

Article

LEMSOFT: Leveraging Extraction Method and Soft Voting for Android Malware Detection

Qiang Han ^{1,2} , Zhichao Shi ^{1,2}, Yao Li ^{3,*}  and Tao Zhang ³ ¹ School of Computer Science and Engineering, North Minzu University, No. 204, Minzu South Street, Yinchuan 750021, China; hanqiang@nmu.edu.cn (Q.H.); 20237486@stu.nmu.edu.cn (Z.S.)² Key Laboratory of Image and Graphics Intelligent Processing of State Ethnic Affairs Commission, North Minzu University, Wenchang Street, Yinchuan 750021, China³ School of Computer Science and Engineering, Macau University of Science and Technology, Avenida Wai Long, Taipa, Macao SAR 999078, China

* Correspondence: 2109853gia30001@student.must.edu.mo

Abstract

The pervasive spread of Android malware poses significant threats to users and systems worldwide. In most existing studies, differences in feature importance are often overlooked, and the calculation of feature weights is conducted independently of the classification model. In this paper, we propose an Android malware detection method, Leveraging Extraction Method and Soft Voting classification (LEMSOFT). This approach includes a novel preprocessing module, lexical occurrence ratio-based filtering (LORF), and an improved Soft Voting mechanism optimized through genetic algorithms. We introduce LORF to evaluate and enhance the significance of permissions, API calls, and opcodes. Each type of feature is then independently classified using tailored machine learning models. To integrate the outputs of these classifiers, this paper proposes an innovative soft voting mechanism that improves prediction accuracy for encountered applications by assigning weights through a genetic algorithm. Our solution outperforms the baseline methods we studied, as evidenced by the evaluation of 5560 malicious and 8340 benign applications, with an average accuracy of 99.89%. The efficacy of our methodology is demonstrated through extensive experiments, showcasing significant improvements in detection rates compared to state-of-the-art (SOTA) methods.

Keywords: Android malware; feature selection; machine learning; static detection; soft voting

MSC: 68N30



Academic Editor: Jonathan Blackledge

Received: 13 October 2025

Revised: 1 November 2025

Accepted: 4 November 2025

Published: 6 November 2025

Citation: Han, Q.; Shi, Z.; Li, Y.; Zhang, T. LEMSOF: Leveraging Extraction Method and Soft Voting for Android Malware Detection.

Mathematics **2025**, *13*, 3569.

<https://doi.org/10.3390/math13213569>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the proliferation of smartphones and the widespread adoption of the Android operating system, Android devices have become integral to daily life and work. Currently, Android accounts for about 72.15% of the global mobile operating system market share (<https://gs.statcounter.com/os-market-share/mobile/worldwide>) accessed on 1 October 2025. The open nature of the Android ecosystem, coupled with its extensive user base, has led to a significant increase in the number and sophistication of Android malware. The rapid growth of Android malware poses a significant threat to users' personal information security and can lead to economic losses and a crisis of trust [1]. Traditional malware detection methods often struggle to keep pace with the rapidly evolving landscape of

Android malware [2]. Therefore, there is an urgent need for innovative approaches that can effectively identify and neutralize such threats.

Current research in Android malware detection has made significant strides by leveraging advanced machine learning techniques and comprehensive feature extraction methods to identify malicious applications. Notably, MADAM [3] employs a multi-level analysis approach to detect and prevent over 96% of malicious applications through behavior-based techniques. DroidCat [4] enhances the detection and categorization of malware by profiling app behaviors and using machine learning, achieving robust performance. Furthermore, PermPair [5] leverages permission usage patterns to distinguish between malicious and benign apps effectively. These innovations collectively enhance the accuracy and robustness of Android malware detection, providing crucial security improvements for the platform.

Notwithstanding these advancements, current approaches remain constrained by notable limitations. Feature selection plays a critical role in these approaches, directly impacting the detection performance of the models. For example, Qiu et al. [6] introduced Cyber Code Intelligence for Android Malware Detection. Despite its innovative approach, the method fails to differentiate the relative importance of the various features, assuming equal significance for all extracted features. This deficiency may result in a less optimized detection model. Similarly, Xu et al. [7] presented a method that calculates feature weights separately from the classification model, using a fixed set of weights that do not dynamically adjust to the varying relevance of features across different datasets or malware types. In view of these developments, accurately assessing and dynamically adjusting the relative importance of features is essential to improving the accuracy of malware detection systems. In light of these considerations, we propose an important question: **What methodologies might be employed to address these limitations and thereby enhance the accuracy of malware detection systems?**

To address these limitations, we identified two main challenges that needed to be overcome. **Challenge 1: The varying importance of features is frequently ignored and feature weights are computed independently of the classification model.** This oversight can lead to suboptimal performance in malware detection systems. For example, some studies primarily focus on the selection of features without considering their varying impacts on different classifiers [8–10]. **Challenge 2: Many approaches calculate feature weights in isolation, failing to integrate them with the classification process, which can result in less effective models [11–13].** Integrating these aspects can provide a more holistic approach, enhancing the ability to accurately detect malware by taking into account the relative importance of different features. Despite the potential benefits, there have been few studies addressing this integrated approach.

To overcome these challenges, we propose the LEMSOF method, a novel approach for Android malware detection. The LEMSOF method leverages lexical occurrence ratio-based filtering to address **Challenge 1** and a soft voting mechanism optimized through genetic algorithms to tackle **Challenge 2**. The LORF method is designed to evaluate and enhance the significance of various features, including permissions, API calls, and opcodes. By accurately assessing the importance of each feature type, LORF enables the extraction of the most relevant data for malware detection. Each feature type is then independently classified using tailored machine learning models, allowing us to utilize the strengths of different classifiers and optimize their performance for specific feature sets. To integrate the outputs of these classifiers, we propose an innovative soft voting mechanism that improves prediction accuracy for encountered applications by assigning weights through a genetic algorithm. The genetic algorithm dynamically adjusts the weights based on the relative importance of each classifier's output, ensuring a more accurate and robust detection process.

Our solution outperforms the baseline methods, as evidenced by the evaluation of 5560 malicious applications from the Drebin [14] dataset and 8340 benign applications from the Google Play Store (<https://play.google.com/store>) accessed on 1 October 2025, achieving an average accuracy of 99.89% and an F1 score of 99.68%. This remarkable performance underscores the efficacy of our methodology and highlights the potential for substantial advancements in Android malware detection.

The main contributions of this work are as follows:

- We propose LEMSOF, a novel approach for Android malware detection that integrates LORF and a genetic algorithm-optimized soft voting mechanism. This method addresses two significant challenges in existing approaches: the varying importance of features and the independent calculation of feature weights.
- The LORF method is designed to accurately evaluate and enhance the significance of various features. By assessing the importance of each feature type, LORF enables the extraction of the most relevant data for malware detection, improving the accuracy of the detection process.
- Our approach employs tailored machine learning models for each feature type, allowing for the utilization of the strengths of different classifiers. The innovative soft voting mechanism integrates the outputs of these classifiers by assigning weights through a genetic algorithm, adjusting based on the relative importance of each classifier's output.

In the following sections, we provide a comprehensive overview of our research methodology and findings. Section 2 presents related work, discussing existing methods and their limitations in the context of Android malware detection. Section 3 details the detection method of LEMSOF, including data collection, preprocessing, and the application of machine learning classifiers. Section 4 describes the experimental settings, including the datasets used, the machine learning algorithms employed, and the evaluation metrics. Section 5 presents the results and discussion, comparing the performance of LEMSOF with baseline methods and analyzing the effectiveness of our approach. Section 6 offers a performance analysis, explaining the key factors contributing to our method's success. Section 7 discusses potential threats to validity and how they were mitigated. Finally, Section 8 concludes the paper and outlines potential directions for future research.

2. Related Work

In recent years, significant progress has been made in the field of Android malware detection. Researchers have developed various methods to enhance detection accuracy against sophisticated malware. These methods can be broadly categorized into three types. In the following sections, we describe related work in these three broad categories.

2.1. Static Detection

Static analysis has been a cornerstone in the detection of Android malware due to its ability to examine an application's code without executing it. This method relies on extracting and analyzing various features from the APK (Android Package) files, such as permissions, API calls, control flow graphs, and data flow analysis [15–18]. MaMaDroid [19] employs a static analysis approach, utilizing machine learning to analyze Android application bytecode for the identification of malicious activities. In order to increase overall performance, Idrees et al. [20] proposed PIndroid, an Android app classifier that leverages intent APIs and permissions for classifier fusion training. However, these techniques often fail to adequately consider the varying importance of features, an issue that our method effectively addresses. Additionally, our method integrates a genetic algorithm-optimized

soft voting mechanism that dynamically adjusts classifier weights based on the relevance of each feature type.

2.2. Dynamic Detection

Dynamic analysis, in contrast to static analysis, involves executing the application in a controlled environment to monitor its runtime behavior. This approach is particularly effective in identifying behaviors that are not evident from the static code alone, such as network activity, file manipulations, and API calls made during execution [21–24]. Melvin et al. [25] released a cloud-focused intrusion dataset using VMM-based introspection to capture malware behavior, demonstrating strong detection performance; Li et al. [26] introduced DMalNet, which integrates API argument encoding with API call graph learning via Graph Neural Networks; and Li et al. [27] proposed a deep framework leveraging API sequence embeddings, convolutional “phrases,” semantic chains, and Bi-LSTM to model intrinsic API features, achieving high accuracy and F1-score on real-world samples. However, dynamic monitoring is resource-intensive, and it may also be less effective against malware that can detect and adapt to runtime analysis environments.

2.3. Hybrid Detection

Hybrid analysis combines the strengths of both static and dynamic analysis to provide a more comprehensive malware detection framework. This approach addresses the limitations inherent in each method when used in isolation. By integrating static and dynamic features, hybrid analysis systems can achieve higher detection rates and robustness against evasion techniques [28–31]. Surendran et al. [32] introduced a novel methodology for Android malware detection that integrates both static and dynamic analysis using a Tree Augmented Naive Bayes model to effectively capture the conditional dependencies among API calls, permissions, and system calls, achieving a high detection accuracy of 0.97 over a long-term evaluation period. Fan et al. [33] present a comprehensive approach to Android malware detection by leveraging a combination of static analysis of application permissions and dynamic analysis of runtime behaviors, which allows for an adaptive and robust defense mechanism against evolving threats. Tong and Yan [34] propose an innovative hybrid methodology for Android malware detection that combines dynamic analysis to collect runtime system call data and static analysis to process this data on a detection server, achieving superior detection accuracy through the use of both malicious and normal pattern sets.

3. Methodology

In this section, we present the overall architecture of LEMSOF, illustrated in Figure 1. We begin by collecting a comprehensive dataset of Android applications, including both benign and malicious samples, from various reputable sources to ensure diversity and representativeness. During feature preprocessing, we employ the LORF method to evaluate and enhance the significance of permissions, API calls, and opcodes, which helps in reducing data dimensionality and focusing on the most relevant features. Each feature type is then classified using tailored machine learning models optimized for their specific characteristics. To integrate the outputs of these classifiers, we use an innovative soft voting mechanism that assigns weights through a genetic optimization algorithm. Each subsection elaborates on the specific techniques utilized to enhance the accuracy and efficiency of our malware detection system.

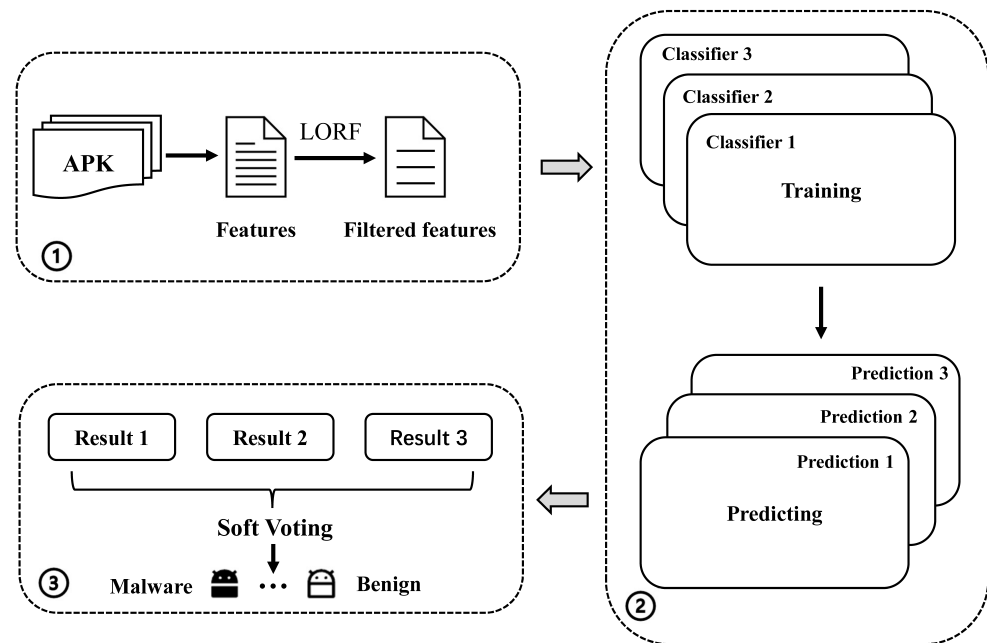


Figure 1. Overall framework of LEMSOF. Our approach consists of three key steps: (1) APK files undergo feature extraction and filtering using the LORF method, resulting in a refined set of features. (2) Multiple classifiers are trained on these filtered features, each generating independent predictions. (3) A combination of a soft voting mechanism and a genetic algorithm is used to assign weights to the classifiers' predictions, leading to the final classification of the application as either malicious or benign.

3.1. Data Collection

The data collection phase involves gathering a comprehensive dataset of Android applications, including both benign and malicious samples. Specifically, we collect 5560 malicious applications from the Drebin [14] dataset and 8340 benign applications from the Google Play Store. All Drebin samples, including those within the same family, exhibit distinct hash values, suggesting the use of obfuscation techniques to hinder the detection of variations. These applications represent a diverse range of categories and sources to ensure the generalization and effectiveness of the detection system.

3.2. Preprocessing

The sensitivity coefficient is calculated to indicate the degree of maliciousness associated with specific features in executing malicious behavior. The measurement is biased if the coefficients of features are calculated solely based on their frequency of occurrence in a malicious dataset. To address this issue, we propose a method called LORF to accurately calculate the measure of feature sensitivity.

In the LORF method, we design four basic metrics to evaluate the importance of features: $mc(f_i)$, $bc(f_i)$, $mr(f_i)$, and $br(f_i)$, where both $mc(f_i)$ and $bc(f_i)$ are initially set to 0. Finally, we develop a formula $cme(f_i)$ to calculate the sensitivity coefficient of the feature f_i based on these four metrics. t_m represents the total number of applications in the malware dataset, t_b represents the total number of applications in the benign dataset, and a_j represents the j -th application. The detailed definitions and explanations of these four basic metrics and the formula are as follows:

$$mc(f_i) = \sum_{j=1}^{t_m} a_j \quad (1)$$

$mc(f_i)$ represents the malicious count of f_i . Denoting the number of malware instances that utilize f_i in the malicious dataset. If the j -th application utilizes the f_i feature, then a_j is 1; otherwise, a_j is 0.

$$bc(f_i) = \sum_{j=1}^{t_b} a_j \quad (2)$$

$bc(f_i)$ represents the benign count of f_i . Denoting the number of benign applications that utilize f_i in the benign dataset. If the j -th application utilizes the f_i feature, then a_j is 1; otherwise, a_j is 0.

$$cme(f_i) = \frac{mc(f_i)}{t_m} \cdot e^{-\frac{bc(f_i)}{t_b}} \quad (3)$$

In this equation, $cme(f_i)$ represents the coefficient of maliciousness evaluation, quantifying a feature's significance in identifying malicious applications. Here, $mc(f_i)$ denotes the number of occurrences of feature f_i in malicious instances, while t_m is the total number of malware instances in the dataset. The term $e^{-\frac{bc(f_i)}{t_b}}$ incorporates the benign occurrence ratio, where $bc(f_i)$ is the count of feature f_i in the benign dataset and t_b represents the total number of benign instances (8340 in our study). This formulation emphasizes that the importance of a feature in detecting malware is influenced both by its prevalence in malicious samples and its rarity in benign samples, enhancing the overall evaluation of its effectiveness in distinguishing between benign and malicious applications.

In traditional methods, the importance of a feature in evaluating malicious software is often assumed to correlate with its occurrence in malicious applications, denoted as $mc(f_i)$ [35,36]. For instance, the feature `openConnection()` is commonly found in both benign and malicious software, leading to high values for both $mr(f_i)$ and $br(f_i)$. However, it is not particularly discriminative. In contrast, the feature `getLineNumber()` exhibits a significantly lower $br(f_i)$ compared to `openConnection()`, despite also having a lower $mr(f_i)$. Figure 2 shows the count of selected features that frequently appear in malware samples but are less common in benign applications. This suggests that `getLineNumber()` may be more critical for identifying malicious applications, as its lower prevalence in benign software enhances its value as an indicator of malicious behavior. Therefore, we posit that the $cme(f_i)$ of a feature is positively correlated with its frequency ratio in $mr(f_i)$ and negatively correlated with its frequency ratio in $br(f_i)$.

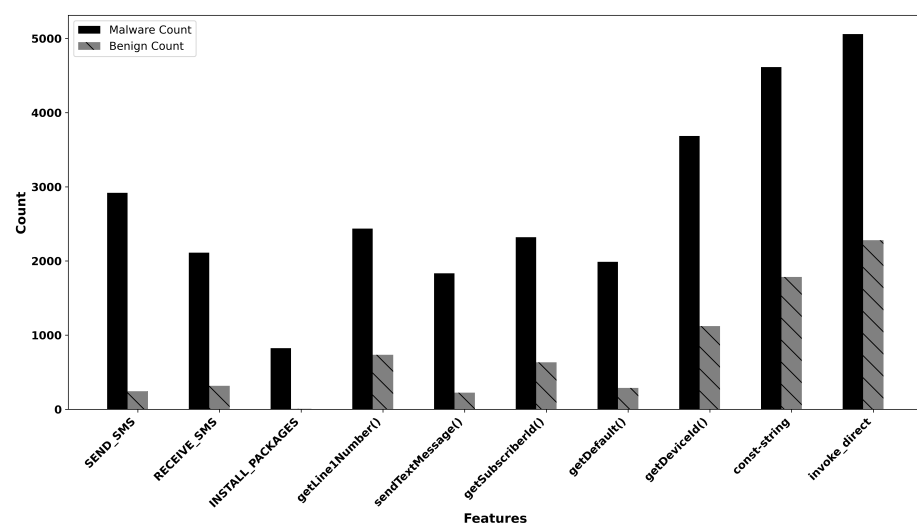


Figure 2. Distribution of high-variance features in malware and benign applications. The black bars represent the count in malware, while the grey bars indicate the count in benign applications.

3.3. Classifier

The soft voting mechanism is an ensemble learning method commonly used for classification tasks. The fundamental idea is to enhance the performance of the overall model by combining the predicted probabilities from multiple base classifiers. Unlike hard voting, soft voting takes into account the predicted probability of each class by each classifier, rather than simply the majority vote. In the soft voting mechanism, multiple base classifiers output a probability distribution for each class. Suppose there are N base classifiers, and each classifier i outputs a predicted probability $p_{i,c}$ for class c (where c refers to the class, benign or malicious). The aggregated predicted probability P_c for class c by the soft voting ensemble model can be expressed as the average or weighted average of these predicted probabilities, given by:

$$P_c = \sum_{i=1}^N w_i \cdot p_{i,c} \quad (4)$$

where w_i is the weight of the i -th classifier, satisfying $\sum_{i=1}^N w_i = 1$. The final classification result is the class with the highest aggregated predicted probability:

$$\hat{y} = \arg \max_a P_a \quad (5)$$

In our model, the number of classifiers k is set to 3, corresponding to the three features: permission, API call, and opcode, while the number of classes c is set to 2, corresponding to benign and malicious classifications. Each feature is independently classified using its dedicated classifier. To integrate the outputs of these classifiers, we introduce an innovative soft voting mechanism [37]. This mechanism combines the individual predictions of each classifier by assigning weights to their outputs based on a genetic optimization algorithm. The genetic algorithm iteratively adjusts these weights to enhance overall classification performance. Refer to Algorithm 1 for detailed steps of the algorithm.

Algorithm 1 Application Classification Algorithm

```

1: Input: set of Apps  $\{A_1, A_2, \dots, A_n\}$ , a dataset
2: Output: Outputs each app as Malware or Normal
3: for each  $A_i$  do
4:   Extract its permissions, API calls, and opcodes
5:   Calculate probabilities  $p_1, p_2, p_3$  for permissions, API calls, and opcodes
6:   Initialize weights  $w_p, w_a, w_o$  and threshold  $T$ 
7:   repeat
8:     Optimize  $w_p, w_a, w_o, T$ 
9:     Calculate final probability  $P_k$ :

$$P_k = w_p p_1 + w_a p_2 + w_o p_3$$

10:  until the improvement in accuracy is less than 0.01% over 10 consecutive iterations
11:  if  $P_k \geq T$  then
12:    Mark  $A_i$  as malware
13:  else
14:    Mark  $A_i$  as normal
15:  end if
16: end for

```

Machine learning, known for its extensive application in data analysis, pattern recognition, and bioinformatics, excels in handling classification problems. It offers computational efficiency and reduced time complexity, making it an ideal choice for our classification phase. During the training phase, we extract binary vectors for each feature from the test set. These vectors, along with their corresponding labels, are used to train the machine

learning algorithm. After training, the resultant model is saved for future use. In the prediction phase, the application under test undergoes the same preprocessing steps to obtain its feature vectors, which are then fed into the model to predict the application's nature (0 for benign and 1 for malicious). In Section 5.2, we examine various machine learning algorithms for each feature classifier, including Support Vector Machine (SVM) [38], Random Forest (RF) [39], and k-Nearest Neighbor (KNN) [40], among others. The best classifier for each feature is then selected. We integrate these optimal classifiers using a soft voting mechanism. Additionally, we compare the soft voting mechanism against single machine learning classifiers, hard voting algorithms, a feature-filtering-based malware detection method, and state-of-the-art malware detection techniques to evaluate its efficacy in achieving superior detection results.

4. Experimental Settings

In this section, we describe the experimental settings used to evaluate the performance of our proposed malware detection system. This section provides a comprehensive overview of the dataset used for our experiments, the machine learning algorithms applied, and the metrics employed to evaluate the performance of our malware detection system. In this paper, all the experiments are conducted with Python 3.7, compiled by Pycharm 2023.2, and run on a PC with 64-bit Windows 10, an Intel(R) Xeon(R) Gold 6154 CPU, and NVIDIA TITAN V GPUs.

4.1. Experimental Dataset

The experimental dataset consists of 5560 malicious applications from the Drebin [14] dataset and 8340 benign applications from the Google Play Store. To ensure the integrity and quality of the dataset, we perform several preprocessing steps, including the removal of duplicate samples and the extraction of relevant features. These features include permissions, API calls, and opcodes, which are critical for accurately distinguishing between benign and malicious applications. All Drebin samples exhibit distinct hash values, indicating the use of obfuscation techniques to hinder the detection of variations. To create a balanced and effective model, we divide the dataset into training and testing sets, allocating 80% for training and 20% for testing. The diverse range of categories and sources represented in this dataset ensures the generalization and effectiveness of the detection system.

4.2. Machine Learning Algorithms

We employ several commonly used machine learning algorithms to conduct classification experiments. These algorithms include Logistic Regression (LR) [41], SVM [38], KNN [40], RF [39], Decision Tree (DT) [42], Category Boosting (CatBoost) [43], and Adaptive Boosting (AdaBoost) [44]. Specifically, for the KNN algorithm, we utilize 1-NN, 3-NN, and 5-NN, selecting three widely adopted values of K . In total, we implement nine distinct machine learning algorithms. Each of these algorithms is tested across three different feature sets to determine the optimal classifier for each feature set.

4.3. Experimental Metrics

To evaluate the performance of our proposed method, we employ several key metrics commonly used in malware detection and machine learning. These metrics provide a comprehensive view of the model's effectiveness, accuracy, and reliability. In the context of classification models, the terms True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) are defined as follows:

- True Positive (TP): Instances where the model correctly predicts the positive class.
- True Negative (TN): Instances where the model correctly predicts the negative class.

- False Positive (FP): Instances where the model incorrectly predicts the positive class for a negative instance.
- False Negative (FN): Instances where the model incorrectly predicts the negative class for a positive instance.

In many studies on binary classification, metrics such as accuracy, recall, F1 score, and precision are commonly utilized to evaluate performance.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (6)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (7)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (8)$$

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (9)$$

Accordingly, we adopt these four metrics to assess the effectiveness of our method.

5. Results and Discussion

In this section, we explore selecting the appropriate feature thresholds, determining the best classifiers, and assigning weights to these classifiers to enhance the accuracy of malware detection. In this paper, we investigate five key research questions (RQs) to validate the effectiveness of our method from various perspectives.

- RQ1:** What is the optimal threshold for LORF for permissions, API calls, and opcodes?
RQ2: What is the best classifier for permissions, API calls, and opcodes?
RQ3: What is the weight of the optimal classifier for permissions, API calls, and opcodes?
RQ4: How effective is the combination of permissions, API calls, and opcodes?
RQ5: Can LEMSOFTE outperform the baseline methods?

To thoroughly investigate our research questions, we have organized our analysis into several subsections. Each subsection addresses a specific research question in detail. In the following subsections, we provide comprehensive answers based on our experimental results and analysis.

5.1. Answer to RQ1: Determining Optimal Thresholds

Motivation: Determining the optimal thresholds for each LORF feature is crucial for accurately filtering and selecting relevant features to enhance malware detection performance.

Methodology: We calculate the coefficient of maliciousness evaluation $cme(f_i)$ for each feature f_i using LORF, sorting features in descending order of their $cme(f_i)$ values. Features with $cme(f_i)$ values greater than 0.3 were selected directly. For features with $cme(f_i)$ values ranging from 0 to 0.3, we apply a threshold selection approach with a step size of 0.02 for filtering. The selected features are then classified using an SVM classifier to compute accuracy.

Results: Figure 3 illustrates the threshold values used for filtering features via LORF. Our experiments demonstrate that the optimal performance for the permissions is achieved with a threshold of 0.08, resulting in 76 features. For the API calls, the best performance is observed with a threshold of 0.16, yielding 134 features. For the opcodes, a threshold of 0.22 provides the best results with 77 features. Table 1 demonstrates that applying LORF significantly improves the performance metrics across all features. For example, using LORF for API calls increases the accuracy from 96.53% to 99.15%, recall from 96.05% to

98.62%, F1 score from 96.34% to 98.82%, and precision from 96.57% to 99.02%. Similar improvements are observed for Permission and Opcode.

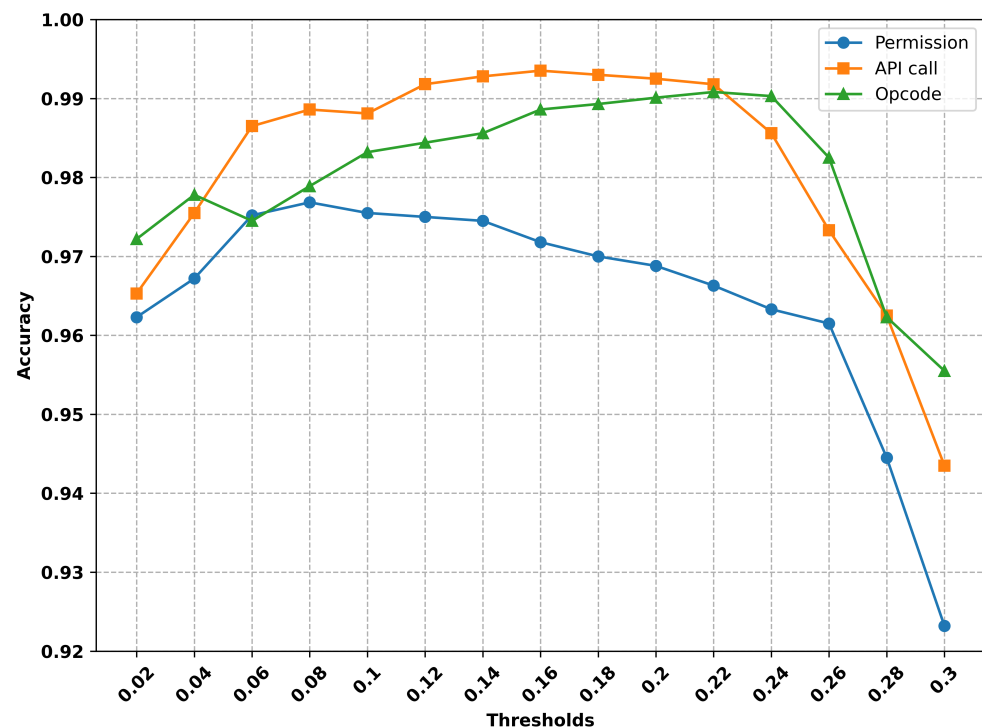


Figure 3. Accuracy of each feature at different LORF thresholds using the SVM model. The abscissa represents the different thresholds, and the ordinate represents the accuracy at each threshold.

Table 1. Comparison of feature-based methods with and without LORF. For each feature two rows are reported: the first without LORF, the second with LORF. Bold numbers mark the higher value within the two rows for the same feature, indicating the gain brought by LORF.

Feature	Method	Accuracy	Recall	F1 Score	Precision
Permission	without LORF	0.9626	0.9677	0.9652	0.9603
	with LORF	0.9768	0.9832	0.9799	0.9686
API call	without LORF	0.9653	0.9605	0.9634	0.9657
	with LORF	0.9915	0.9862	0.9882	0.9902
Opcode	without LORF	0.9703	0.9655	0.9623	0.9698
	with LORF	0.9908	0.9864	0.9832	0.9897

5.2. Answer to RQ2: Identifying the Best Classifiers

Motivation: To achieve optimal detection accuracy, it is crucial to select the most effective classifier for each type of feature. Since various classifiers may excel with different features, our goal is to identify the machine learning algorithm that delivers the best detection performance. This approach ensures a tailored analysis for each feature type, enhancing the overall effectiveness of our malware detection system.

Methodology: During the classification phase, we utilize three types of features filtered by LORF. Choosing the right parameters for machine learning algorithms is crucial, as different parameter settings can significantly impact classification performance. Therefore, we must first identify the optimal parameters for each algorithm. The default settings provided by the Sklearn package often serve as a reasonable starting point.

For the RF, DT, and AdaBoost algorithms, which rely on tree ensemble methods, the depth of the tree is a key parameter that influences performance. As shown in Figure 4, we evaluate the accuracy of these algorithms with various parameter settings across differ-

ent features. Selecting appropriate depth values requires experimentation, so we test depths of 8, 16, 32, 64, 128, and 256 to determine which setting yields the best classification results.

In the case of the KNN algorithm, the parameter K indicates the number of nearest neighbors considered. We typically select odd values such as 1, 3, or 5 to minimize ties in classification, which often leads to improved outcomes. In total, we employ nine different machine learning algorithms to classify these features, with the aim of identifying the most effective algorithm for each specific feature type.

Results: The results are organized into two sections. First, we refer to Table 4, which presents the accuracy of three tree-based algorithms evaluated with various parameter settings. Next, we detail the accuracy achieved through 10-fold cross-validation across nine different classifiers, as illustrated in Figure 5.

Figure 4 illustrates the malware detection accuracy of the three tree-based algorithms across various parameter settings. The optimal parameters identified in this analysis were then employed to compare these algorithms with other machine learning methods. We utilized ten-fold cross-validation to assess three distinct features: permissions, API calls, and opcodes. Each classifier's accuracy was analyzed to determine the most effective model for each feature. For the API calls, our experiments revealed that the RF algorithm achieved impressive accuracy scores, with values ranging from 99.40% to 99.51% for parameter settings of 8 to 256. In the opcode feature analysis, RF also performed well, yielding a maximum score of 99.21% at a parameter setting of 32. Conversely, DT and AdaBoost showed slightly lower performances, with DT achieving a peak accuracy of 98.77% and AdaBoost reaching 99.21% for opcodes. Regarding permissions, RF excelled with scores reaching 99.06% at a parameter of 64, while DT and AdaBoost also demonstrated solid performance, with their best scores being 98.87% and 99.32%, respectively.

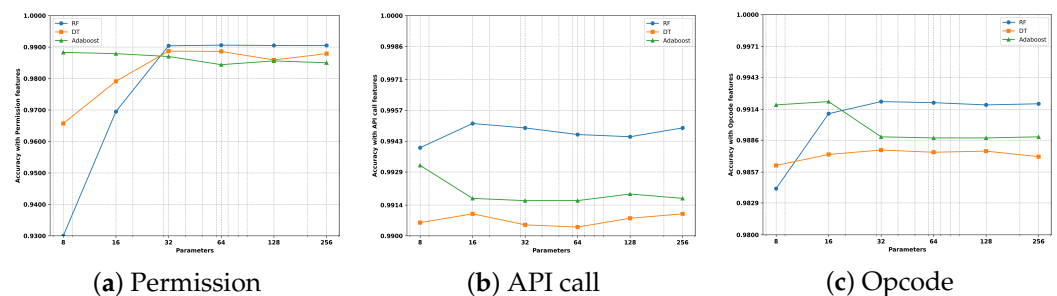


Figure 4. Accuracy of machine learning algorithms with different parameter settings for various features. Each panel corresponds to a specific feature type. The x-axis denotes the parameter settings (tree depth), and the y-axis shows the accuracy.

As illustrated in Figure 5, both RF and CatBoost demonstrated high accuracy across all features. In contrast, the performances of the 1NN, 2NN, and 3NN algorithms were relatively lower. Our experimental results indicate that RF is the most suitable classifier for permissions, LR is best for API calls, and CatBoost excels with opcodes. Consequently, we combine these three classifiers using a soft voting mechanism to enhance application detection based on these features. This integrated approach leverages the strengths of each classifier, ensuring a robust malware detection framework. Consequently, we utilize these three classifiers in conjunction with a soft voting mechanism to detect applications based on these features.

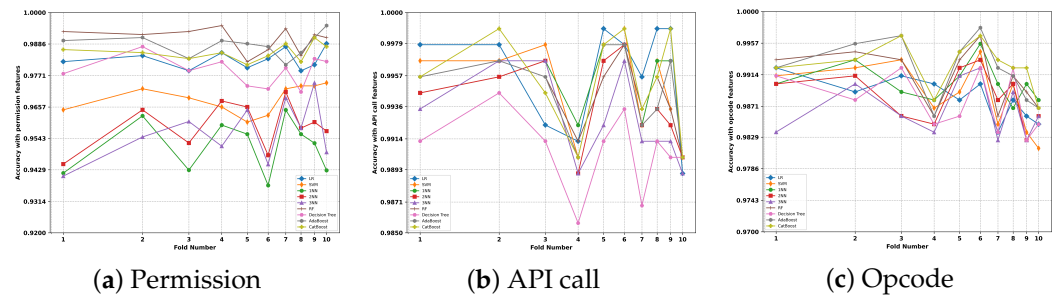


Figure 5. Accuracy of various machine learning methods across ten folds for different feature types. The x-axis denotes the fold index, and the y-axis indicates the accuracy.

5.3. Answer to RQ3: Determining Optimal Weights

Motivation: To accurately combine the three features and enhance the overall performance of our malware detection system, it is essential to determine the optimal weights for each classifier.

Methodology: In Section 5.2, we experimentally identify the optimal classifiers for the three features. However, the significance of each feature in malware detection varies. To address this, we utilize a soft voting mechanism to accurately combine the three features, employing a genetic algorithm to determine the optimal parameters and the best threshold T for each classifier. Finally, we compare our method with the hard voting mechanism.

Results: Figure 6 depicts the iterative process of the genetic algorithm in optimizing the accuracy of the model. The graph shows that the accuracy improves steadily from 99.52% at the first iteration to approximately 99.89% by the 21st iteration, as indicated by the red vertical line. The accuracy stabilizes and reaches its optimal performance at 99.89% after 31 iterations. This stabilization point marks the completion of the genetic algorithm's optimization process.

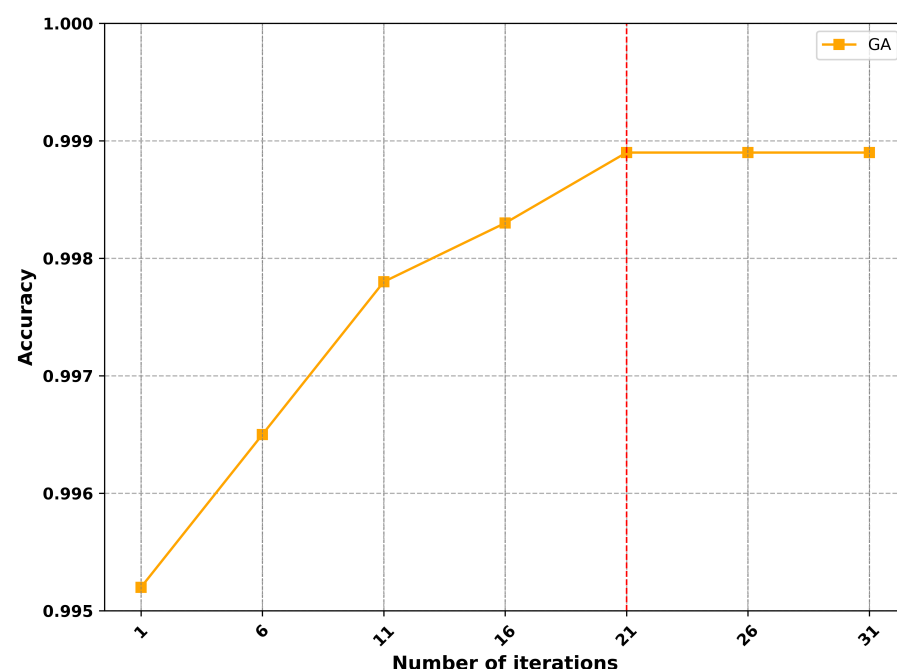


Figure 6. Iterative process of the genetic algorithm. The x-axis represents the number of iterations, while the y-axis indicates the achieved accuracy. The red vertical line marks the algorithm has stabilized and reached optimal performance.

5.4. Answer to RQ4: Evaluating Feature Combination Effectiveness

Motivation: To validate the effectiveness of combining different LORF features, we need to compare the performance of the combined method with individual classifiers and other voting mechanisms.

Methodology: Our dataset comprises 8340 benign samples from the Google Play Store and 5560 malicious samples from the Drebin dataset, providing a balanced representation of both benign and malicious Android applications. In this section, we compare the optimal classifier selected for each feature in Section 5.2 with the final combined soft voting method.

Results: Table 2 illustrates the comparative performance of LEMSOF using combined features versus individual features, as well as a comparison with the Hard Voting mechanism. LEMSOF, when utilizing all combined features, achieves high performance across all metrics, with an accuracy of 99.89%, recall of 99.62%, F1 score of 99.68%, and precision of 99.77%. This represents an improvement in accuracy of 0.87% over LEMSOF-per, 0.31% over LEMSOF-api, 0.62% over LEMSOF-opc, and 0.27% over the Hard Voting mechanism. These results underscore the effectiveness of the LEMSOF method in enhancing malware detection by leveraging a combination of features and optimized voting algorithms.

Table 2. Comparative performance of LEMSOF using combined and individual features. The results are categorized as follows: LEMSOF (using all features combined), LEMSOF-feature (using only individual features), and Hard Voting mechanism. Bold numbers indicate the highest value in each column.

Method	Accuracy	Recall	F1 Score	Precision
LEMSOF	0.9989	0.9962	0.9968	0.9977
LEMSOF-per	0.9902	0.9855	0.9863	0.9872
LEMSOF-api	0.9958	0.9973	0.9957	0.9913
LEMSOF-opc	0.9927	0.9910	0.9912	0.9929
Hard Voting	0.9962	0.9942	0.9951	0.9959

5.5. Answer to RQ5: Comparisons to Baselines

Motivation: To demonstrate the efficacy of LEMSOF, it is necessary to compare its performance against existing SOTA baseline methods.

Methodology: We compare LEMSOF against four SOTA baseline methods: FEDroid, HYDRA, MalScan, and Drebin.

FEDroid [45] leverages federated learning to enhance malware detection across multiple devices without centralizing the data, ensuring privacy and security while collaboratively training a shared model. Federated learning-based detection was tested on a diverse dataset, demonstrating its effectiveness in detecting malware while preserving user privacy.

HYDRA [46] combines multiple data modalities, such as static and dynamic features, using a deep learning framework to classify malware. It leverages hashing approaches to extract heterogeneous features from the API name and runtime parameters, which are then concatenated and fed into a deep learning model that aggregates multiple gated Convolutional Neural Networks (CNNs) [47] and bidirectional Long Short-Term Memory (LSTM) [48] networks.

MalScan [49] is a state-of-the-art Android malware detection method that treats the function call graphs of Android applications as a social network. It conducts social network-based centrality analysis to extract semantic features for detecting malware. MalScan focuses on rapid and scalable malware detection by analyzing market-wide mobile applications.

Drebin [14] is a classic Android malware detection method that conducts a broad analysis to extract a wide range of features from an APK, such as permissions, URLs, and intents. After embedding these features into a joint vector space, they trained an SVM-based model to detect malware.

Currently, there are many ways to solve this problem. We select a SOTA method FEDroid and a classic method Drebin. HYDRA is chosen because it represents a sophisticated deep learning approach that integrates multiple types of features. MalScan’s social network-based analysis offers a unique perspective on feature extraction and detection. Comparing LEMSOF with these diverse methods allows us to highlight its strengths and improvements over existing solutions.

Results: In this study, we employ a ten-fold cross-validation method, dividing samples into ten equivalent folds, with both malicious and benign applications in each fold. The four methods we compare used only a subset of the metrics employed in this study and different datasets, making it impractical to directly replicate their experimental results to validate our method’s effectiveness. Therefore, we meticulously reproduce these methods based on their research descriptions. We create training and test sets according to the evaluation methods used in their studies. Finally, classifiers are trained on the training set and predictions are made on the test set to evaluate the effectiveness of these methods.

As shown in Table 3, our proposed LEMSOF achieved the highest accuracy, recall, F1 score, and precision among all models in both validation and test results. Our method consistently achieves the highest F1 score on each sub-dataset. Specifically, LEMSOF’s average F1 score is 99.68%, which is 1.05%, 0.17%, 1.64%, and 1.61% higher than that of FEDroid, HYDRA, MalScan, and DREBIN, respectively. Additionally, its average accuracy is 99.89%, which is 0.77%, 0.14%, 1.77%, and 1.88% higher than that of FEDroid, HYDRA, MalScan, and Drebin, respectively. Additionally, our method also achieved the best recall, F1 score, and precision on these datasets. This superior performance can be attributed to our method’s combination of the LORF method for feature importance assessment with a genetic algorithm-optimized soft voting mechanism, effectively integrating the strengths of various features for comprehensive malware detection.

Table 3. Performance comparison between our method and the baseline methods on the Drebin dataset. Bold numbers indicate the highest value in each column.

Method	Accuracy	Recall	F1 Score	Precision
LEMSOF	0.9989	0.9962	0.9968	0.9977
FEDroid	0.9912	0.9855	0.9863	0.9902
HYDRA	0.9975	0.9951	0.9951	0.9952
MalScan	0.9812	0.9885	0.9804	0.9753
Drebin	0.9801	0.9819	0.9807	0.9796

The experimental dataset comprises 5560 malicious applications from the Drebin dataset [14] and 8340 benign applications collected from the Google Play Store. To further assess the robustness of our approach, we additionally employ the AndroZoo dataset [50], which contains 1000 malware samples and 1500 benign samples. Prior to analysis, we carry out several preprocessing steps, including deduplication and the extraction of discriminative static features that are essential for distinguishing benign from malicious applications. Notably, every Drebin sample has a unique hash value, reflecting the use of obfuscation techniques intended to thwart variant detection.

Each dataset is randomly partitioned into training and test sets with an 80/20 split. Table 4 compares the performance of our method with four baselines on the AndroZoo dataset. LEMSOF achieves the best results across all metrics, attaining 95.43% accuracy,

93.73% recall, 95.60% F1 score and 91.92% precision. The closest competitors, HYDRA and FEDroid, reach accuracies of 92.56% and 91.98%, respectively; HYDRA records 88.46% recall and a 90.03% F1 score, whereas FEDroid attains 92.21% recall and a 92.89% F1 score. MalScan and Drebin deliver substantially lower scores on all metrics.

Table 4. Performance comparison between our method and the baseline methods on the AndroZoo dataset. Bold numbers indicate the highest value in each column.

Method	Accuracy	Recall	F1 Score	Precision
LEMSOFT	0.9543	0.9373	0.9560	0.9192
FEDroid	0.9198	0.9221	0.9289	0.8884
HYDRA	0.9256	0.8846	0.9003	0.8872
MalScan	0.9163	0.8105	0.8541	0.8649
Drebin	0.8883	0.8334	0.8600	0.8708

These results highlight the superior effectiveness of LEMSOF in Android-malware detection, combining high precision with high recall. Moreover, the diversity of datasets used demonstrates the model’s ability to generalize across application categories and sources.

6. Performance Analysis

Our proposed method demonstrates superior performance due to several key factors: comprehensive feature extraction, optimized classifier integration, and dynamic weight adjustment.

LEMSOFT employs LORF to assess feature importance more accurately. By considering the frequency of features in both benign and malicious datasets, LORF effectively distinguishes between relevant and irrelevant features. This results in a high-quality feature set that enhances the overall detection accuracy. Using the LORF method, the accuracy increased by 1.42% for permissions, 2.26% for API calls, and 2.05% for opcodes. Additionally, LEMSOF integrates multiple machine learning classifiers using a soft voting mechanism. This approach leverages the strengths of various classifiers, such as SVM, RF, and KNN, each optimized for different feature types. Soft voting, which considers the predicted probabilities of each class, results in a more accurate detection system compared to hard voting. Moreover, the genetic algorithm used in LEMSOF dynamically adjusts the weights assigned to each classifier in the soft voting mechanism. Using the soft voting mechanism improves the F1 score by 1.05% for permissions, 0.11% for API calls, and 0.56% for opcodes compared to not using it and relying on a single feature. Since the F1 score has already exceeded 99%, further improvement is limited. After applying LORF, the F1 score exceeds 99%, indicating that there is limited room for further improvement. This ensures that the most effective classifiers have a greater influence on the final prediction. The iterative optimization process maximizes overall classification performance, allowing LEMSOF to adapt effectively to different datasets and malware types.

7. Threats to Validity

Internal threats to validity. One internal threat to the validity of our method is the potential bias in feature extraction due to the static nature of our analysis. Static analysis techniques, while useful, might not capture all the dynamic behaviors of malware, leading to incomplete feature sets. To mitigate this, we have incorporated LORF to enhance feature selection. However, this approach still relies on static features, which could limit its effectiveness. In future work, we aim to integrate dynamic analysis techniques to capture runtime behaviors, providing a more comprehensive feature set and improving the overall

accuracy and robustness of the detection system. Additionally, we plan to explore the use of advanced unpacking systems and machine learning algorithms from other studies to further enhance our feature extraction process.

External threats to validity. The generalizability of our results may be limited by the datasets used in our experiments. Although we utilize a diverse set of samples from the Drebin and AndroZoo datasets, these datasets may not encompass the full spectrum of malware and benign applications found in the wild. To address this external threat to validity, future research will focus on continuously updating and expanding our dataset to include the latest malware samples and benign applications. Additionally, as the Android operating system continues to evolve, there will be a need to periodically revisit and revise the feature set and classification strategies used by LEMSOFT to adapt to new system architectures and user behavior patterns.

8. Conclusions and Future Work

In this paper, we introduce LEMSOFT, an advanced Android malware detection system that leverages LORF and a soft voting mechanism optimized through genetic algorithms. Our approach addresses the limitations of existing methods by integrating feature importance assessment and classifier optimization. We demonstrate the effectiveness of our system through extensive experiments on a comprehensive dataset of Android applications, including both benign and malicious samples. Our results show that LEMSOFT significantly outperforms baseline methods, achieving an average accuracy of 99.89%. The innovative combination of tailored machine learning classifiers and the genetic algorithm-based soft voting mechanism ensures that our system can accurately identify a wide variety of malware types.

Future research could combine static and dynamic analysis techniques. By incorporating runtime behaviors and network traffic analysis, the detection framework could become more comprehensive and resilient against sophisticated obfuscation techniques.

Author Contributions: Conceptualization, Z.S. and Y.L.; methodology, Z.S.; software, Z.S.; validation, Q.H., Z.S. and Y.L.; formal analysis, Z.S.; investigation, Z.S.; resources, Q.H. and T.Z.; data curation, Z.S.; writing—original draft preparation, Z.S.; writing—review and editing, Y.L., Q.H. and T.Z.; visualization, Z.S.; supervision, Q.H. and T.Z.; project administration, Q.H. and T.Z.; funding acquisition, Q.H. and T.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ningxia Natural Science Foundation Project (Grant No. 2025AAC030079), the Macao Science and Technology Development Fund (FDCT) (Grant ID: 0161/2023/RIA3) and National Natural Science Foundation of China (Grant No. 61862001).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets used in this study are publicly available: the Drebin dataset can be accessed at <https://tianchi.aliyun.com/dataset/172774/>; accessed on 1 October 2025 benign applications were collected from the Google Play Store; and the AndroZoo dataset is available upon request from the AndroZoo project (<https://androzoo.uni.lu/>) accessed on 1 October 2025.

Acknowledgments: The authors would like to thank their respective institutions for the support provided during the preparation of this study.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

AdaBoost	Adaptive Boosting
API	Application Programming Interface
APK	Android Application Package
CatBoost	Category Boosting
CNN	Convolutional Neural Network
DEX	Dalvik Executable
DT	Decision Tree
FCG	Function Call Graph
FNN	Feedforward Neural Network
GA	Genetic Algorithm
KNN	k-Nearest Neighbors
LEMSOFT	Leveraging Extraction Method and Soft Voting
LORF	Lexical Occurrence Ratio-Based Filtering
LR	Logistic Regression
LSTM	Long Short-Term Memory
RF	Random Forest
SOTA	State-of-the-Art
SVM	Support Vector Machine

References

- Li, X.; Liu, L.; Liu, Y.; Zhao, Y.; Zhang, P.; Liu, H. Multimodal Fusion for Android Malware Detection Based on Large Pre-trained Models. *IEEE Trans. Softw. Eng.* **2025**, *51*, 1569–1590. [\[CrossRef\]](#)
- Yadav, P.; Menon, N.; Ravi, V.; Vishvanathan, S.; Pham, T.D. EfficientNet Convolutional Neural Networks-Based Android Malware Detection. *Comput. Secur.* **2022**, *115*, 102622. [\[CrossRef\]](#)
- Saracino, A.; Sgandurra, D.; Dini, G.; Martinelli, F. MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Trans. Dependable Secur. Comput.* **2018**, *15*, 83–97. [\[CrossRef\]](#)
- Cai, H.; Meng, N.; Ryder, B.; Yao, D. DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling. *IEEE Trans. Inf. Forensics Secur.* **2019**, *14*, 1455–1470. [\[CrossRef\]](#)
- Arora, A.; Peddoju, S.K.; Conti, M. PermPair: Android Malware Detection Using Permission Pairs. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 1968–1982. [\[CrossRef\]](#)
- Qiu, J.; Han, Q.L.; Luo, W.; Pan, L.; Nepal, S.; Zhang, J.; Xiang, Y. Cyber Code Intelligence for Android Malware Detection. *IEEE Trans. Cybern.* **2023**, *53*, 617–627. [\[CrossRef\]](#)
- Xu, J.; Li, Y.; Deng, R.H.; Xu, K. SDAC: A Slow-Aging Solution for Android Malware Detection Using Semantic Distance Based API Clustering. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 1149–1163. [\[CrossRef\]](#)
- Peynirci, G.; Eminağaoğlu, M.; Karabulut, K. Feature Selection for Malware Detection on the Android Platform Based on Differences of IDF Values. *J. Comput. Sci. Technol.* **2020**, *35*, 946–962. [\[CrossRef\]](#)
- Wu, Y.; Li, M.; Zeng, Q.; Yang, T.; Wang, J.; Fang, Z.; Cheng, L. DroidRL: Feature selection for android malware detection with reinforcement learning. *Comput. Secur.* **2023**, *128*, 103126. [\[CrossRef\]](#)
- Tarwireyi, P.; Terzoli, A.; Adigun, M. Using Multi-Audio Feature Fusion for Android Malware Detection. *Comput. Secur.* **2023**, *131*, 103282. [\[CrossRef\]](#)
- Cui, L.; Hao, Z.; Jiao, Y.; Fei, H.; Yun, X. VulDetector: Detecting Vulnerabilities Using Weighted Feature Graph Comparison. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 2004–2017. [\[CrossRef\]](#)
- Niño-Adan, I.; Manjarres, D.; Landa-Torres, I.; Portillo, E. Feature weighting methods: A review. *Expert Syst. Appl.* **2021**, *184*, 115424. [\[CrossRef\]](#)
- Xia, Y.; Chen, K.; Yang, Y. Multi-label classification with weighted classifier selection and stacked ensemble. *Inf. Sci.* **2021**, *557*, 421–442. [\[CrossRef\]](#)
- Arp, D.; Spreitzerbarth, M.; Hubner, M.; Gascon, H.; Rieck, K. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2014.
- Taha, A.; Barukab, O.; Malebary, S. Fuzzy Integral-Based Multi-Classifiers Ensemble for Android Malware Classification. *Mathematics* **2021**, *9*, 2880. [\[CrossRef\]](#)

16. Shen, L.; Fang, M.; Xu, J. GHGDroid: Global heterogeneous graph-based android malware detection. *Comput. Secur.* **2024**, *141*, 103846. [\[CrossRef\]](#)
17. Liu, Z.; Zhang, L.F.; Tang, Y. Enhancing Malware Detection for Android Apps: Detecting Fine-Granularity Malicious Components. In Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, 11–15 September 2023; pp. 1212–1224. [\[CrossRef\]](#)
18. Alabrah, A. A Novel Neural Network Architecture Using Automated Correlated Feature Layer to Detect Android Malware Applications. *Mathematics* **2023**, *11*, 4242. [\[CrossRef\]](#)
19. Onwuzurike, L.; Mariconti, E.; Andriotis, P.; Cristofaro, E.D.; Ross, G.; Stringhini, G. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version). *ACM Trans. Priv. Secur.* **2019**, *22*, 1–34. [\[CrossRef\]](#)
20. Idrees, F.; Rajarajan, M.; Conti, M.; Chen, T.M.; Rahulamathavan, Y. PIndroid: A novel Android malware detection system using ensemble learning methods. *Comput. Secur.* **2017**, *68*, 36–46. [\[CrossRef\]](#)
21. Aboaoja, F.A.; Zainal, A.; Ali, A.M.; Ghaleb, F.A.; Alsolami, F.J.; Rassam, M.A. Dynamic Extraction of Initial Behavior for Evasive Malware Detection. *Mathematics* **2023**, *11*, 416. [\[CrossRef\]](#)
22. Cui, Y.; Sun, Y.; Lin, Z. DroidHook: A novel API-hook based Android malware dynamic analysis sandbox. *Autom. Softw. Eng.* **2023**, *30*, 10. [\[CrossRef\]](#)
23. Casado-Vara, R.; Severt, M.; Díaz-Longueira, A.; Rey, Á.; Calvo-Rolle, J.L. Dynamic Malware Mitigation Strategies for IoT Networks: A Mathematical Epidemiology Approach. *Mathematics* **2024**, *12*, 250. [\[CrossRef\]](#)
24. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. DL-Droid: Deep learning based android malware detection using real devices. *Comput. Secur.* **2020**, *89*, 101663. [\[CrossRef\]](#)
25. Melvin, A.A.R.; Kathrine, G.J.W.; Ilango, S.S.; Vimal, S.; Rho, S.; Xiong, N.N.; Nam, Y. Dynamic Malware Attack Dataset Leveraging Virtual Machine Monitor Audit Data for the Detection of Intrusions in Cloud. *Trans. Emerg. Telecommun. Technol.* **2022**, *33*, 4287. [\[CrossRef\]](#)
26. Li, C.; Cheng, Z.; Zhu, H.; Wang, L.; Lv, Q.; Wang, Y.; Li, N.; Sun, D. DMalNet: Dynamic Malware Analysis Based on API Feature Engineering and Graph Learning. *Comput. Secur.* **2022**, *122*, 102872. [\[CrossRef\]](#)
27. Li, C.; Lv, Q.; Li, N.; Wang, Y.; Sun, D.; Qiao, Y. A Novel Deep Framework for Dynamic Malware Detection Based on API Sequence Intrinsic Features. *Comput. Secur.* **2022**, *116*, 102686. [\[CrossRef\]](#)
28. Zhu, H.J.; Wang, L.M.; Zhong, S.; Li, Y.; Sheng, V.S. A Hybrid Deep Network Framework for Android Malware Detection. *IEEE Trans. Knowl. Data Eng.* **2022**, *34*, 5558–5570. [\[CrossRef\]](#)
29. Zhang, Y.; Sui, Y.; Pan, S.; Zheng, Z.; Ning, B.; Tsang, I.; Zhou, W. Familial Clustering for Weakly-Labeled Android Malware Using Hybrid Representation Learning. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 3401–3414. [\[CrossRef\]](#)
30. da Costa, F.H.; Medeiros, I.; Menezes, T.; da Silva, J.V.; da Silva, I.L.; Bonifácio, R.; Narasimhan, K.; Ribeiro, M. Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification. *J. Syst. Softw.* **2022**, *183*, 111092. [\[CrossRef\]](#)
31. Faghihi, F.; Zulkernine, M.; Ding, S. CamoDroid: An Android application analysis environment resilient against sandbox evasion. *J. Syst. Archit.* **2022**, *125*, 102452. [\[CrossRef\]](#)
32. Surendran, R.; Thomas, T.; Emmanuel, S. A TAN based hybrid model for android malware detection. *J. Inf. Secur. Appl.* **2020**, *54*, 102483. [\[CrossRef\]](#)
33. Wang, H.; Zhang, W.; He, H. You are what the permissions told me! Android malware detection based on hybrid tactics. *J. Inf. Secur. Appl.* **2022**, *66*, 103159. [\[CrossRef\]](#)
34. Tong, F.; Yan, Z. A hybrid approach of mobile malware detection in Android. *J. Parallel Distrib. Comput.* **2017**, *103*, 22–31. [\[CrossRef\]](#)
35. Bhat, P.; Dutta, K. A multi-tiered feature selection model for android malware detection based on Feature discrimination and Information Gain. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 9464–9477. [\[CrossRef\]](#)
36. Shar, L.K.; Demissie, B.F.; Ceccato, M.; Tun, Y.N.; Lo, D.; Jiang, L.; Bienert, C. Experimental Comparison of Features and Classifiers for Android Malware Detection. In Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems, Seoul, Republic of Korea, 13–15 July 2020; pp. 50–60.
37. Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*; Complex Adaptive Systems; MIT Press: Cambridge, MA, USA, 1992. Available online: <https://direct.mit.edu/books/monograph/2574> (accessed on 1 October 2025).
38. Cortes, C.; Vapnik, V. Support-Vector Networks. *Mach. Learn.* **1995**, *20*, 273–297. [\[CrossRef\]](#)
39. Breiman, L. Random Forests. *Mach. Learn.* **2001**, *45*, 5–32. [\[CrossRef\]](#)
40. Cover, T.; Hart, P. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* **1967**, *13*, 21–27. [\[CrossRef\]](#)
41. Cox, D.R. The Regression Analysis of Binary Sequences. *J. R. Stat. Soc. Ser. B (Methodol.)* **2018**, *20*, 215–232. [\[CrossRef\]](#)
42. Quinlan, J.R. Induction of Decision Trees. *Mach. Learn.* **1986**, *1*, 81–106. [\[CrossRef\]](#)

43. Prokhorenkova, L.; Gusev, G.; Vorobev, A.; Dorogush, A.V.; Gulin, A. CatBoost: Unbiased boosting with categorical features. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, Montreal, QC, Canada, 3–8 December 2018; pp. 6639–6649.
44. Freund, Y.; Schapire, R.E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. Syst. Sci.* **1997**, *55*, 119–139. [[CrossRef](#)]
45. Fang, W.; He, J.; Li, W.; Lan, X.; Chen, Y.; Li, T.; Huang, J.; Zhang, L. Comprehensive Android Malware Detection Based on Federated Learning Architecture. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 3977–3990. [[CrossRef](#)]
46. Gibert, D.; Mateu, C.; Planes, J. HYDRA: A multimodal deep learning framework for malware classification. *Comput. Secur.* **2020**, *95*, 101873. [[CrossRef](#)]
47. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [[CrossRef](#)]
48. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [[CrossRef](#)] [[PubMed](#)]
49. Wu, Y.; Li, X.; Zou, D.; Yang, W.; Zhang, X.; Jin, H. MalScan: Fast Market-Wide Mobile Malware Scanning by Social-Network Centrality Analysis. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 11–15 November 2019; pp. 139–150. [[CrossRef](#)]
50. Hurier, M.; Suarez-Tangil, G.; Dash, S.K.; Bissyandé, T.F.; Le Traon, Y.; Klein, J.; Cavallaro, L. Euphony: Harmonious Unification of Cacophonous Anti-Virus Vendor Labels for Android Malware. In Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, Argentina, 20–21 May 2017; pp. 425–435. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.