Drebin dataset, 97% accuracy for binary classification on KronoDroid, and 96% accuracy for malware family classification. The PCA–ANOVA approach performs comparably to tree-based feature selection methods but requires less computation. Overall, the findings indicate that combining efficient feature selection with a lightweight neural network offers a practical, scalable, and effective solution for real-world Android malware detection.

**Contribution/Originality:** This study contributes to the existing literature by proposing a computationally efficient static Android malware detection framework. This study uses a PCA–ANOVA feature optimization methodology with formal computational complexity analysis. The paper's primary contribution is demonstrating scalable, high-accuracy detection with reduced computational cost.

## 1. INTRODUCTION

In recent years, Android malware has grown rapidly, and new types of malware are created annually. Android is the most widely used mobile operating system globally and is generally more affordable than others, such as iOS. Since Android is open-source and operates on many different devices, it becomes an easier target for cyber-attacks [1]. This increases the risk of privacy breaches, financial fraud, and unauthorized access to user data. Most commercial malware detection tools are already available, with some being free, but most rely on rigid signatures and heuristic rules and cannot detect zero-day threats.

Recent studies show that malware often evolves within families, with new variants inheriting behaviors from earlier ones. Therefore, analyzing malware at the family level and using supervised methods is crucial for effective

detection. Frameworks that incorporate knowledge of family relationships and structured behavior models have enhanced generalization against new Android threats [2, 3].

There are three well-known approaches for Android malware detection: static, dynamic, and hybrid. Static analysis detects Android malware without executing the application by extracting features from the manifest and code, such as permissions, intents, and API calls. Dynamic analysis monitors the runtime behavior of the application by running it in a safe environment, such as a sandbox or a virtual box [4]. Static analysis is computationally efficient, easy to scale [5, 6], whereas dynamic analysis is more robust against obfuscation; still, it is heavier and more vulnerable to sandbox-evasion tactics. Hybrid approaches are based on a mix of two approaches but often introduce complexity.

Traditional machine learning techniques, including Random Forest, SVM, KNN, and clustering methods, have helped improve classification performance [7]. In the recent past, deep learning models such as CNNs and RNN variants (LSTM, GRU, BiLSTM) [8] have gained attention for their ability to model long-term dependencies and detect zero-day malware patterns [9, 10]. Deep Learning techniques performed very well in terms of malware detection accuracy, but many deep learning models operate on high-dimensional static or dynamic features, which causes the model's high training time, memory consumption, and susceptibility to overfitting.

Feature selection and dimensionality reduction methods, such as Genetic Algorithms, Chi-square tests, transformer embeddings, and autoencoder-based reduction, have been introduced to reduce redundancy and computational load [1, 7, 11]. Android datasets contain many features that have a strong relation among them and can change over time. Due to these factors, feature optimization is necessary to ensure that machine learning and deep learning models perform accurately.

Due to the above-mentioned challenges, it is important to use feature optimization techniques that must include expert knowledge and solid statistics. These approaches can help improve the performance of malware detection systems.

Although dynamic analysis is highly capable of capturing behavioral dynamics, it is difficult to apply at scale due to its high computational complexity and evasiveness. Moreover, there are only a few studies in the existing literature that discuss computational complexity. Most deep learning-based models are highly accurate but are often computationally expensive and require considerable resources, such as GPUs or CPUs, for computation [12]. Few recent research studies have proposed optimized architectures to overcome this complexity, but data generalizability remains an issue [13].

Static Android features such as permissions, API calls, intents, and opcode patterns are often sparse, noisy, and highly correlated. CNNs extract local feature interactions, while BiGRU models dependencies across feature dimensions in the optimized feature space. In this study, we use a compact and leakage-safe pipeline that:

- Standardizes and decorrelates static features using z-score normalization. It then applies PCA to reduce the dimensionality and noise of this dataset.
- It ranks discriminative components using ANOVA k-best selection and classifies malware by using a hybrid CNN–BiGRU architecture that can learn both spatial and temporal patterns.

For a balanced evaluation, the PCA–ANOVA pipeline was compared with a PCA–Random Forest feature selection baseline under the same experimental conditions. Performance was assessed using accuracy, weighted F1-score, and ROC–AUC through 5-fold cross-validation to ensure consistent and reliable results.

### *1.1. Problem Statement*

Despite significant developments in Android malware detection, existing approaches still fall short.

- High computational complexity due to large and redundant feature spaces.
- Instability of tree-based feature selection methods, particularly in high-dimensional and imbalanced datasets.

- Limited generalization across benchmark and contemporary datasets as a result of noise, feature correlation, and concept drift.
- A lack of lightweight deep learning frameworks suitable for real-time or resource-constrained environments.

These challenges show that there is a need for a detection method that is efficient, stable, and works well across different datasets. This study makes the following contributions:

- It introduces a two-step feature selection process (PCA followed by ANOVA-k) that reduces unnecessary features, lowers computational cost, and provides more stable feature selection compared to tree-based methods.
- It develops a lightweight CNN–BiGRU model that can learn both local feature patterns and sequential relationships, while using fewer parameters than LSTM-based models.
- It evaluates the method on both the Drebin and KronoDroid datasets, showing consistent performance on older as well as more recent malware samples.
- A detailed computational complexity analysis comparing PCA–ANOVA and PCA–RF in terms of cost, stability, and scalability, an aspect rarely covered in prior malware research.
- Statistical validation using paired t-tests and Wilcoxon tests, ensuring that performance differences are significant and reliable.

This study presents an efficient and reliable framework for Android malware detection. It focuses on solving the problem of handling very large static feature sets. The proposed method first uses PCA to reduce the number of features, then applies ANOVA-based k-ranking to select the most important ones. The selected features are then passed to a CNN–BiGRU deep learning model for classification.

The framework achieves high detection accuracy while keeping computational costs low. It is suitable for large datasets and can adapt to new types of malware, making it practical for real-world use.

To systematically evaluate the proposed framework and validate its computational and predictive effectiveness, the study is guided by the following research questions:

- RQ1: Can PCA-ANOVA k-best features reduce the desired number of features while maintaining the detection accuracy?
- RQ2: How do PCA-ANOVA and PCA-RF (Random Forest) compare in terms of Accuracy and computational cost?
- RQ3: Is the proposed CNN–BiGRU model stable across different Android malware detection scenarios under cross-validation?

The remainder of this paper is organized as follows: Section 2 explains related work on Android malware detection, including different approaches, types, and the research gap. Section 3 describes the proposed pipeline, dataset, preprocessing, and feature optimization technique. Section 4 presents results, discussion, and comparison with state-of-the-art methods. Section 5 concludes the paper with limitations and future directions.

## 2. RELATED WORK

This literature study mainly focused on static, dynamic, and hybrid analyses, including previously reported feature-optimization techniques for Android malware detection. It aligned with the objectives of this work.

### *2.1. Static Analysis and Machine Learning Approaches*

Early approaches to Android malware detection primarily focused on static features such as permissions, API calls, intents, and opcode patterns. Using these features, various traditional machine learning classifiers, including Random Forests, Support Vector Machines, k-Nearest Neighbors, Naïve Bayes, and Decision Trees, achieved baseline detection performance on benchmark datasets such as Drebin and AndroZoo.

For example, Bhat and Dutta [5] used permission and API-based static features with information gain and frequency filtering, achieving 96.28% accuracy. Similarly, a few studies applying Genetic Algorithms or Chi-square tests show that feature reduction improves classification performance [7, 11]. Permission-only detection systems such as Ariffin and Casinto [14] and Şahin, et al. [15] have also obtained reasonable accuracy by eliminating redundant features.

Static ML models are sensitive to high dimensionality, multicollinearity, and noisy feature spaces, leading to unstable performance. They depend on manual feature engineering and cannot capture complex behavioral patterns. Additionally, they are computationally expensive, even though most ML-based works do not address the computational cost of large handcrafted feature sets.

### *2.2. Dynamic and Hybrid Behavioral Detection Methods*

Dynamic analysis extracts dynamic features while running, such as network activity, system calls, and memory use. Many studies show that dynamic analysis gives better generalization against obfuscation and new malware. For instance, Islam, et al. [1] used dynamic logs from the CICMalDroid dataset with ensemble learning, achieving 95% accuracy after reducing features with PCA. Hybrid models like DBN-GRU [16] integrate both types of features (Static and Dynamic) to improve robustness.

More advanced systems, such as DL-AMDet [17], use CNN-BiLSTM and deep autoencoders to model behavioral sequences, achieving 99.93% accuracy.

Dynamic analysis usually requires a lot of computing power, is hard to use directly on devices, and can be avoided by sandbox evasion. Many hybrid methods also rely on large amounts of runtime data and need long training times, so they are not practical for large-scale or real-time use. Because of these issues, there is a clear need for efficient malware detection methods that use only static analysis.

### *2.3. Deep Learning Architectures for Static or Dynamic Features*

Zhang, et al. [18] utilized the CNN-BiGRU method with semantic API embeddings, acquiring an F1-score of 98.27%. The use of the CNN-LSTM hybrid model for real-time detection approaches perfection, as depicted in the study by Akhtar and Feng [13]. The importance of intelligent selection for feature selection using reinforcement learning has been illustrated in the study by Wu, et al. [19] amongst others.

Deep models typically operate on high-dimensional, sparse, and correlated feature spaces, requiring heavy computation and longer training times. Most works do not incorporate dimension reduction or cost analysis, and many depend on unstable feature selection methods, limiting their scalability and deployment feasibility.

### *2.4. Feature Selection and Dimensionality Reduction for Efficiency*

Feature selection helps improve classification accuracy and lower computational costs. Many dimensionality reduction and feature-ranking methods, including Principal Component Analysis, Chi-square testing, permutation importance, Genetic Algorithms, and wrapper-based approaches, have been used to reduce redundant Android malware features. Previous studies show that removing irrelevant features often makes models more efficient and accurate.

For example, Pathak, et al. [20] reduce the number of permission-based features by almost 90% using importance scores from Gradient Boosting, without losing detection accuracy. Other studies using PCA or Chi-square selection [1, 7] found faster execution and better efficiency, but did not compare different ranking methods in detail. Tree-based importance measures like Extra Trees and Random Forests are still widely used because they perform well, but can be affected by class imbalance and feature correlation. Waheed and Qadir [21] used Extra Tree-based ranking on the KronoDroid dataset and reached high accuracy, though this increased computational cost.

Even with recent improvements, there is still a lack of detailed comparison between PCA combined with statistical ranking methods like ANOVA and PCA combined with tree-based feature selection. Very few studies check whether the selected features remain stable across different cross-validation folds. Additionally, many works do not clearly analyze computational complexity. As a result, important questions about efficiency and scalability remain unanswered.

From the existing studies, several critical gaps have been identified:

- High computational costs and redundant features are still issues that have not been solved.
- Tree-based ranking methods do not perform consistently across cross-validation folds when working with high-dimensional and imbalanced datasets.
- Not many studies look at lightweight deep learning frameworks, such as those based on GRU, that focus on efficiency instead of just accuracy.
- There are no studies that combine PCA for decorrelation, ANOVA for stable ranking, and a CNN–BiGRU classifier into one efficient pipeline.
- Only a few models have been tested on both Drebin and KronoDroid, which means there is limited proof that they work well for both older and newer malware types.

Most existing papers did not include formal analysis of computational complexity or tests for statistical significance. Table 1 summarizes a comparative study of the proposed work with previous work in this field.

**Table 1.** Comparative Assessment of Existing and Proposed Android Malware Detection Frameworks.

| Study | Feature Strategy | Model Type | Complexity Analysis | Statistical Testing | Datasets | Lightweight Design |
|---|---|---|---|---|---|---|
| Bhat and Dutta [5] | Info Gain | ML Ensemble | No | No | Drebin | No |
| Zhang, et al. [18] | API Embeddings | CNN–BiGRU | No | No | Drebin | Partial |
| Waheed and Qadir [21] | ExtraTree Ranking | Random Forest | Limited | No | KronoDroid | No |
| AlSobeh, et al. [22] | Time-Aware Learning | Classical ML | No | No | KronoDroid | No |
| Proposed | PCA–ANOVA (Two-Stage) | Lightweight CNN-BiGRU | Yes (Formal) | Yes (t-test & Wilcoxon) | Drebin + KronoDroid | Yes |

Previous machine learning, deep learning, and hybrid approaches have several limitations, such as high computational costs, weak dimensionality reduction strategies, and unstable performance. These issues highlight the need for a lightweight, efficient malware detection system capable of performing well in various situations.

The proposed framework solves these issues by removing extra features, improving class separation, reducing computational cost, and maintaining stable performance across different datasets. Table 2 summarizes the related studies reviewed in this paper.

**Table 2.** Summary of previous work.

| SR | Previous work | Accuracy | Dataset | Features used | Technique used |
|---|---|---|---|---|---|
| 1 | Rahali, et al. [23] | 93.36% | Large balanced Set (Image-encoded static features) | Static: permissions, intents, hardware encoded as images | CNN used with image-encoded feature maps |
| 2 | Islam, et al. [1] | 95% | CCCS-CIC-And Mal-2020 | Dynamic logs: memory, logcat, API calls, network | Weighted-voting ensemble |
| 3 | Wu, et al. [19] | 95.6%, (24/1083 features) | Drebin /Andro Zoo-style corpora | Static features; RL-selected compact subset | DDQN wrapper selection |
| 4 | Waheed and Qadir [21] | 98.03 (binary), 87.56 (15-class) | Enhanced KronoDroid | Hybrid: 200 static + 289 dynamic; Top-50 selected | Classical Machine |

| SR | Previous work | Accuracy | Dataset | Features used | Technique used |
|---|---|---|---|---|---|
| | | | | | learning and RF Tree for feature ranking. Malware category classified |
| 5 | Zhang, et al. [18] | 98.28% (F1 98.27) | 3rd party public (API sequences) | Behavioral API call sequences | CNN–BiGRU captures local/global semantics |
| 6 | Bhat and Dutta [5] | 96.28% | Static corpus (multi-source) | Static features -> frequency filtering + feature discrimination + information gain | Machine learning algorithm used |
| 7 | Akhtar and Feng [13] | ≈99% (reported) | malware behavioral dataset (generic) | Behavioral/NLP-style signals | CNN-LSTM outperforms DT (98%) and SVM (95%) |
| 8 | Karat, et al. [24] | 96% | Behavioral logs (own collection) | Dynamic logs; API monitoring, extension checker | Hybrid CNN-LSTM for behavioral detection (Dynamic) |
| 9 | Morán, et al. [7] | Accuracy: 99.52% F1-score: 96.99% | DREBIN Dataset 123,453 benign & 5,560 malware apps | Static features (manifest + disassembled code); Applied Chi-Square feature reduction (9% of total features) | Random Forest, SVM, Naive Bayes, Decision Tree, Logistic Regression |
| 10 | Kauser and Anu [12] | Accuracy: 98.7% AUC: 0.99 | DREBIN Dataset 129,013 apps (5,560 malware, 123,453 benign) | Static (permissions, API, intents) + Dynamic (System calls, IPC, network); used PCA for feature selection | Hybrid DBN–GRU Combining static & dynamic analysis |

## 3. METHODOLOGY

This study adopts a structured experimental methodology designed to jointly evaluate predictive performance, computational efficiency, and statistical stability within a unified framework.

Much of the literature on Android malware detection has focused on the accuracy of malware detection. The proposed work not only constrains the accuracy but also integrates dimensionality reduction, statistical feature ranking, and lightweight deep learning for classification and detection. The computational complexity analysis within this framework has also been included. The framework compares PCA-ANOVA with PCA-Random Forest feature selection strategies under identical cross-validation conditions. However, existing studies often did not consider complexity assessment or ranking stability analysis. This work emphasizes computational cost, fold-wise stability, and statistical significance, providing a more efficient evaluation of Android malware detection models.

During classification, one-dimensional convolutional layers identify local patterns in the optimized features, and MaxPooling reduces data size and speeds processing. The feature maps are then sent to a Bidirectional Gated Recurrent Unit, which learns patterns in both directions to better model sequences in the data.

A final fully connected layer provides class probability estimates for either binary malware detection or multiclass family classification, depending on the experiment. The framework is tested with k-fold cross-validation to ensure it performs well on new data. Results indicate that the PCA → ANOVA → CNN–BiGRU pipeline offers strong detection performance at low computational cost and consistently outperforms tree-based feature selection methods.

This section explains the steps of the proposed PCA → ANOVA → CNN–BiGRU framework for detection and classification of Android malware. This framework addresses common issues in static malware analysis, high dimensionality, redundant features, and slow processing.

### *3.1. Overview of the Proposed Framework*

To design the pipeline of the proposed framework, first apply the preprocessing step to the raw Android application features, suitable for the next step. In preprocessing, noisy, redundant, missing, and inconsistent attributes are removed, and feature normalization is applied to ensure consistent scaling across all variables.

After preprocessing, PCA is applied to reduce the dimensionality of the high-dimensional feature space. Correlated features are projected by PCA, forming a smaller set of independent components while maintaining high variance. This results in lower computational costs and helps the model create more useful and insightful features.

Following the PCA, the framework uses an ANOVA F-score feature ranking to examine how well the principal components separate the classes. It picks the top components to create an optimized set of features, which are then used as input for the CNN–BiGRU classifier. The convolutional layers find spatial patterns, and the bidirectional gated recurrent unit captures context in the feature sequence. The framework is tested using 5-fold cross-validation and standard metrics to assess its robustness, fairness, and generalization across datasets.

### *3.2. Data Description*

This subsection describes the datasets used in this study and explains their importance. Two well-known datasets, Drebin and KronoDroid, were chosen because they are reliable and widely used in Android malware detection. Both datasets are publicly available and offer diverse samples, enabling a thorough evaluation of the proposed model in binary and multiclass classification. More details are provided in subsections 3.2.1 and 3.2.2.

#### *3.2.1. KronoDroid Dataset (Multi-Class and Binary Classification)*

The KronoDroid dataset is a large-scale, hybrid Android malware corpus comprising 41,382 malicious and 36,755 benign applications collected between 2008 and 2020, enabling robust evaluation under evolving malware behavior and concept drift. It includes 200 static features (permissions, intents, etc.) and 289 dynamic features (system calls, API calls, network, and memory usage), with dynamic traces captured on real devices to counter emulator-evasive malware. Although originally designed for binary classification, the dataset was later extended to include 15 malware families such as Adware, Trojan, Ransomware, Spyware, and Trojan-Banker. Some class imbalance persists (e.g., 11,185 Adware vs. 62 Trojan-Dropper), and approximately 30% of samples remain unlabeled. Nevertheless, its hybrid nature, temporal coverage, and broad accessibility make KronoDroid one of the most comprehensive datasets for both binary and multiclass Android malware research.

#### *3.2.2. Drebin Dataset (Binary Classification)*

The Drebin dataset includes 15,036 Android apps, each described by 215 static, binary features such as permissions, API calls, and code attributes. It contains 5,560 malware samples and 9,476 benign samples, making it suitable for binary classification. Some features, like Write_External_Storage, are common and appear in 67% of apps, while others, like HARDWARE_TEST, are rare and found in less than 1%. These features capture both normal and malicious behaviors. Drebin is a feature-rich (static) dataset that is efficient and does not require a running application. This has made it a popular benchmark for research on Android malware detection and feature selection.

### *3.3. Data Preprocessing*

The preprocessing pipeline is a crucial step in this methodology. The raw data contains missing values and inconsistent entries that must be cleaned for proper classification results. In this article, two datasets have been used for classification purposes: the Kronodroid dataset and the Drebin dataset. KronoDroid contains structured static and dynamic features. Non-feature identifiers (e.g., hashes, package names, timestamps) were removed. The remaining numeric feature matrix was normalized and used for PCA-based feature optimization. Z-score normalization is applied to standardize all features to a zero mean and unit variance. This step is especially important for deep learning models

because they can be sensitive to differences in feature scales. Normalization ensures that no single feature with larger values will overshadow the rest; hence, the model will treat all features fairly. Moreover, it speeds up the learning process and makes training more effective and stable. The Drebin dataset contains inconsistencies in data; therefore, data cleaning techniques are applied to obtain better results.

The data is now ready to pass over the next step: feature extraction and selection. For feature reduction, Principal Component Analysis (PCA) is applied. PCA converts a high-dimensional feature space into a low-dimensional space. Given the high-dimensional feature space, the PCA is employed as $Z = XW$, where $X \in R^{nxd}$ is a feature matrix, W is the transformation matrix of the top k eigenvectors, and Z is the reduced feature representation.

After reducing dimensionality with PCA, ANOVA-based k-ranking is used to find the most important features for malware classification. PCA transforms correlated features into independent components, creating a more stable basis for analysis. ANOVA then checks each component by comparing differences between classes and within classes, so only features with a strong ability to distinguish malware are kept.

In comparison, decision tree–based ranking depends strongly on the model and can become unstable when applied to high-dimensional or imbalanced datasets such as Drebin and KronoDroid. These methods may give greater weight to features from the majority classes, leading to different rankings across experiments. The results show that ANOVA produces more consistent feature sets with less variation between tests. At the same time, it achieves similar or better accuracy and F1 scores, while requiring much lower computational cost. Overall, the combined PCA–ANOVA method provides an efficient and stable approach for selecting features for Android malware detection and classification, while maintaining reliable performance.

### *3.4. Feature Space Behavior Analysis*

Android malware datasets often have many features, are sparse, and exhibit significant feature overlap. These issues can negatively impact computational cost and model generalization. This section examines the feature space to identify where variance concentrates, detect redundancy, and assess whether feature reduction is necessary before classification.

The Android malware dataset contains many features that are sparse and partially overlapped. Such characteristics increase computational requirements and may affect the model's ability to generalize to unseen samples. Therefore, this section analyzes the feature space to determine how variance is distributed and the need for dimensionality reduction before classification, thereby identifying redundant attributes.

#### *3.4.1. High-Dimensional Feature Characteristics*

Android malware datasets are usually highly sparse, with many features, because they include numerous static attributes related to permissions, API usage, and system operations. Most of these attributes are binary, indicating whether a specific action occurs, leading to uneven patterns across different apps. While a few features appear frequently and help distinguish between apps, many others add little value and are often redundant because they are strongly correlated with other features.

Having so many features makes it hard for deep learning classifiers. Redundant and correlated features increase the time and resources needed for training and prediction. Features that are noisy or rarely change can also make the model less accurate and cause overfitting. For these reasons, using the raw feature space is not efficient, especially when working with large datasets or limited resources. It is better to review the features before training to create smaller sets that keep the important information about malware and remove what is not needed.

### *3.4.2. PCA-Based Variance Analysis*

Principal Component Analysis was used to study how variance is spread in the high-dimensional feature space. As shown on the right side of Figure 1, only a few principal components capture most of the total variance. This means that the main information is found in these leading components.

As a result, the feature space can be compressed with little loss of information, which also reduces redundancy and makes computation easier. The sharp rise in cumulative explained variance indicates that a small number of principal components can capture most of the meaningful information in the dataset. This shows the importance of retaining a limited number of features for further analysis and of maintaining the behavioral patterns associated with Android malware.

After reducing the dimensionality, an ANOVA-based ranking method was used to identify components with the greatest ability to separate groups. The left panel of Figure 1 shows the Pareto distribution of the top 50 ANOVA-ranked features, illustrating their individual contributions. The Pareto plot indicates that permission uses, API calls, and system-level activities are highly important. These features are critical in distinguishing malicious applications from benign ones.

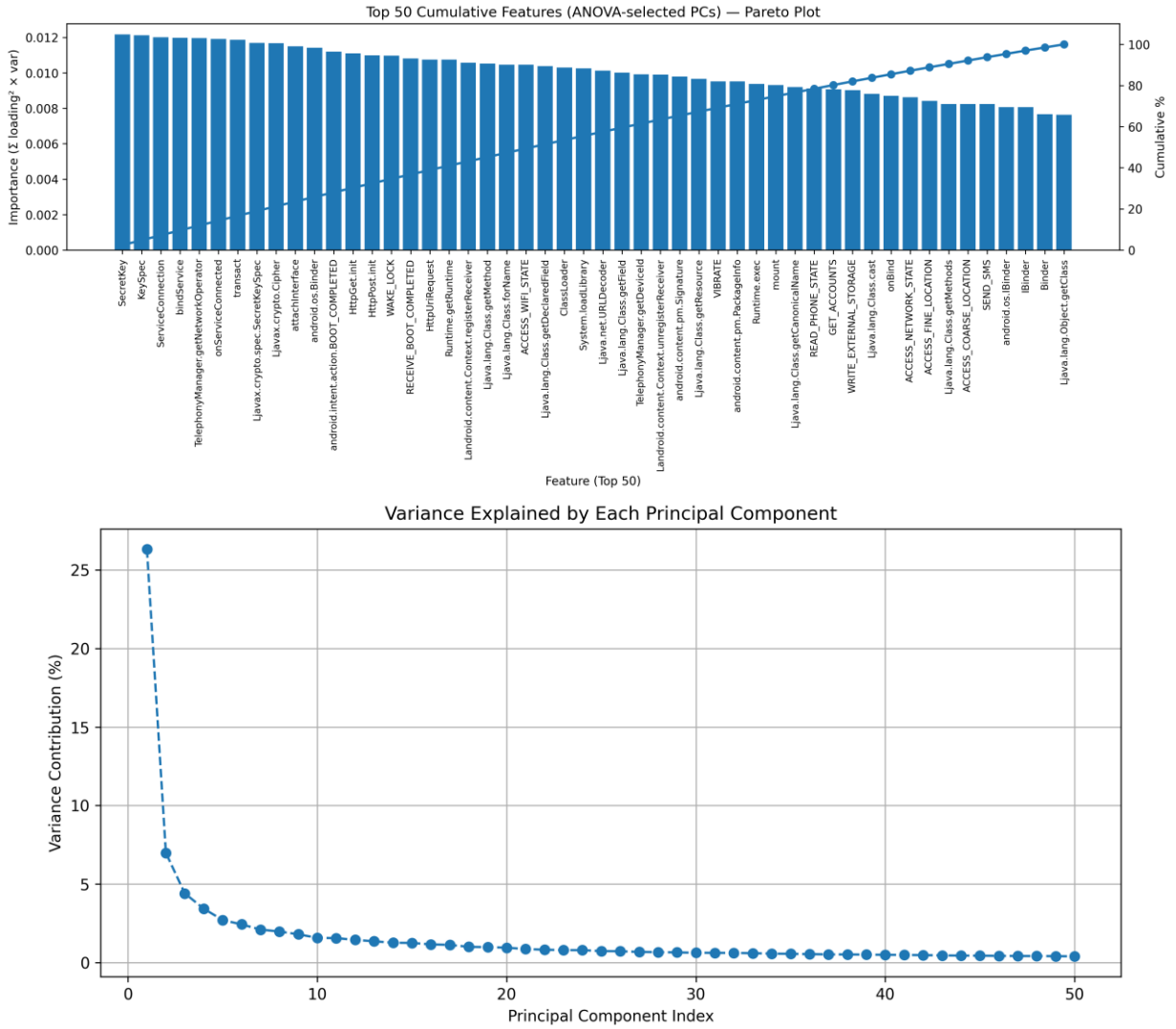


**Figure 1**. PCA Variance Distribution and ANOVA-Based Feature Pareto.

The PCA variance plot shows that most of the variance is accounted for by the first few principal components. As seen in Figure 1 (right panel), PC1 alone explains 27% of the total variance, and the first 15 components together

account for about 70%. After PC50, each component adds less than 1% variance, so the information gained drops off. This indicates PCA effectively reduces the Android feature space dimensions without significant information loss. Table 3 summarizes the variance contributions.

**Table 3.** Variance Contribution details.

| PC Index | Variance Contribution in % | Cumulative Variance % | Interpretation |
|---|---|---|---|
| PC1 | 27% | 27% | Captures major global variance due to highly common malware–benign differences |
| PC2 | 7% | 34% | Captures secondary patterns such as permission clusters |
| PC3 | 5% | 39% | Adds signals related to API usage diversity |
| PC4 | 4% | 43% | Reflects subtle configuration-related attributes |
| PC5 | 3% | 46% | Captures minor but meaningful behavioral groupings |
| PC6-PC15 | ≈2% → 1% each | ~70% cumulative | Combined small but stable contribution; covers rare but useful patterns. |
| PC16-PC50 | <1% each | >90% cumulative | Remaining PCs carry negligible variance; suitable for dimensional reduction |

#### *3.4.3. ANOVA-Based Discriminative Feature Analysis*

A PCA-based variance analysis, along with an ANOVA F-score, has been used to measure how well individual components and their related behaviors can separate malicious apps from benign ones. This method helps identify the features that best distinguish between the two classes and provides insight into typical malware behaviors. The rankings from this process guide the feature selection strategy in our proposed framework.

The PCA-transformed features are further evaluated using ANOVA-based ranking to determine how effectively each component differentiates between classes. Features like SecretKey, KeySpec, ServiceConnection, bindService, and TelephonyManager.getNetworkOperator, as well as cryptographic or reflective calls such as javax. crypto.Cipher, Class.forName, and System.loadLibrary are strong signs of malicious activity. These actions are often linked to accessing sensitive data, dynamic loading, reflection, encryption, and running native code, common techniques used by Android malware to hide their functionality and trigger payloads. Together, the top 50 features account for more than 90% of the total discriminative variance, indicating that ANOVA effectively identifies behaviorally relevant components.

The analysis also shows that Android malware feature spaces have a lot of redundancy and uneven variance, with only a few components truly helping to separate classes. While PCA reduces dimensionality by keeping the most important variance, this alone does not guarantee the best separation between malware and benign samples. Because of this, we added a second step for discriminative ranking. We now use a two-stage feature optimization strategy that combines PCA with ANOVA-based k-best selection, as described in the next section.

### *3.5. Feature Optimization Pipeline*

This study compares two feature selection pipelines: PCA–ANOVA and PCA–Random Forest. Both use PCA for dimensionality reduction, then apply another method to rank and select the most informative features. The analysis examines each pipeline's computational cost based on the number of samples (n), original feature dimensionality (d), retained principal components (p), selected feature subset size (k), and number of classes (C).

#### *3.5.1. PCA–ANOVA Hybrid Feature Selection*

After applying PCA, the resulting components are tested with the ANOVA F-test to see how well they separate class labels. Components with higher F-scores show a clearer distinction between malicious and benign samples. The top-k components from this ranking are then kept to build the optimized feature set for classification.

In this pipeline, PCA is first applied to the normalized input matrix $X \in R^{n*d}$ to project it into an uncorrelated subspace of $p$ principal components. The ANOVA F-test is subsequently applied to these components to identify the most statistically significant features for classification.

### *3.5.2. Computational Complexity Analysis of PCA–ANOVA*

When randomized (Singular Value Decomposition) SVD is employed to retain only p components, the cost of computation is:

$T_{PCA}$ = O(ndp), the $T_{PCA}$ Computational complexity is represented in an upper bound.

Followed by feature ranking on Principal components. Each component's discriminative power across C classes is evaluated by computing the between-class and within-class variance, giving a cost of $T_{ANOVA}$.

$T_{ANOVA}$ = O (np + Cp) => O(np) because Cp<<np, Total cost using ANOVA: $T_{PCA\text{-}ANOVA}$ = O (ndp + np)

### *3.5.3. Baseline Comparator: PCA → Random Forest Feature Importance*

In this Pipeline, the Random Forest (RF) is employed on the PCA-transformed features to rank and select the most important features. A Random Forest consists of T decision trees. If one tree costs O(p.nlogn) time, then for T trees it will cost O(T. pn logn).

$$\text{Total cost} = T_{PCA} + T_{RF} \quad => \quad T_{PCA\text{-}RF} = O(ndp) + O(T.\ pn\log n)$$

### *3.5.4. Comparative Complexity Analysis*

Table 4 summarizes the comparative analysis of the hybrid feature selection approaches.

**Table 4.** Comparative analysis of the hybrid feature selection approaches.

| Pipeline | Step1 (PCA) | Step2 (Selector) | Total Cost | Remarks |
|---|---|---|---|---|
| PCA-ANOVA | O(ndp) | O(np) | O(ndp)+O(np) | Linear & Efficient |
| PCA-RF | O(ndp) | O(Tnp logn) | O(ndp)+O (Tnp logn) | Computationally Expensive |

The computational cost of the PCA–RF method is much higher than the PCA–ANOVA approach. The results show that using Random Forest for feature ranking does not provide a clear improvement in detection accuracy, even though it requires significantly more time and memory. In contrast, the PCA–ANOVA method achieves similar or slightly better accuracy while using much less computational power. This makes it a more efficient and practical choice for Android malware detection.

### *3.5.5. Summary of Computational Efficiency*

The hybrid PCA–ANOVA pipeline balances accuracy and computational efficiency for several reasons.

- PCA reduces the number of features early on, which makes later processing easier.
- ANOVA k feature ranking requires minimum computational cost.
- This method avoids the expensive and repetitive tree-building process found in RF.
- Results show that this method cuts computational load by about 40 to 60 percent compared to PCA-RF.
- Accuracy stays high, between 96 and 98 percent, so there is no loss in discriminative power (Shown in section 4).

This indicates that PCA–ANOVA is both theoretically efficient and practically suitable for large malware datasets.

### *3.6. CNN–BiGRU Classification Model*

The optimized set of features produced by the PCA and ANOVA pipeline is then further processed using a Hybrid Convolutional Neural Network and Bidirectional Gated Recurrent Unit model that facilitates the identification of

both spatial and contextual feature dependencies within the set of feature attributes of Android malware. The Convolutional Neural Network layer acts as a spatial feature extractor that relies on a number of one-dimensional convolutional layers with varying kernel sizes between 3 and 5. ReLU activation ensures that non-linearity has been incorporated to facilitate fast convergence. MaxPooling layers are used after convolutional layers to reduce noise.

The feature representations produced by the CNN layers are subsequently forwarded to a Bidirectional GRU (BiGRU) module to capture sequential dependencies in both forward and backward directions. GRUs have a simpler structure than BiLSTM networks. GRU has only an update and a reset gate. This reduces the number of parameters for training and allows the model to learn long-range dependencies. The Bidirectional GRU helps understand long-range context from both directions, allowing later states to influence earlier classification decisions.

Dropout is applied after the convolutional and recurrent layers, helping the model generalize better and reduce overfitting. L2 regularization limits the weights in the dense layers. Batch normalization stabilizes the training process by normalizing activation values within each mini-batch.

Finally, a sigmoid function is used for binary malware detection, and a SoftMax function for classifying malware families; both produce probability-based outputs.

The proposed framework uses CNN features together with a Bidirectional GRU to learn local patterns and overall context from data. By combining these components, the model captures important patterns with relatively few parameters, making it practical for large-scale Android malware detection.

Figure 2 shows the complete Methodological Architecture of the proposed work. The architecture begins with the preprocessing step of the two benchmark datasets, KronoDroid and Drebin, followed by PCA-ANOVA feature ranking for the top k most important features. It then proceeds with five-fold cross-validation and concludes with a CNN–BiGRU hybrid classifier to classify malicious and benign apps. The proposed framework is compared with the PCA-RF baseline using accuracy and ROC-AUC scores across all folds. Results demonstrate a high malware detection rate while maintaining very low computational costs compared to state-of-the-art methods.

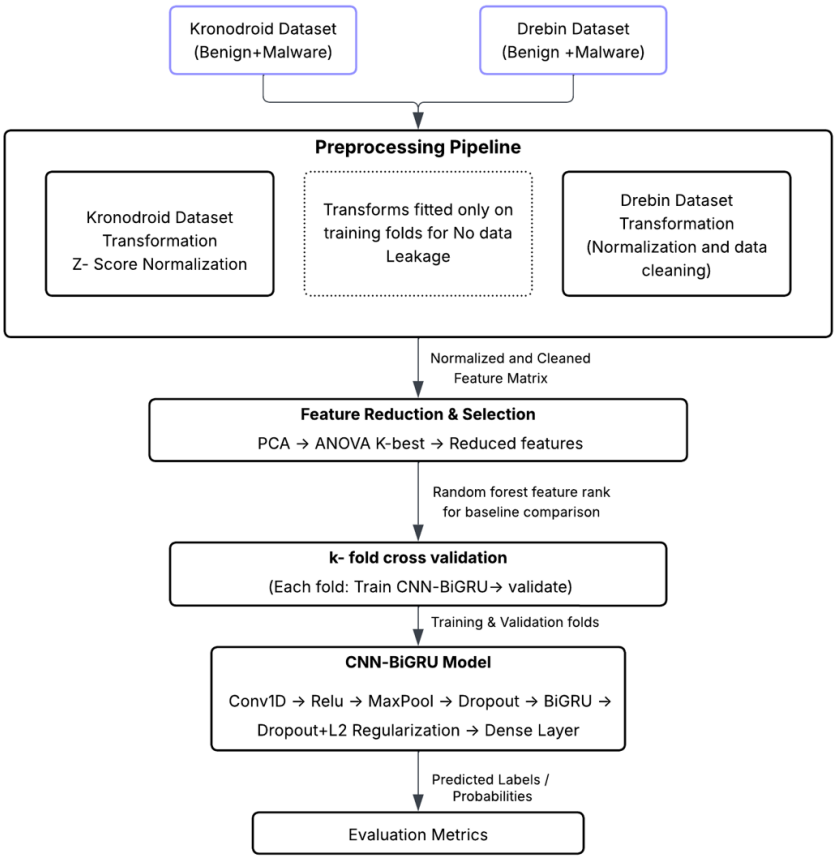


**Figure 2**. Complete methodological pipeline.

## 4. RESULTS AND DISCUSSION

This section describes the experimental evaluation of the proposed PCA–ANOVA–CNN–BiGRU framework using two well-known Android malware datasets. The evaluation is divided into two main parts.

In the first part, the Drebin dataset is used to examine how well the model performs in binary malware detection (malware vs. benign apps). PCA is applied to reduce feature dimensions, and the ANOVA F-score is used to rank the most important components. The results are then compared with a baseline approach that combines PCA with a Random Forest (RF) k-rank method.

In the second part, the KronoDroid dataset is tested using the same feature selection and optimization process. Here, the model is evaluated for both binary malware detection and multiclass classification, where different malware families are identified.

The performance of the framework is measured using standard metrics such as accuracy, precision, recall, F1-score, and ROC–AUC. Additionally, statistical tests are conducted to confirm whether the observed improvements are significant.

### *4.1. Experimental Setup and Evaluation Metrics*

The experiments were conducted using the Drebin dataset, which contains 15,036 Android applications described by 215 static features such as permissions, API calls, and intent information. Of these applications, 5,560 are labeled as malware, and the remaining are benign.

To ensure the model provides reliable results and does not simply memorize the data, we used a 5-fold cross-validation method. This means the dataset was split into five equal groups. Each group was used once as test data, while the other four groups were used to train the model. This process was repeated five times so that every sample was tested once.

Using this method provides a fair evaluation and helps confirm that the results are stable. The hybrid CNN–BiGRU model was trained and evaluated on these folds to classify applications as either malicious or benign.

To understand how feature optimization affects both classification performance and computational cost, we tested two different feature selection approaches. In the first approach, we used Principal Component Analysis (PCA) to reduce the number of features, then applied ANOVA to select the top k most important components from the reduced set.

In the second approach, PCA was again used for dimensionality reduction, but the final feature ranking was done using Random Forest. In this case, the most important features were selected based on the importance scores calculated by the Random Forest model.

These two pipelines were compared to see which method provides better accuracy while keeping the computational cost lower.

### *4.2. Binary Malware Detection Performance*

This section describes how effectively the proposed framework detects malware using two commonly used Android datasets. The evaluation examines the model's accuracy, consistency, and performance on new, unseen data. To verify this, a five-fold cross-validation method was employed.

The results from the Drebin and KronoDroid datasets show how effective the feature selection method and the hybrid CNN–BiGRU model are at distinguishing between malicious and benign applications while maintaining computational efficiency.

### *4.2.1. Drebin Dataset Results*

The binary malware detection performance was first evaluated on the Drebin dataset using the proposed framework. A 5-fold cross-validation strategy was employed to ensure statistical robustness and mitigate overfitting. Feature selection and model training were performed independently within each fold to avoid information leakage.

The resulting model has attained an average accuracy of 98%, and the F1 score is approximately 0.98. The high F1 score indicates that the classifier maintains equal weights on precision and recall values, and it does not favor either the malware or benign side. The ROC calculation also provides the best possible AUC value of 0.996, as summarized in Table 5.

These results validate that aggressive dimensionality reduction through PCA, followed by ANOVA-based discriminative feature ranking, does not degrade classification performance. Instead, the optimized feature representation enables the CNN–BiGRU model to learn compact yet informative behavioral patterns from the static features of Android.

**Table 5.** Performance Summary of CNN–BiGRU Model on Drebin Dataset (5-Fold Cross-Validation).

| Metric | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Mean ± Std |
|---|---|---|---|---|---|---|
| Accuracy | 97.2% | 97.7% | 97.4% | 98.1% | 98.2% | 97.72 % ± 0.30 % |
| Precision | 97.0% | 97.5% | 97.2% | 98.0% | 98.1% | 97.56 % ± 0.39 % |
| Recall | 97.1% | 97.6% | 97.3% | 98.0% | 98.1% | 97.62 % ± 0.35 % |
| F1-Score | 97.1% | 97.5% | 97.3% | 98.0% | 98.1% | 97.60 % ± 0.32 % |
| ROC-AUC | 0.995 | 0.996 | 0.995 | 0.996 | 0.996 | 0.996 ± 0.0005 |

The CNN–BiGRU model exhibited reliable convergence on all the folds. Figure 3 illustrates the evolution of training and validation accuracy, loss, and F1-score, ensuring that the model actually generalizes without overfitting.

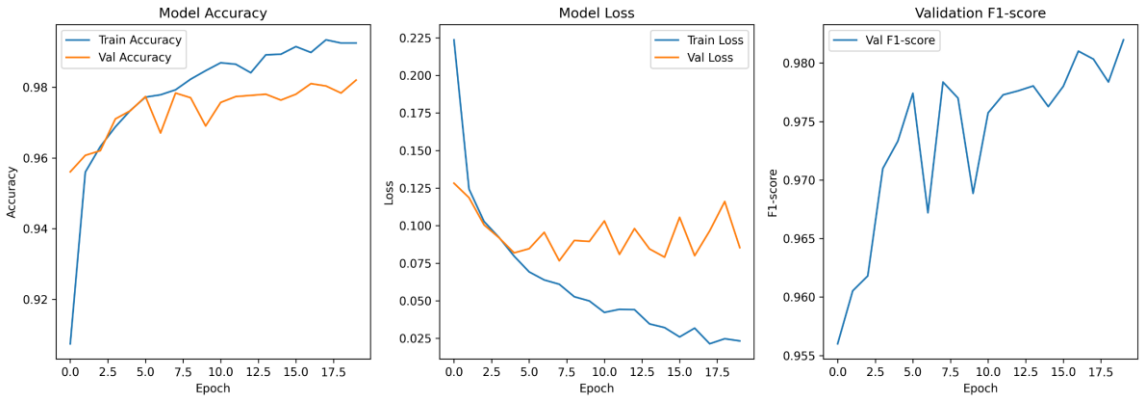


**Figure 3**. Training and validation accuracy (Drebin dataset).

The CNN–BiGRU model was tested on the Drebin dataset using 5-fold cross-validation to ensure stable, trustworthy results. Over the five folds, the model achieved an average validation accuracy of 97.72% (± 0.30%) and an average F1-score of 97.60% (± 0.32%).

During training, both training and validation accuracy improved gradually, indicating that the model was learning useful patterns rather than merely memorizing data. The small differences between the folds also suggest that the model performs consistently and works well on new, unseen samples.

The convolutional layers helped identify important local patterns in the features, while the BiGRU layer learned how these features relate to each other in sequence. By combining these two parts, the model was able to detect malware accurately. The results show that combining PCA–ANOVA feature selection with the CNN–BiGRU model provides reliable and accurate Android malware detection.

The model's performance was evaluated using the confusion matrix and ROC-AUC curve. As shown in Figure 4, the CNN-BiGRU model effectively identified benign and malicious applications, with detection accuracy above 98%. This indicates consistent performance. The ROC–AUC for this model was 0.996, demonstrating its ability to distinguish malicious apps from normal ones. The confusion matrix shows a clear distribution.

Across all samples, the model correctly classified 9,459 benign apps and 5,445 malware apps, resulting in only 127 total errors. This indicates that the model makes very few mistakes and is particularly effective at reducing false negatives, which is very important in malware detection because failing to detect a malicious app can lead to serious security risks.

The ROC curve also confirms this strong performance. Its steep shape toward the top-left corner shows a high true positive rate (0.97) and a very low false positive rate (0.01). This means the model is both sensitive to malware and accurate in recognizing safe apps.

These results show that the CNN–BiGRU model, combined with PCA and ANOVA feature selection, achieves high detection accuracy with minimal errors. It also maintains a lower computational cost than many recent static deep learning approaches, making it both effective and efficient.

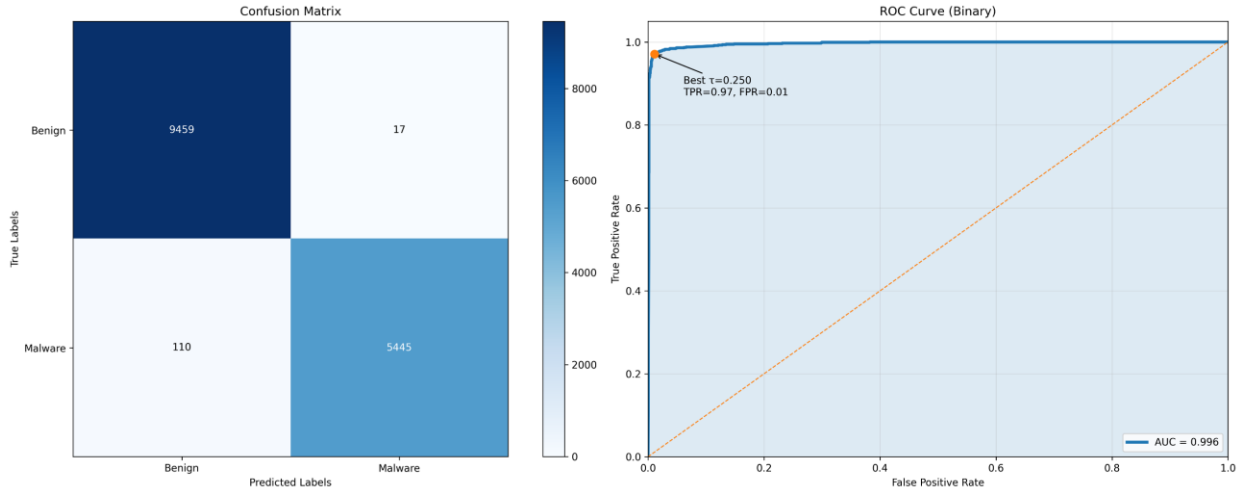


**Figure 4**. Confusion Matrix and ROC Curve (Drebin dataset).

### *4.2.2. KronoDroid Dataset Results (Binary Classification)*

To check whether the model can handle larger, more varied data, the same method was tested on the KronoDroid dataset. This dataset includes a bigger and more diverse collection of Android apps, both benign and malicious, with different behavior patterns. As in the Drebin experiment, 5-fold cross-validation was used to ensure the results were reliable.

On the KronoDroid dataset, the proposed framework achieved an average accuracy of 97% and an average F1-score of about 96.4% across all folds. The average AUC score was 0.9935 (shown in Figure 6), indicating the model's ability to distinguish clearly between malware and benign apps.

For each fold, the best decision threshold was chosen from the ROC curve using Youden's J statistic. This helped maintain a proper balance between catching malware and reducing false alarms. With this threshold, the model achieved a high true positive rate of about 0.97 and a low false positive rate of around 0.04. In simple terms, it was able to detect most malicious apps while making only a small number of mistakes with safe apps.

The stable results across all folds show that the CNN–BiGRU model, trained on features selected using PCA and ANOVA, is robust and does not depend heavily on a specific dataset. It performs consistently well even on larger and more diverse data. Table 6 presents the detailed results for each fold on the KronoDroid dataset.

**Table 6.** Fold-wise validation summary for KronoDroid dataset.

| Fold | Validation AUC | (τ)* | Test Accuracy | F1-Score |
|---|---|---|---|---|
| 1 | 0.9932 | 0.432 | 0.9631 | 0.9612 |
| 2 | 0.9934 | 0.422 | 0.9664 | 0.9645 |
| 3 | 0.9939 | 0.311 | 0.9679 | 0.9664 |
| 4 | 0.9935 | 0.535 | 0.9670 | 0.9652 |
| 5 | 0.9935 | 0.538 | 0.9657 | 0.9636 |

Figure 5 presents the training progress of the CNN–BiGRU model. Both validation accuracy and F1-score improved steadily during the first 20 epochs and then stabilized at around 97% validation accuracy and 96.7% F1-score. The training and validation loss decreased smoothly, indicating a stable learning process without overfitting. The accuracy curves for training and validation remained close, demonstrating good generalization.

After the 10th epoch, a small gap between training and validation loss indicated that regularization was effectively controlling overfitting. The model continued to improve steadily, with validation accuracy stabilizing around 96.7%. The smooth decrease in loss and consistent F1-score show stable learning behavior.

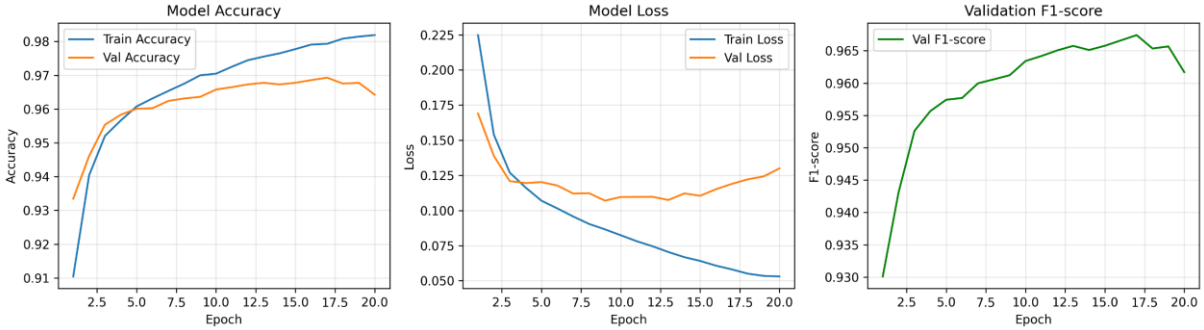


**Figure 5**. Training and validation accuracy (KronoDroid dataset).

Figure 6 shows the confusion matrix for the proposed CNN–BiGRU model, indicating it accurately identified most benign and malware instances, with few false positives (1,614) and false negatives (1,040). It has high discriminative power, correctly classifying 39,768 benign and 35,715 malware samples with very few errors.

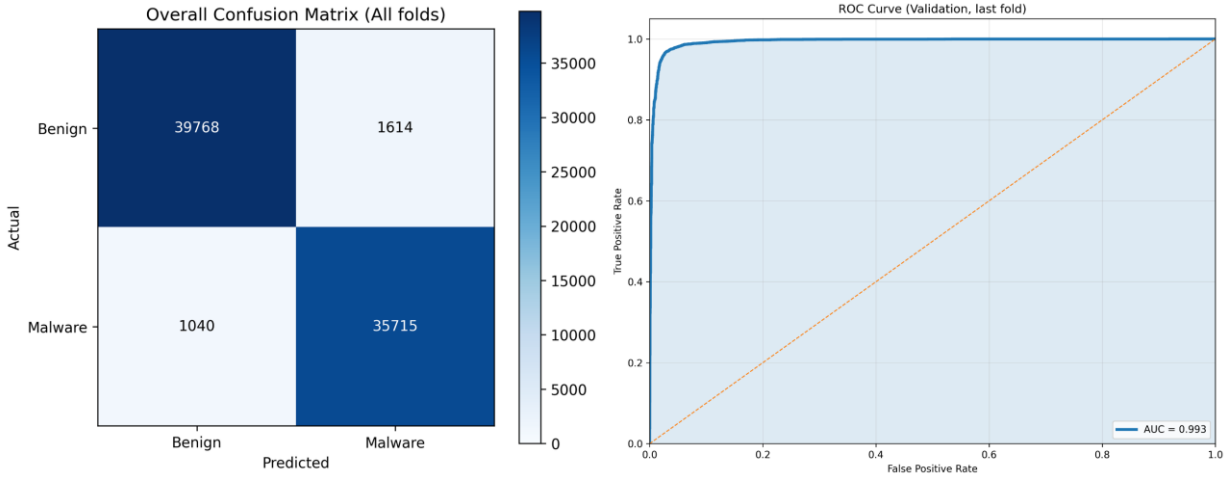


**Figure 6**. Confusion Matrix and ROC Curve (KronoDroid dataset).

Therefore, the KronoDroid detection model achieved excellent detection precision (96.6%), balanced recall (0.97), and a low false-positive rate (4%), surpassing conventional practices that struggled with high-dimensional and imbalanced datasets.

### *4.3. Feature Selection Impact and Computational Efficiency*

The comparative analysis presented in this section was conducted on the KronoDroid dataset. KronoDroid was selected for this evaluation due to its larger scale and greater feature diversity, which made it suitable for assessing classification stability and the computational efficiency of feature selection strategies. All performance metrics and statistical analyses reported in this section were derived from identical training–validation splits to ensure a fair and unbiased comparison between the PCA–ANOVA and PCA–Random Forest pipelines. Many existing studies on Android malware detection primarily focus on improving detection performance while paying little attention to the computational cost of feature selection methods. Accordingly, this section examined the trade-off between detection effectiveness and computational efficiency by comparing the proposed PCA–ANOVA feature optimization pipeline with a PCA–Random Forest baseline. Recent work by Polatidis, et al. [25] showed that the number of features in Android malware datasets can be greatly reduced without affecting detection accuracy. In their FSSDroid framework, thousands of features were narrowed down to just a few dozen through careful preprocessing and feature selection. This demonstrates that removing unnecessary features is important for lowering computational costs and improving efficiency. These findings support the idea behind the proposed PCA–ANOVA pipeline, which also focuses on using a smaller, more compact set of features while maintaining strong detection performance.

#### *4.3.1. PCA–ANOVA vs PCA–RF: Accuracy–Cost Trade-Off*

We compared how different feature selection strategies affect performance and computational efficiency by analyzing PCA→ANOVA-k-best and PCA→Random Forest (RF)-k-best pipelines, both using the CNN–BiGRU classifier. All experiments used the same 5-fold cross-validation settings for fairness.

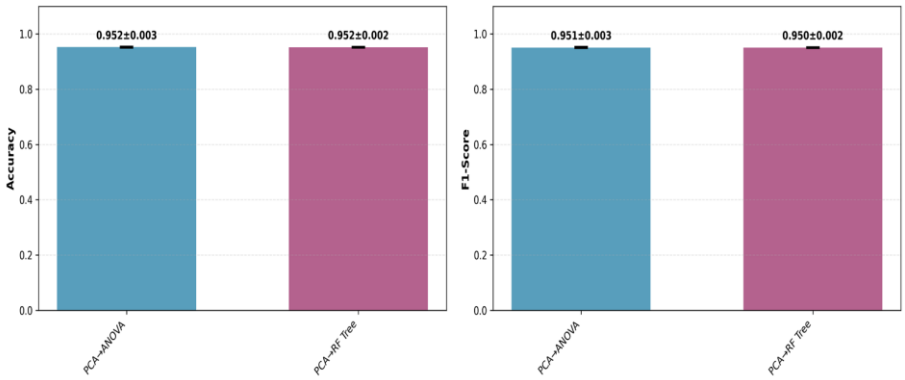


**Figure 7**. Accuracy Metrics: PCA-ANOVA vs PCA-RF.

The PCA→ANOVA pipeline achieved a mean accuracy of 0.9518 ± 0.0028 and a mean F1-score of 0.9505 ± 0.0027, as shown in Figure 7, while the PCA→RF pipeline achieved 0.9515 ± 0.0019 accuracy and 0.9503 ± 0.0019 F1-score. Although the performance difference is marginal, PCA→ANOVA demonstrated slightly better stability across folds (as shown in Figure 8), winning in 3 out of 5 folds for both accuracy and F1-score.

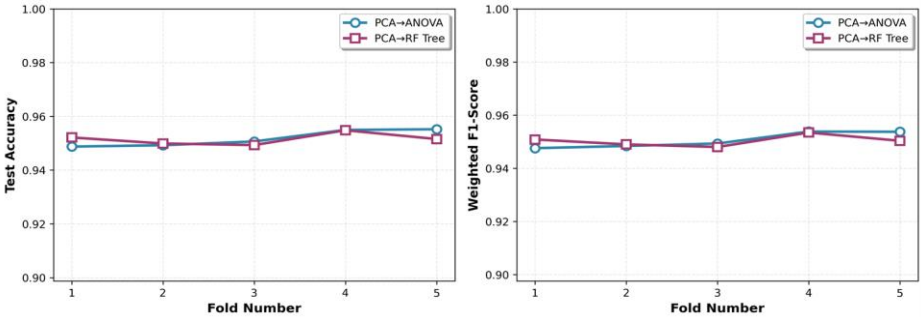


**Figure 8**. Fold-wise performance: PCA-ANOVA vs PCA-RF.

From a computational standpoint, PCA→ANOVA had linear-time complexity after dimension reduction. In contrast, PCA→Random Forest required much more computing power due to the construction of multiple trees and the estimation of feature importance. The Random Forest-based ranking also did not improve accuracy enough to justify the extra time and memory it used. These results show that PCA→ANOVA maintains a good balance between accuracy, stability, and computational efficiency. This makes it better suited for large-scale or resource-limited malware detection. Table 7 shows a comparison of this method with the baseline tree-based feature ranking approach.

**Table 7.** Comparative discussion of PCA-ANOVA & PCA-RF.

| Metrics | PCA→ANOVA | PCA→RF Tree | Winner |
|---|---|---|---|
| Mean Accuracy (±SD) | 0.9518 ± 0.0028 | 0.9515 ± 0.0019 | ANOVA |
| Mean F1-Score (±SD) | 0.9505 ± 0.0027 | 0.9503 ± 0.0019 | ANOVA |
| Fold-wise accuracy wins | 3 | 2 | ANOVA |
| Fold-wise f1 score wins | 3 | 2 | ANOVA |

#### *4.3.2. Statistical Stability and Significance Analysis*

To examine the performance differences between the PCA→ANOVA and PCA→Random Forest feature selection pipelines, which were significant, we used paired t-tests and Wilcoxon signed-rank tests on the fold-wise evaluation metrics shown in Table 8. The p-values for both measures were above the significance threshold ($\alpha = 0.05$), so the differences between the two pipelines were not statistically significant. This means that PCA→ANOVA performed as well as PCA→Random Forest but required much less computing power. The lower standard deviation across cross-validation folds also showed that the PCA→ANOVA pipeline was more stable. In summary, these results show that the proposed feature optimization strategy is an efficient and scalable alternative to tree-based methods. It does not reduce detection effectiveness, so it is suitable for practical and large-scale Android malware detection systems.

**Table 8.** Statistical significance analysis between PCA–ANOVA and PCA–RF Pipelines.

| Metric | PCA → ANOVA (Mean ± SD) | PCA → RF (Mean ± SD) | t-Statistic | p-Value | Significance ($\alpha = 0.05$) |
|---|---|---|---|---|---|
| Accuracy | 0.9518 ± 0.0028 | 0.9515 ± 0.0019 | 0.1880 | 0.8600 | Not Significant |
| F1-Score | 0.9505 ± 0.0027 | 0.9503 ± 0.0019 | 0.2200 | 0.8367 | Not Significant |

### *4.4. Feature Importance Interpretation*

In addition to achieving high detection accuracy, understanding which features influence classification decisions was crucial. This improved the interpretability, trust, and practical application of Android malware detection systems. This section discusses the most important behavioral features identified by the PCA–ANOVA pipeline and explains how they assist in distinguishing malware. Mahindru, et al. [26] introduced PermDroid, a large-scale static malware detection framework that uses multi-stage statistical feature selection and ensemble neural models. Their method achieves high detection accuracy on large real-world datasets, but it depends on several ANN training strategies and ensemble methods. In comparison, the proposed framework reaches similar detection results by using a compressed PCA–ANOVA feature space and a lightweight CNN–BiGRU architecture, making it a more efficient option.

#### *4.4.1. High-Impact Behavioral Features*

Permissions such as Read_Phone_State, Write_External_Storage, and ACCESS_WIFI_State really stand out because they allow apps to access user data and device resources. These are the kinds of permissions that spyware, data-stealing malware, and trojans typically target, posing security risks to users.

API calls such as TelephonyManager.getNetworkOperator() and TelephonyManager.getDeviceId() are problematic. They seem to be used for device fingerprinting and collecting user data without permission. Apps that

do not care about privacy often do this. READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE, and ACCESS_WIFI_STATE can be used to gather extensive information about the user and their device.

System-level operations like System.loadLibrary, Class.forName, and cryptographic routines using javax.crypto.Ciphers are also important features. These APIs are often linked to dynamic code loading, reflection-based execution, encryption, and obfuscation. Advanced malware uses these techniques to avoid static analysis. As shown in Table 9, the strong presence of these features indicates that the feature optimization pipeline captured real behavioral indicators rather than just surface-level patterns.

**Table 9.** Top 10 most discriminative features identified via PCA–ANOVA ranking.

| Rank | Feature Name | Category | Importance Score |
|---|---|---|---|
| 1 | android.permission.READ_PHONE_STATE | Permission | High |
| 2 | android.permission.WRITE_EXTERNAL_STORAGE | Permission | High |
| 3 | android.permission.ACCESS_WIFI_STATE | Permission | High |
| 4 | System.loadLibrary | System API | High |
| 5 | getDeviceId() | API Call | High |
| 6 | SEND_SMS | Permission | Medium-High |
| 7 | RECEIVE_BOOT_COMPLETED | Permission | Medium-High |
| 8 | READ_CONTACTS | Permission | Medium |
| 9 | READ_SMS | Permission | Medium |
| 10 | BIND_ACCESSIBILITY_SERVICE | Permission | Medium |

### *4.4.2. Cumulative Feature Contribution Analysis*

To better understand the feature contribution, a Pareto analysis of the feature rank was shown in Figure 9. The Pareto curve indicates that only a small group of features accounts for most of the model's ability to distinguish between classes. In fact, the top 50 features together explain over 90% of the total discriminative variance, while the rest contribute very little.

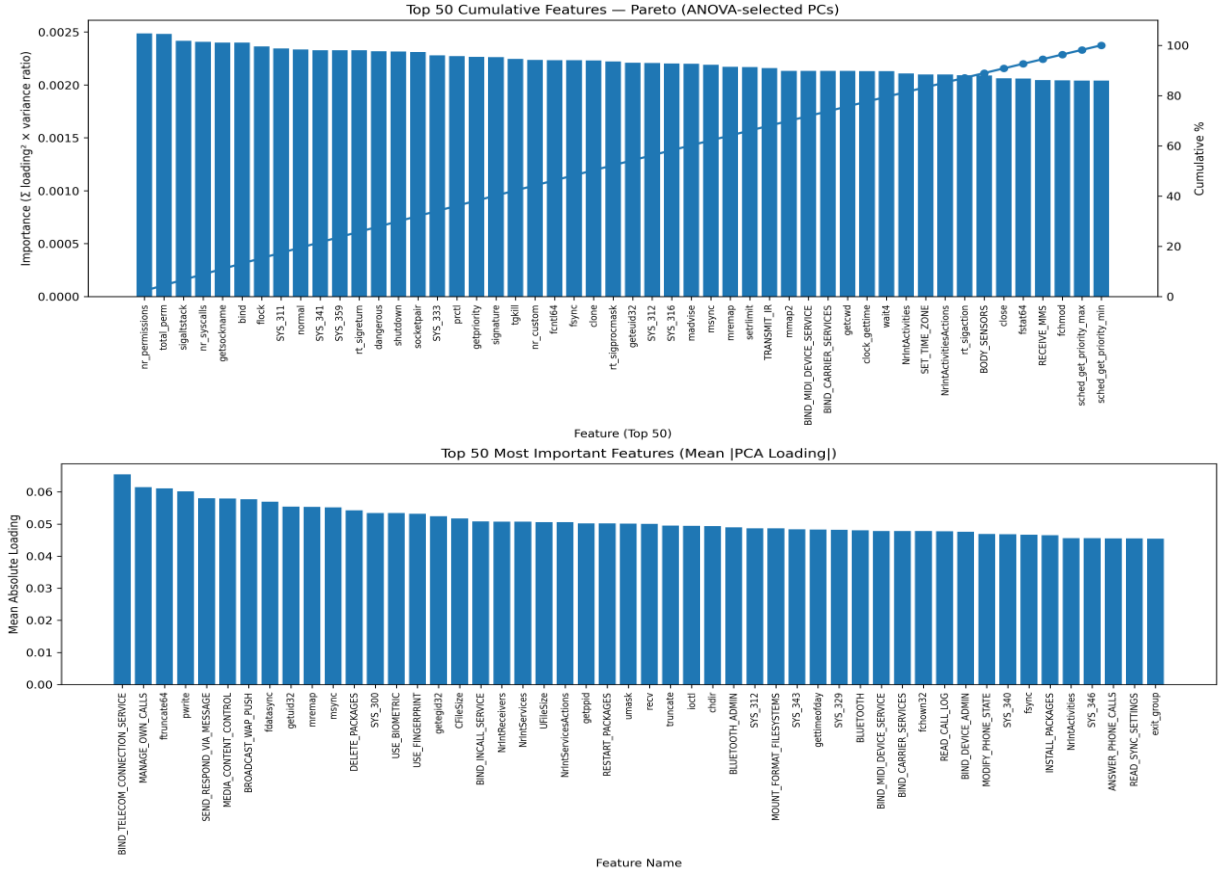


**Figure 9**. Feature Contribution Analysis (Krono Droid Dataset).

Key API calls such as System.loadLibrary and getDeviceId() indicated native library use and device fingerprinting, both signs of advanced malware. Permissions like RECEIVE_BOOT_COMPLETED, READ_CONTACTS, and READ_SMS helped the classifier identify different malware types. These main features enhanced detection performance and maintained the model's clarity and efficiency.

This result shows that a small group of key features can describe Android malware behavior well. By using these features, the PCA–ANOVA framework removed unnecessary and less useful data, creating a smaller, more effective feature set. Having most of the important information in just a few features explains why the CNN–BiGRU model performed well and used less computing power.

### *4.5. Malware Family Classification on the KronoDroid Dataset (Multiclass Evaluation)*

This section describes the performance of the proposed PCA to ANOVA to CNN–BiGRU framework on a 15-class malware family classification task using the KronoDroid dataset. Unlike the binary detection in Section 4.2, this experiment distinguishes benign applications from various malware families, including Adware, Trojan variants, Ransomware, Spyware, Scareware, Riskware, PUA, File-Infector, and Blank-Cat.

All KronoDroid feature vectors were reduced to 50 PCA components, and the top features were selected using ANOVA F-score analysis before training the model. This approach retains the most important differences in the data while reducing computational costs.

#### *4.5.1. Overall Multiclass Performance*

The proposed framework achieved a multiclass accuracy of 96%, demonstrating its ability to distinguish different types of malware behavior effectively. Major families such as Benign, Adware, Ransomware, Trojan-Spy, and Trojan-SMS exhibited high precision and F1-scores. This indicates that the CNN–BiGRU model can understand both small feature patterns and their sequences from PCA-reduced data. The weighted F1-score of 0.95 reflects a strong balance between precision and recall for most classes. The macro F1-score of approximately 0.75 suggests it performs fairly well on less common malware families.

**Table 10.** Multiclass performance on KronoDroid dataset.

| Class | Precision | Recall | F1-Score |
|---|---|---|---|
| Adware | 0.94 | 0.96 | 0.95 |
| Riskware | 0.93 | 0.94 | 0.93 |
| Trojan | 0.84 | 0.89 | 0.87 |
| Trojan-SMS | 0.96 | 0.95 | 0.95 |
| Trojan-Spy | 0.96 | 0.93 | 0.95 |
| Ransomware | 1.0 | 0.93 | 0.96 |
| Scareware | 1.0 | 0.19 | 0.32 |
| Benign | 0.99 | 0.99 | 0.99 |
| Trojan-Backdoor | 1.0 | 0.73 | 0.84 |
| Trojan-Banker | 0.97 | 0.95 | 0.96 |
| Trojan/Riskware | 0.94 | 0.95 | 0.94 |
| Trojan Dropper | 0.0 | 0.0 | 0.0 |
| PUA | 0.81 | 0.74 | 0.77 |
| File Infector | 0.88 | 0.66 | 0.75 |
| Spyware | 0.87 | 0.67 | 0.76 |
| Blank - cat | 0.70 | 0.04 | 0.08 |
| Weighted Avg | 0.96 | 0.96 | 0.95 |

Table 10 shows that benign samples achieved an F1-score of 0.99, confirming the model's ability to distinguish between malicious and normal Android behavior.

Malware families such as Trojan-Banker, Trojan-Spy, and Trojan-SMS also had F1-scores above 0.94. This strong performance likely results from the optimized feature space that retains important sequential behavioral features.

### *4.5.2. Effects of Class Imbalance*

Very small classes, such as Trojan-Dropper, had very few training samples, resulting in zero recall for those categories. This outcome appears to stem from severe class imbalance rather than any weakness in the proposed architecture.

Some samples from smaller malware families were incorrectly classified as similar Trojan subclasses. This likely happened because they shared similar permission requests and API usage patterns. Even with these challenges, the macro F1-score remained stable, indicating that the model's overall performance remained consistent.

### *4.5.3. Confusion Matrix and Family-Level Separability*

The confusion matrix, as shown in Figure 10, displays a clear diagonal structure, confirming the accurate assignment of most malware families and benign applications. Minor leakage has been observed among some malware families, such as Trojan variants sharing similar boot persistence and SMS-interception traits.

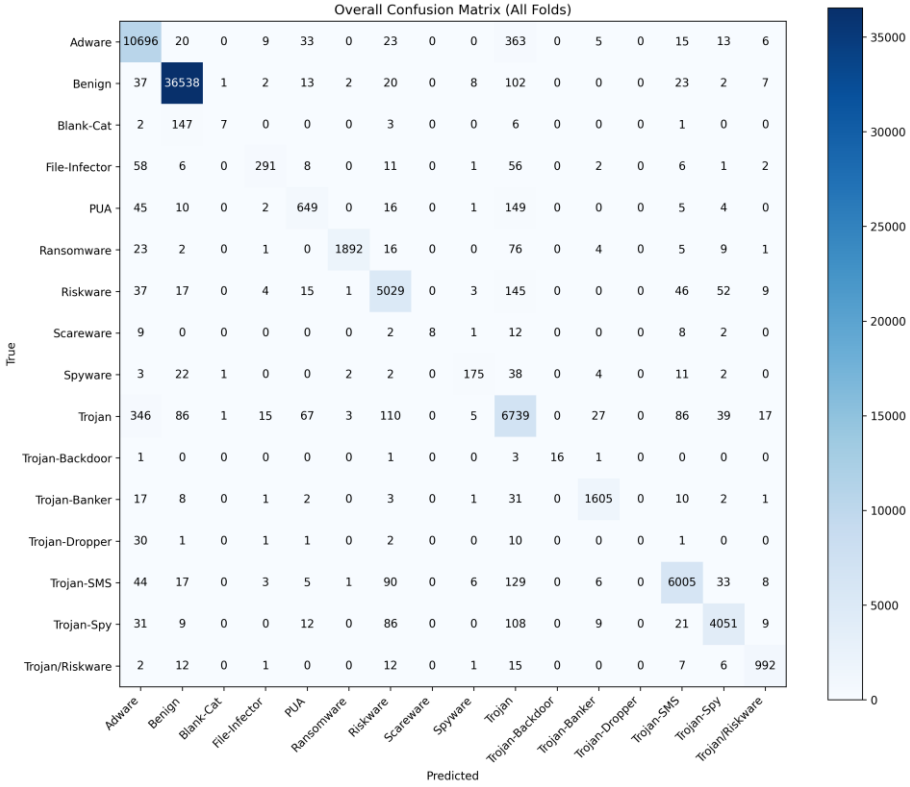


**Figure 10**. Confusion Matrix Malware category.

The ROC-AUC analysis, as shown in Figure 11, demonstrates the strong discriminative capability of the framework, proving the strong class separability. The model achieved a micro-average AUC of 0.999 and a macro-average AUC of 0.993, demonstrating consistent high performance across all classes. Malware families, including Trojan-Backdoor and Ransomware, have AUC values close to 1.

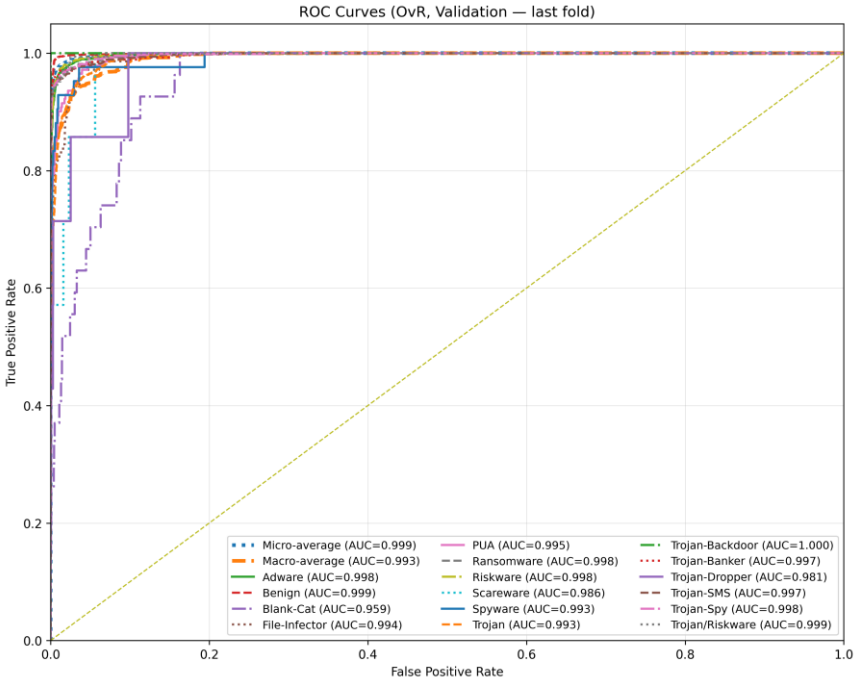


**Figure 11**. ROC-AUC Curve Malware category.

### *4.5.4. Convergence and Training Stability*

The training curves in Figure 12 show that the model learned smoothly and reached stable performance within 20 epochs. The validation accuracy settled around 0.96, and the validation F1-score stayed above 0.94 throughout training. The training and validation loss curves remained close, with only a small gap. This indicates that regularization was effective and the model generalized well, despite the dataset's class imbalance.

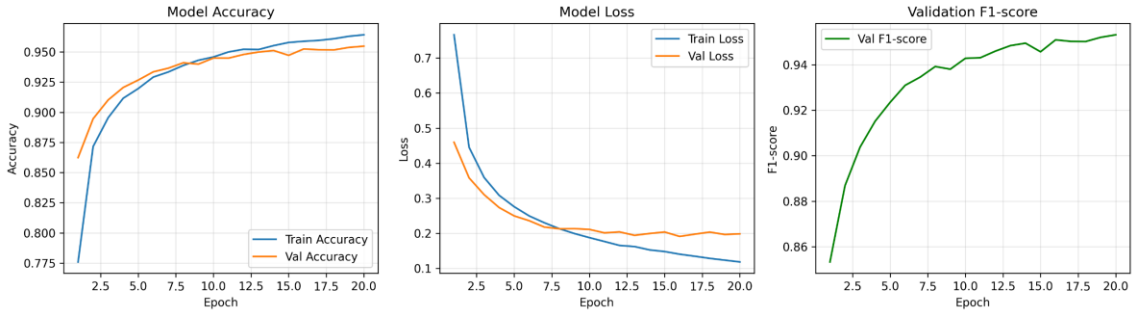


**Figure 12**. Validation accuracy (Malware category).

The reduced parameter footprint of BiGRU, combined with the dimensionality compression delivered by PCA and discriminative filtering via ANOVA, prevents overfitting and shortens model convergence time compared to heavier BiLSTM-based designs.

### *4.5.5. Summary*

The multiclass results on the KronoDroid dataset show that the PCA–ANOVA–CNN–BiGRU framework can clearly separate different malware families. The model maintains good accuracy while using less computational power. This indicates the framework's suitability for large-scale Android malware classification, even in systems with limited memory and processing resources.

### *4.6. Comparison with State-of-the-Art*

To contextualize the contribution of the proposed framework, a comparative assessment is conducted against recent state-of-the-art models reporting performance on the Drebin and KronoDroid benchmarks.

#### *4.6.1. Comparison with State-of-the-Art on the Drebin Dataset*

The framework has been tested using the Drebin dataset, which is the most widely used static dataset in Android malware research and offers a clear, reproducible baseline. We aimed to keep detection accuracy high while reducing preprocessing time and training costs. Recent studies using Drebin have achieved high detection rates, as shown in Table 11, by employing deeper recurrent models, multi-headed CNN pipelines, or hybrid static and dynamic setups. These methods slightly improved accuracy but required expensive preprocessing, GPU-intensive training, or large models that are difficult to run on resource-limited devices. Our approach uses a small static feature set (about 50 PCA–ANOVA ranked components) and a single-stream CNN–BiGRU model, maintaining high accuracy without heavy sequential processing.

**Table 11.** Comparison with State-of-the-Art (Drebin Dataset).

| Paper | Feature Strategy | Model Type | Dataset | Accuracy | Computational Cost |
|---|---|---|---|---|---|
| Morán, et al. [7] | PCA + Reduction | RF, SVM, XGBoost | Drebin | 95–96% | Moderate |
| Bhat and Dutta [5] | Info gain filtering | ML ensemble | Drebin | 96.3% | Higher |
| Akhtar and Feng [13] | No reduction | CNN-LSTM | Drebin | 99% | Very High |
| Zhang, et al. [18] | Word2Vec API embedding | CNN-BiGRU | Drebin | 98.3% | Moderately Higher |
| Ariffin and Casinto [14] | N-gram + Permission-based filtering and wrapper feature selection | Ensemble ML Model | Drebin | 95-99% | High |
| Proposed work | PCA + ANOVA | CNN-BiGRU | Drebin | 98% | Lowest |

Table 11 shows that LSTM and BiLSTM models need to evaluate several gates at each timestep, which uses more memory and slows down backpropagation. In contrast, reinforcement-based feature pruning and dynamic behavior modeling require repeated sampling or running execution traces, making them unsuitable for on-device use.

Using a 5-fold cross-validation protocol, the proposed model demonstrated competitive performance on the Drebin dataset compared to recent static deep learning approaches. Unlike methods that rely on dynamic logs or hybrid feature sets, this framework uses only static features, ensuring reproducibility and lower computational costs. The compact architecture also reduces training time while maintaining strong detection capabilities. Figure 13 illustrates a comparison of accuracy and computational costs across different Drebin dataset studies, highlighting that the proposed work achieves competitive accuracy with the lowest computational cost.

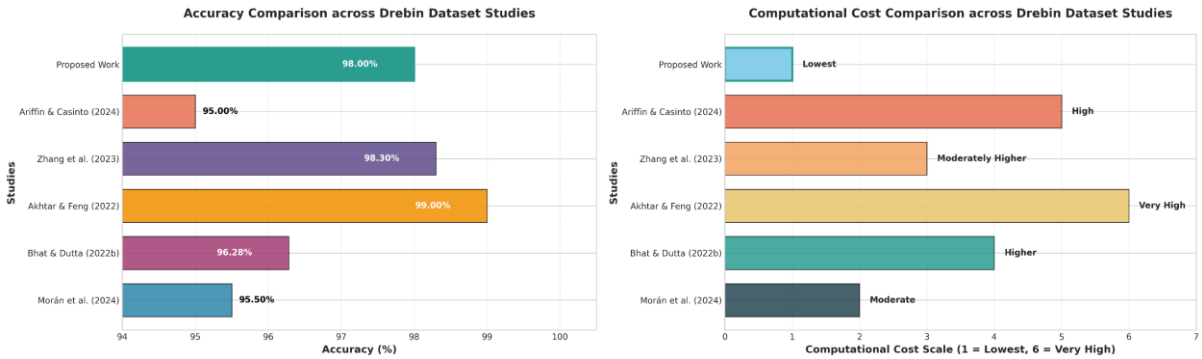


**Figure 13**. Accuracy vs Cost Trade-off (Drebin Dataset).

**Source:** Ariffin and Casinto [14]; Zhang, et al. [18]; Akhtar and Feng [13]; Bhat and Dutta [5] and Morán, et al. [7].

### *4.6.2. Comparison with State-of-the-Art on the KronoDroid Dataset (Binary and Multiclass)*

A focused comparison was conducted against recent studies using the KronoDroid dataset to evaluate both accuracy and computational demands. KronoDroid is widely adopted because it spans more than a decade of Android applications and exposes the temporal evolution of malware behavior. Table 12 shows a comparative study with previous work.

**Table 12.** Comparison with State-of-the-Art (KronoDroid Dataset).

| Study | Classification type | Feature strategy | Model type | Reported performance | Computational cost |
|---|---|---|---|---|---|
| Kılıç, et al. [27] | Binary | Factor analysis + broad learning | Broad learning network | 98.20% (real device), 97.90% (emulator) | High – wider latent expansion |
| AlSobeh, et al. [22] | Binary | Time-aware / time-agnostic temporal processing | Classical ML with temporal weighting | F1 ≈ 99.98% (time-agnostic), 91% avg (time-aware) | High – yearly retraining and temporal slicing |
| Waheed and Qadir [21] | Binary | ExtraTree feature ranking | Random Forest | 98.03% | Moderate |
| Proposed | Binary | PCA + ANOVA top-k (50 features) | CNN–BiGRU | 97% (binary KronoDroid) + high AUC | Lowest |

AlSobeh, et al. [22] reported very high binary performance using time-aware learning (F1 ≈ 99.98%). However, when they used yearly segmentation, the average performance dropped to about 91% and required frequent retraining. The proposed method does not depend on time-based data splitting and has shown stable performance in various settings of evaluation. This is more realistic and reliable for application in environments where the characteristics of data may change over time.

Waheed and Qadir [21] achieved 98.03% binary accuracy and gained 87.56% in multiclass accuracy with an extra tree-based random forest. The proposed model achieves 96% multiclass accuracy with only 50 compressed components. These findings show better family-level classification [27] reported 98% accuracy for their work, which used factor analysis with broad learning networks, but this approach required more memory for intent expansions.

Figure 14 compares recent binary malware detection studies based on accuracy, computational overhead, and model complexity. In panel (a), the PCA-optimized CNN–BiGRU achieves approximately 97% accuracy and avoids the performance drops seen in time-aware pipelines, which peak at about 99.98% but average around 91%. Panel (b) indicates that this method has the lowest computational cost compared to ensemble and broad-learning models. Panel (c) illustrates the shift from traditional models to the lightweight CNN–BiGRU. Overall, these results demonstrate that the proposed approach offers a good balance of accuracy, efficiency, and simplicity for binary malware detection on KronoDroid.

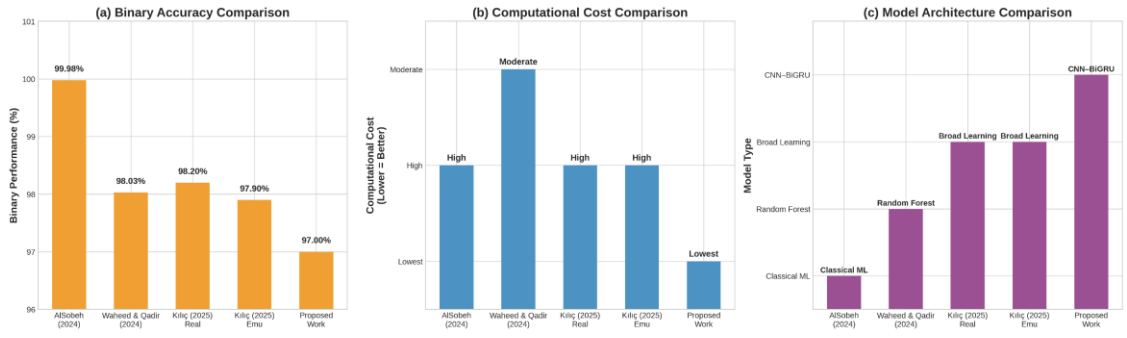


**Figure 14.** Accuracy vs Cost Trade-off (KronoDroid Dataset).

**Source:** AlSobeh, et al. [22], Waheed and Qadir [21], and Kılıç, et al. [27].

Table 13 presents a comparative study of multiclass classification (Malware category classification) on the KronoDroid dataset.

**Table 13.** Comparative study for Malware category classification for KronoDroid Dataset.

| Metric | Waheed and Qadir [21] | Kulkarni and Stamp [28] | Proposed work |
|---|---|---|---|
| Multiclass accuracy | 87.56% | 93.22% | 96.00% |
| F1-Score | 87.02% | 93.14% | 95.80% |
| Computational cost | Moderate | Moderate | Lowest |

The proposed model achieved competitive performance compared to earlier studies, with an F1-score of 95.80% and a multiclass accuracy of 96.00%. Some previous studies, like Waheed and Qadir [21] and Kulkarni and Stamp [28], have reported multiclass accuracies 87.56% and 93.22%, respectively. The model can capture both spatial and sequential patterns in malware behavior. It also has a lower computational cost and maintains high detection accuracy, enabling faster execution and better scalability. These findings report that the modified CNN-BiGRU framework is an effective and reliable solution for detecting and classifying Android malware in real-world scenarios. Figure 15 illustrates a comparison of multiclass accuracy, F1-score, and computational cost, showing that the proposed work achieves higher performance with lower computational cost compared to baseline models.

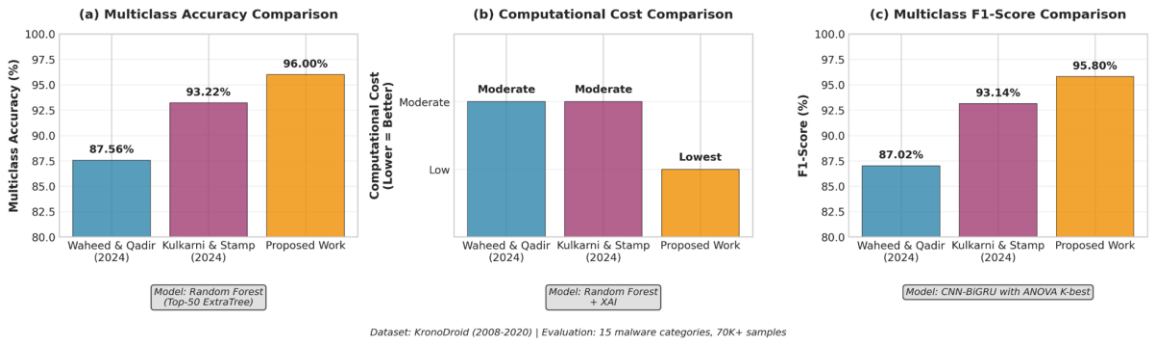


**Figure 15**. Accuracy Metrics Malware Category.

**Source:** Waheed and Qadir [21] and Kulkarni and Stamp [28].

### *4.7. Key Observations and Discussion*

The results on the Drebin and KronoDroid datasets show that Android malware can be detected effectively using a small set of important features from large datasets. The PCA–ANOVA method removes unnecessary features and selects the most useful ones quickly. The CNN–BiGRU model learns both local and sequential patterns while using fewer parameters than LSTM-based models. Since only static features are used, there is no need to run apps dynamically. Testing on both older and newer datasets, along with statistical analysis, shows that the framework is stable and dependable.

## 5. CONCLUSION

This paper presents an efficient method for Android malware classification. It combines PCA to reduce the number of features, ANOVA to select the most important ones, and a compact CNN–BiGRU model for classification. The results on the Drebin and KronoDroid datasets show that high detection accuracy can be achieved while lowering computational costs. By removing unnecessary and repeated features before training, the model becomes faster and more efficient. Additionally, statistical tests were performed to confirm that the results are reliable and valid.

### *5.1. Practical and Theoretical Implications*

The results of this study offer both theoretical and practical value. From a research perspective, the work demonstrates that combining feature reduction with a lightweight deep learning model can control model size while maintaining strong prediction performance. Instead of relying on very large feature sets or complex ensemble models, this framework achieves similar results using a smaller, more compact feature space.

From a practical point of view, the reduced number of features and simpler network design make the model suitable for real-world use. It can be applied in large-scale malware monitoring systems and may also be used for on-device Android detection, where memory and processing power are limited.

### *5.2. Limitations*

This study shows promising results, but it also has some limitations.

First, only static features were used. The model does not analyze how an app behaves while running, which could help in detecting new or advanced malware.

Second, real-world applications were not tested directly in this study. The computational cost was estimated through analytical calculations, and the model was not deployed on actual Android devices. Therefore, real-device performance was not evaluated.

### *5.3. Future Research Directions*

This work can be extended for future research directions by integrating dynamic system call traces or hybrid static-dynamic representations to improve robustness against new obfuscation techniques. Improvements in minority family recognition can be achieved by addressing class imbalances through class-weighted optimization, a focal loss function, or data-level resampling. The use of adaptive or federated learning frameworks can enhance scalability while maintaining users' privacy. Cross-dataset validation is also an area for further enhancement of this research.

**Funding:** This study received no specific financial support.
**Institutional Review Board Statement:** Not applicable.
**Transparency:** The authors state that the manuscript is honest, truthful, and transparent, that no key aspects of the investigation have been omitted, and that any differences from the study as planned have been clarified. This study followed all writing ethics.
**Competing Interests:** The authors declare that they have no competing interests.
**Authors' Contributions:** Both authors contributed equally to the conception and design of the study. Both authors have read and agreed to the published version of the manuscript.
**Disclosure of AI Use:** The author(s) used OpenAI's ChatGPT to edit and refine the wording of the Introduction section. All generated outputs were carefully reviewed, verified, and approved by the authors to ensure accuracy and integrity.

## REFERENCES

[1] R. Islam, M. I. Sayed, S. Saha, M. J. Hossain, and M. A. Masud, "Android malware classification using optimum feature selection and ensemble machine learning," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 100-111, 2023. https://doi.org/10.1016/j.iotcps.2023.03.001

[2] Q. Li, Q. Hu, Y. Qi, S. Qi, X. Liu, and P. Gao, "Semi-supervised two-phase familial analysis of Android malware with normalized graph embedding," *Knowledge-Based Systems*, vol. 218, p. 106802, 2021. https://doi.org/10.1016/j.knosys.2021.106802

[3] Z. Wang, K. Zeng, J. Wang, and D. Li, "FAGnet: Family-aware-based Android malware analysis using graph neural network," *Knowledge-Based Systems*, vol. 289, p. 111531, 2024. https://doi.org/10.1016/j.knosys.2024.111531

[4] V. K. Dwivedi and A. A. Waoo, "Advances in Android malware detection: Integrating static, dynamic, and hybrid techniques," *Journal of the Maharaja Sayajirao University of Baroda*, vol. 58, no. 1 (VIII), pp. 1–10, 2024.

[5] P. Bhat and K. Dutta, "A multi-tiered feature selection model for Android malware detection based on feature discrimination and information gain," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 9464-9477, 2022. https://doi.org/10.1016/j.jksuci.2021.11.004

[6] A. Pathak, T. S. Kumar, and U. Barman, "Static analysis framework for permission-based dataset generation and Android malware detection using machine learning," *EURASIP Journal on Information Security*, vol. 2024, no. 1, p. 33, 2024. https://doi.org/10.1186/s13635-024-00182-3

[7] P. Morán, A. Robles-Gómez, A. Duque, L. Tobarra, and R. Pastor-Vargas, "Machine learning models and dimensionality reduction for improving the Android malware detection," *PeerJ Computer Science*, vol. 10, p. e2616, 2024. https://doi.org/10.7717/peerj-cs.2616

[8] F. Ullah, S. Ullah, G. Srivastava, J. C.-W. Lin, and Y. Zhao, "NMal-Droid: Network-based Android malware detection system using transfer learning and CNN-BiGRU ensemble," *Wireless Networks*, vol. 30, no. 6, pp. 6177-6198, 2024. https://doi.org/10.1007/s11276-023-03414-5

[9] A. Lakshmanarao and M. Shashi, "Android malware detection with deep learning using RNN from opcode sequences," *International Journal of Interactive Mobile Technologies*, vol. 16, no. 1, pp. 145–157, 2022. https://doi.org/10.3991/ijim.v16i01.26433

[10] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based Android malware detection using real devices," *Computers & Security*, vol. 89, p. 101663, 2020. https://doi.org/10.1016/j.cose.2019.101663

[11] J. Lee, H. Jang, S. Ha, and Y. Yoon, "Android malware detection using machine learning with feature selection based on the genetic algorithm," *Mathematics*, vol. 9, no. 21, p. 2813, 2021. https://doi.org/10.3390/math9212813

[12] H. S. K Kauser and M. V. Anu, "Hybrid deep learning model for accurate and efficient Android malware detection using DBN-GRU," *PLoS ONE*, vol. 20, no. 5, p. e0310230, 2025. https://doi.org/10.1371/journal.pone.0310230

[13] M. S. Akhtar and T. Feng, "Detection of malware by deep learning as CNN-LSTM machine learning techniques in real time," *Symmetry*, vol. 14, no. 11, p. 2308, 2022. https://doi.org/10.3390/sym14112308

[14] N. A. M. Ariffin and H. P. Casinto, "Android malware detection using permission based static analysis," *Journal of Advanced Research in Applied Sciences and Engineering Technology*, vol. 33, no. 3, pp. 86–97, 2024. https://doi.org/10.37934/araset.33.3.8697

[15] D. Ö. Şahin, O. E. Kural, S. Akleylek, and E. Kılıç, "A novel permission-based Android malware detection system using feature selection based on linear regression," *Neural Computing and Applications*, vol. 35, no. 7, pp. 4903-4918, 2023. https://doi.org/10.1007/s00521-021-05875-1

[16] T. Lu, Y. Du, L. Ouyang, Q. Chen, and X. Wang, "Android malware detection based on a hybrid deep learning model," *Security and Communication Networks*, vol. 2020, no. 1, p. 8863617, 2020. https://doi.org/10.1155/2020/8863617

[17] A. R. Nasser, A. M. Hasan, and A. J. Humaidi, "DL-AMDet: Deep learning-based malware detector for Android," *Intelligent Systems with Applications*, vol. 21, p. 200318, 2024. https://doi.org/10.1016/j.iswa.2023.200318

[18] Y. Zhang, S. Yang, L. Xu, X. Li, and D. Zhao, "A malware detection framework based on semantic information of behavioral features," *Applied Sciences*, vol. 13, no. 22, p. 12528, 2023. https://doi.org/10.3390/app132212528

[19] Y. Wu *et al.*, "DroidRL: Feature selection for Android malware detection with reinforcement learning," *Computers & Security*, vol. 128, p. 103126, 2023. https://doi.org/10.1016/j.cose.2023.103126

[20] A. Pathak, U. Barman, and T. S. Kumar, "Machine learning approach to detect Android malware using feature-selection based on feature importance score," *Journal of Engineering Research*, vol. 13, no. 2, pp. 712-720, 2025. https://doi.org/10.1016/j.jer.2024.04.008

[21] M. Waheed and S. Qadir, "Effective and efficient Android malware detection and category classification using the enhanced KronoDroid dataset," *Security and Communication Networks*, vol. 2024, no. 1, p. 7382302, 2024. https://doi.org/10.1155/2024/7382302

[22] A. M. R. AlSobeh, K. Gaber, M. M. Hammad, M. Nuser, and A. Shatnawi, "Android malware detection using time-aware machine learning approach," *Cluster Computing*, vol. 27, no. 9, pp. 12627-12648, 2024. https://doi.org/10.1007/s10586-024-04484-6

[23] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, "DIDroid: Android malware classification and characterization using deep image learning," in *Proceedings of the 2020 10th International Conference on Communication and Network Security (ICCNS)*, Tokyo, Japan, 2020, pp. 70–82.

[24] G. Karat, J. M. Kannimoola, N. Nair, A. Vazhayil, V. G. Sujadevi, and P. Poornachandran, "CNN-LSTM hybrid model for enhanced malware analysis and detection," *Procedia Computer Science*, vol. 233, pp. 492-503, 2024. https://doi.org/10.1016/j.procs.2024.03.239

[25] N. Polatidis, S. Kapetanakis, M. Trovati, I. Korkontzelos, and Y. Manolopoulos, "FSSDroid: Feature subset selection for Android malware detection," *World Wide Web*, vol. 27, no. 5, p. 50, 2024. https://doi.org/10.1007/s11280-024-01287-y

[26] A. Mahindru *et al.*, "PermDroid a framework developed using proposed feature selection approach and machine learning techniques for Android malware detection," *Scientific Reports*, vol. 14, no. 1, p. 10724, 2024. https://doi.org/10.1038/s41598-024-60982-y

[27] K. Kılıç, İ. Atacak, and İ. A. Doğru, "FABLDroid: Malware detection based on hybrid analysis with factor analysis and broad learning methods for Android applications," *Engineering Science and Technology, an International Journal*, vol. 62, p. 101945, 2025. https://doi.org/10.1016/j.jestch.2024.101945

[28] M. Kulkarni and M. Stamp, "*XAI and Android malware models*," in *Machine Learning, Deep Learning and AI for Cybersecurity*, M. Stamp and M. Jureček, Eds. Cham, Switzerland: Springer, 2025, pp. 327–355.