Journal of Soft Computing Exploration
Homepage: htttps://shmpublisher.com/index.php/joscex
p-ISSN: 2746-7686
e-ISSN: 2746-0991

.

How few features are enough? A systematic study of feature reduction in system call N-Grams for android malware detection

Rajif Agung Yunmar[1], Andika Setiawan[2]
[1,2]Department of Informatics Engineering, Insititut Teknologi Sumatera, Indonesia.

Article Info

Article history:

Received May 17, 2026
Revised June 24, 2026
Accepted July 5, 2026

Keywords:

Android malware
System call
N-gram
Feature Selection
Contextual richness
Feature sparsity

ABSTRACT

System call N-gram representations have been widely utilized in Android malware detection due to their ability to capture behavioral execution patterns. However, previous studies have not systematically examined how aggressive feature reduction affects different N-gram orders, particularly in extremely low-dimensional feature spaces. This study evaluates the impact of feature reduction on system call N-gram representations ranging from 1-gram to 4-gram using Chi-Square feature selection across subsets varying from Top-3 to Top-100 features. Malware classification was performed using the XGBoost algorithm with leakage-free stratified 10-fold cross-validation. Experimental results show that the 2-gram representation achieved the highest accuracy of 96.98% under full-feature conditions. Surprisingly, under aggressive feature reduction, the 1-gram representation consistently outperformed higher-order N-grams across all reduced feature subsets, indicating greater robustness in low-dimensional feature spaces. Statistical significance analysis using paired t-test and Wilcoxon signed-rank test further confirmed significant performance differences between 1-gram and higher-order N-gram representations. Although higher-order N-grams provide richer contextual and more discriminative behavioral representations, their information becomes increasingly fragmented in sparse feature spaces, reducing effectiveness under extreme dimensionality reduction. Furthermore, aggressive feature selection substantially improved computational efficiency, reducing the execution time of the 4-gram representation from 3000.78 seconds to 8.28 seconds under the Top-100 scenario. Overall, the findings reveal a trade-off between contextual richness, feature sparsity, detection performance, and computational efficiency in system call N-gram-based Android malware detection.

Corresponding Author:

Rajif Agung Yunmar,
Department of Informatics Engineering,
Institut Teknologi Sumatera,
Jl. Terusan Ryacudu, Way Huwi, Kec. Jati Agung, Kabupaten Lampung Selatan, Lampung, Indonesia.
Email: rajif@if.itera.ac.id
https://doi.org/10.52465/joscex.v7i2.120

1. INTRODUCTION

Malware poses a significant cybersecurity threat to Android users, leading to unauthorized access to personal information, financial theft, and privacy violations. The increasing use of Android devices for communication and transactions has made them attractive targets for cybercriminals. Kaspersky reports that

cyberattacks on Android smartphones rose by 29 percent in 2025 compared to the previous year [1]. This highlights the evolving nature of Android malware threats and the need for effective detection strategies.

Detection method generally falls into two categories: signature-based and behavior-based detection. Signature-based detection identifies malware using known malicious patterns and is computationally efficient, but requires continuous signature database updates and remains ineffective against zero-day or obfuscated malware [2]. In contrast, behavior-based detection analyzes suspicious application behaviors during analysis or execution, enabling more effective detection of previously unseen or obfuscated malware variants [3].

Recent review studies suggest that behavior-based Android malware detection generally relies on static, dynamic, or hybrid analysis, each using different feature representations and learning strategies [4]. Static analysis extracts features such as permissions, API calls, metadata, opcode sequences, and graph structures without execution [5], [6], but remains vulnerable to obfuscation [7]. In contrast, dynamic analysis monitors runtime behaviors, including system calls, memory interactions, and network activities, providing more realistic behavioral evidence [8], [9]. Hybrid analysis combines both approaches to improve robustness at the cost of higher computational complexity.

System calls have received significant attention in Android malware detection due to their direct representation of interactions between applications and the operating system kernel [10]. All application activities, including malicious ones, depend on system calls to access resources like files, memory, processes, and network communications. Consequently, system call traces can effectively illustrate malware behavior.

Malware activities identified through system calls often exhibit specific sequential patterns [11]. For example, accessing a file typically involves system calls like `open()`, `fstat()`, `read()`, and `close()`. The sequential traits of these calls make them well-suited for the N-gram approach [12], which divides a sequence into smaller contiguous subsequences of N consecutive elements. This method effectively captures local behavioral patterns and dependencies among system calls, aiding in the detection of repetitive or suspicious behaviors linked to malicious activities. By representing system call traces as N-gram sequences, machine learning algorithms can better distinguish between benign and malicious behaviors.

Low-order N-grams, particularly 1-grams, are computationally efficient and widely used in Android malware detection [13], [14]. However, because they only model individual system call frequencies, they cannot capture sequential dependencies, limiting their ability to represent complex malware execution behaviors. To address this limitation, higher-order N-grams have been introduced to capture richer local execution contexts. Prior studies have evaluated 2-gram to 6-gram representations and reported improved behavioral expressiveness [15], [16]. However, increasing N-gram order substantially enlarges feature dimensionality, producing sparse feature spaces with higher computational cost.

Feature selection is commonly applied to reduce N-gram dimensionality while preserving discriminative behavioral patterns. Previous studies have explored various feature selection techniques for N-gram-based malware detection [17]-[21], yet many do not clearly specify feature subset sizes or selection thresholds, limiting reproducibility and obscuring the impact of aggressive dimensionality reduction.

Xu et al. [22] demonstrated that selecting the top twenty N-gram features improved efficiency while maintaining performance. Similarly, Deepa et al. [23] showed that 1-gram feature subsets from ten to one hundred features reduced computational overhead with acceptable accuracy. However, these studies either did not systematically evaluate aggressive reduction across multiple subset sizes or were limited to 1-gram, leaving the robustness of higher-order N-grams under extreme reduction insufficiently explored.

The effectiveness of malware detection also depends on the learning algorithm used to model sparse N-gram features. Recent studies have employed both traditional machine learning and deep learning methods. Traditional models such as Random Forest, SVM, and gradient boosting remain popular due to interpretability and computational efficiency, whereas deep learning methods can automatically learn complex feature representations but typically require larger datasets and higher computational resources [24]. A study by Jyothish et al. [25] reported that XGBoost, outperformed both conventional machine learning and deep learning approaches for system call-based Android malware detection, motivating its use in this study.

Despite these advances, three important research gaps remain. First, prior studies have not systematically evaluated aggressive feature reduction across multiple N-gram orders. Second, the trade-off between contextual richness and feature sparsity remains insufficiently quantified. Third, limited work has examined this problem under leakage-free evaluation settings with statistical significance analysis.

Therefore, a systematic investigation is essential to gain a deeper understanding of the behavior of various N-gram orders under different sizes of feature subsets, particularly in extreme low-dimensional feature spaces. The main contributions of this study are summarized as follows: This study presents a systematic

leakage-free evaluation of aggressive Chi-Square-based feature reduction on system call N-gram representations (1-gram to 4-gram) for Android malware detection, covering feature subset scenarios ranging from extremely low-dimensional to moderately sized spaces. This study provides quantitative, behavioral, and statistical analyses of the trade-off between contextual richness, feature sparsity, and discriminative capability across different N-gram orders, including significance validation using paired statistical tests. This study identifies practical feature subset configurations for lightweight Android malware detection systems, demonstrating how reduced feature spaces can maintain competitive detection performance while substantially improving computational efficiency.

2. RESEARCH METHODOLOGY

This section outlines the systematic approach used to assess the impact of aggressive feature reduction on system call N-gram representations for Android malware detection. The methodology includes dataset preparation, dynamic system call extraction, N-gram feature construction, feature selection, classification, and performance evaluation. Figure 1 presents the overall research workflow, while Figure 4 details the leakage-free classification pipeline. The experimental results are analyzed to investigate trade-offs related to contextual richness, feature sparsity, detection performance, and computational efficiency across different N-gram orders and levels of feature reduction.

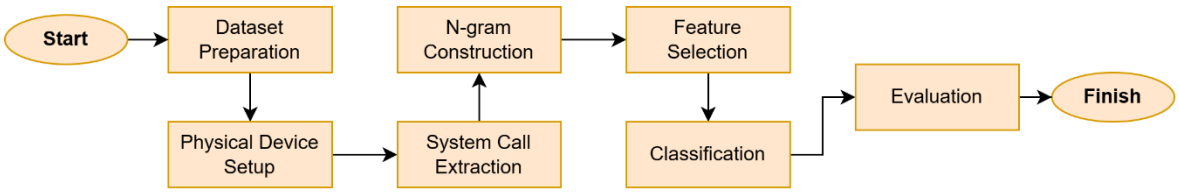

Figure 1. Overview of the proposed research methodology.

Dataset

The dataset used in this study is sourced from two main repositories: MalDroid2020 [26] and CICAndMal2017 [27]. A specific subset of 3,279 applications was selected, comprising 1,545 malware and 1,734 benign. The selection criteria focused on compatibility with API level 24 and successful execution on the physical device, the Samsung SM-J330. It includes various threats in the contemporary mobile malware landscape, such as ransomware, adware, riskware, SMS malware, trojans, and spyware. Table 1 presents the distribution of malware samples across these malware families, providing diverse behavioral characteristics for evaluating system call-based Android malware detection.

Table 1. Distribution of malware samples across malware families.

Type of Malware	Total Samples
SMS Malware	437
Trojan	319
Spyware	260
Riskware	244
Adware	212
Ransomware	73
Total	1,545

Physical Device Setup

In this research, a Samsung J330 physical device with API level 24 was employed as the primary execution environment to ensure the authenticity and reliability of the collected system call traces. The choice of a physical device over an emulator was driven by its ability to produce more realistic runtime behaviors and system call activities, as contemporary Android malware often employs anti-emulation and anti-analysis techniques that can hinder proper observations in virtualized environments [28], [29]. To further enhance the execution environment's realism, several configurations were established to simulate a real world scenario, including the insertion of a SIM card, population of call logs, setup of SMS messages, and enabling internet connectivity via Wi-Fi. This approach aims to elicit more realistic application behaviors, allowing malware samples to engage in their malicious activities naturally, thus ensuring that the collected system call traces more accurately reflect the actual behaviors of Android malware [30], [31].

System Call Extraction Using Dynamic Analysis

The feature extraction process outlined in this study is based on dynamic analysis, where system call sequences are generated during application execution on a physical Android device. The operating system records all application interactions as system call traces, facilitating the analysis of malicious behaviors exhibited by Android malware through these logs. This extraction process includes three key phases: preparation of the execution environment, application execution, and post-execution cleanup. The first phase involves configuring the physical device environment to simulate a real-world usage scenario.

During the application execution phase, each application is installed and prepared to remain on the home screen without distractions. Subsequently, the Monkey tool [32] is used to generate 2,000 randomized user interaction events, such as touch and swipe operations, simulating user activities and encouraging diverse application behaviors. In the final phase, the execution is terminated, and the generated system call logs are retained for analysis. The application is then uninstalled to maintain a clean environment on the device for subsequent samples. The workflow of dynamic system call extraction is illustrated in Figure 2.

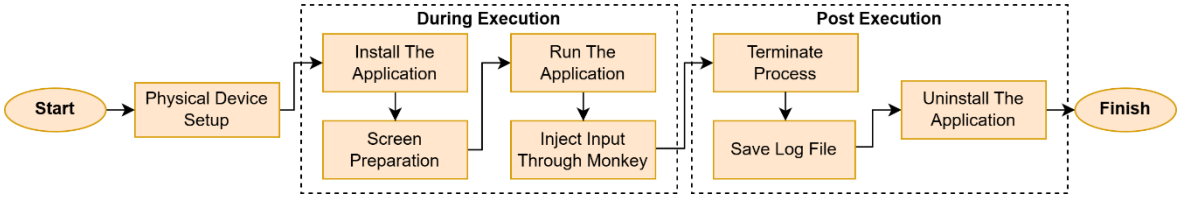

Figure 2. The workflow of dynamic feature extraction on physical Android devices.

N-Gram Construction

The system call logs obtained during dynamic analysis reveal a sequential nature, as shown in Figure 3, capturing the chronological interactions between an application and the Android kernel. To transform these continuous traces into a machine learning-friendly format, this study utilizes the N-gram modeling technique. N-grams effectively capture local contextual dependencies within the system call sequence, which are crucial for identifying behavioral patterns typical of malicious software.

```
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
connect(sin_port=htons(443), sin_addr=inet_addr("172.217.24.170")) = -1
fcntl64(3, F_SETFL, O_RDWR|O_NONBLOCK)  = 0
poll([{fd=3, events=POLLOUT}], 1, 100)  = 1 ([{fd=3, revents=POLLOUT}])
sendto("GET / HTTP/1.1\r\nHost: example.com\r\n...", 156, 0, NULL, 0) = 156
recvfrom(3, "HTTP/1.1 200 OK\r\n...", 4096, 0, NULL, NULL) = 850
close(3) = 0
```

Figure 3. Example of system call trace captured during app execution.

Formally, let $S = \{s_1, s_2, s_3, \dots, s_n\}$ represent a sequence of system calls extracted from an app's execution log. An N-gram is defined as a contiguous subsequence composed of n items derived from the specified sequence S. Mathematically, an N-gram g_i starting at position i can be represented Equation (1).

$$g_i = (s_i, s_{i+1}, \dots, s_{i+n-1}) \tag{1}$$

Here, n signifies the order of the N-gram (for example, $n = 1$ corresponds to 1-grams, and $n = 2$ pertains to bigrams). The total number of N-grams generated from a sequence of length L is $L - n + 1$.

Once the N-grams are constructed, the subsequent step entails quantifying the occurrence of each unique N-gram in order to build a numerical feature vector. This process, commonly referred to as term frequency, assesses the significance of a specific sequence pattern within the execution trace of the application. Mathematically, let $G = \{g_1, g_2, \dots, g_m\}$ denote the set of all unique N-grams identified throughout the entire training dataset, which serves as the feature vocabulary. For a specific application A, the frequency of a unique N-gram g_j is computed by Equation (2).

$$f(g_j, A) = \sum_{i=1}^{L-n+1} \delta(g_j, \text{ngram}_i) \tag{2}$$

The function $\delta\left(g_j, \text{ngram}_i\right)$ serves as an indicator function that operates at each position i within the sliding window. It returns a value of 1 if the N-gram at that specific position corresponds to the target pattern

g_j, and a value of 0 otherwise. Furthermore, the term $f\left(g_j, A\right)$ denotes the resulting frequency value, which may exceed 1 significantly, contingent upon the frequency with which the pattern is reiterated during the execution process.

Feature Selection with Chi-Square

The N-gram construction process generates numerous features, particularly when utilizing higher-order N-grams such as 3-grams and 4-grams. As the order of N-grams increases, the feature space grows and becomes sparse due to the multitude of unique system call combinations. This results in high dimensionality, which raises computational complexity and may introduce redundant features, consequently affecting malware detection performance. To mitigate this issue, feature selection techniques are employed to reduce dimensionality while preserving crucial behavioral patterns from system call sequences.

In this study, the Chi-Square test is employed as a statistical method to evaluate the independence between a specific N-gram feature and the occurrence of designated classes, namely Malware or Benign. A high Chi-Square statistic indicates a significant dependence, implying that the feature contains valuable discriminative information for classification purposes.

In such sparse higher-order N-gram feature spaces, certain rare features may become constant within specific training subsets, causing the Chi-Square (χ^2) computation to produce undefined or not-a-number (`NaN`) values. If not properly handled, such non-informative features may distort feature ranking and adversely affect Top-K selection, particularly for small feature subsets. To address this issue, features with `NaN` or non-positive ($\chi^2 \leq 0$) scores are excluded prior to ranking, ensuring that only statistically valid and discriminative features are considered during feature selection.

After filtering invalid features, all remaining N-gram features are systematically ranked according to their Chi-Square scores. A comprehensive analysis of various feature subset sizes is performed, including lower-dimensional scenarios such as the top-3, top-5, and top-10 features, in addition to more moderate and larger subsets such as the top-20, top-30, top-50, top-75, and top-100 features.

Classification Model

This study uses the Extreme Gradient Boosting (XGBoost) classifier to evaluate the discriminative capability of selected N-gram features. XGBoost is a tree-based ensemble algorithm ideal for high-dimensional and sparse feature spaces, like system call N-grams, where feature importance is often uneven and interactions are non-linear. Previous studies have demonstrated the effectiveness of XGBoost for Android malware detection. Jyothish et al. reported that XGBoost outperformed conventional machine learning models, including Random Forest and SVM as well as several deep learning approaches when using system call N-gram features [25]. Based on these findings, XGBoost was selected as a robust baseline classifier. To ensure experimental consistency and reproducibility, a fixed set of hyperparameters was used across all experiments, covering tree complexity, learning rate, sampling ratios, and execution settings. The complete hyperparameter configuration is summarized in Table 2.

Table 2. XGBoost hyperparameter settings used in the experiments.

Hyperparameter	Value
Number of Trees (n_estimators)	100
Maximum Tree Depth (max_depth)	6
Learning Rate	0.3
Subsample Ratio	0.8
Column Sampling Ratio	0.8
Evaluation Metric	Multiclass Log Loss
Random Seed	42
Parallel Threads	All available CPU cores

To provide a clearer illustration of the end-to-end classification workflow, Figure 4 presents the detailed model pipeline used in this study. System call traces are first transformed into multiple N-gram representations ranging from 1-gram to 4-gram. The generated features are then evaluated using stratified 10-fold cross-validation. To explicitly prevent data leakage, Chi-Square feature ranking and Top-K feature selection are performed exclusively within each training fold rather than prior to cross-validation on the full dataset. The resulting feature subset is subsequently applied to both the corresponding training and test folds. The selected training features are then used to train the XGBoost classifier, while the transformed test fold is used solely for prediction and evaluation. Finally, performance is assessed using classification metrics and statistical significance testing.

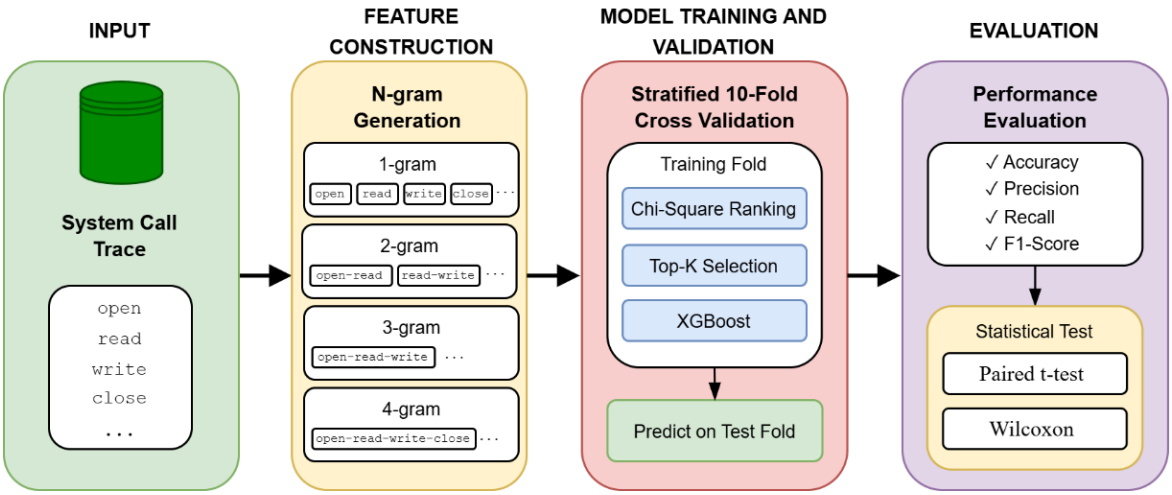

Figure 4. Detailed workflow of feature construction, model training, and evaluation.

Evaluation Scheme and Metrics

To assess the effectiveness and robustness of the proposed Android malware detection model, this study employs stratified 10-fold cross-validation. This method is used XGBoost to provide reliable performance estimations and reduce evaluation bias from specific data partitioning. In the 10-fold scheme, the dataset is divided into ten equal subsets. In each iteration, nine subsets are used for training the classification model, while one serves as the testing set. This process is repeated ten times, with each subset acting as the testing set once. The final performance results are obtained by averaging the evaluation scores across all folds.

The proposed malware detection model is evaluated using metrics such as accuracy, precision, recall, and F1-score. Accuracy reflects the overall proportion of correctly classified applications and provides a general view of performance, calculated as shown in Equation (3). In this case, TP represents True Positive, TN denotes True Negative, FP indicates False Positive, and FN stands for False Negative.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3}$$

Relying solely on accuracy is insufficient for malware detection, as false positives and false negatives have distinct security implications. Precision measures the proportion of classified malware that is genuinely malicious, and high precision is essential to reduce false positives, which can wrongly label benign applications as malware, negatively impacting user experience. This is calculated using Equation (4). Recall, or detection rate, measures the proportion of actual malware samples detected by the model. This metric is critical in cybersecurity, as undetected malware poses significant security risks. Recall is computed using Equation (5).

$$Precision = \frac{TP}{TP+FP} \tag{4}$$

$$Recall = \frac{TP}{TP+FN} \tag{5}$$

To reconcile the trade-off between precision and recall, this study also utilizes the F1-score, which represents the harmonic mean of both metrics. The F1-score is especially valuable in the evaluation of malware detection systems, as it provides a more balanced assessment of classification effectiveness across varying feature selection and N-gram representation scenarios. It is computed according to Equation (6).

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision+Recall} \tag{6}$$

To further validate whether performance differences among N-gram representations are statistically significant, this study employs both parametric and non-parametric statistical significance tests, namely the paired t-test and the Wilcoxon signed-rank test. The paired t-test is used to determine whether the mean performance differences between two paired experimental conditions differ significantly under the assumption of approximately normally distributed paired differences. Since cross-validation results may not always satisfy normality assumptions, the Wilcoxon signed-rank test is additionally employed as a non-parametric alternative

that compares paired performance distributions based on rank differences. The combined use of these two statistical methods provides a more rigorous and robust evaluation of performance differences across N-gram representations under various feature selection scenarios.

3. RESULTS AND DISCUSSIONS

This section presents the experimental results of Android malware detection under aggressive feature reduction. The goal is to identify the minimum behavioral information threshold, represented by Chi-Square ranked N-grams, necessary for acceptable classification standards. By analyzing feature subsets from 100 to 3 features, the study aims to determine how much behavioral noise can be eliminated without losing the signal of malicious intent. The XGBoost classifier serves as a benchmark to assess the information density and discriminative power of these constrained feature sets.

Baseline Performance and Statistical Analysis Under Full Features

The classification results in Table 3 and Figure 5 indicate detection performance varies with different N-gram orders. The 2-gram model demonstrated the highest overall performance, achieving an accuracy of 96.98 percent, precision of 96.99 percent, recall of 96.98 percent, and F1-score of 96.98 percent. These results suggest that 2-gram representations effectively balance contextual behavioral representation and feature compactness. By capturing relationships between adjacent system calls, 2-gram features preserve essential execution patterns linked to malicious activities while maintaining a manageable feature space.

Table 3. XGBoost classification performance with full features.

N-Gram	Num. of Features	Accuracy	Precision	Recall	F1-Score
1-Gram	160	94.82	94.81	94.82	94.81
2-Gram	3,065	96.98	96.99	96.98	96.98
3-Gram	18,879	96.77	96.77	96.77	96.77
4-Gram	76,330	96.95	96.95	96.95	96.95

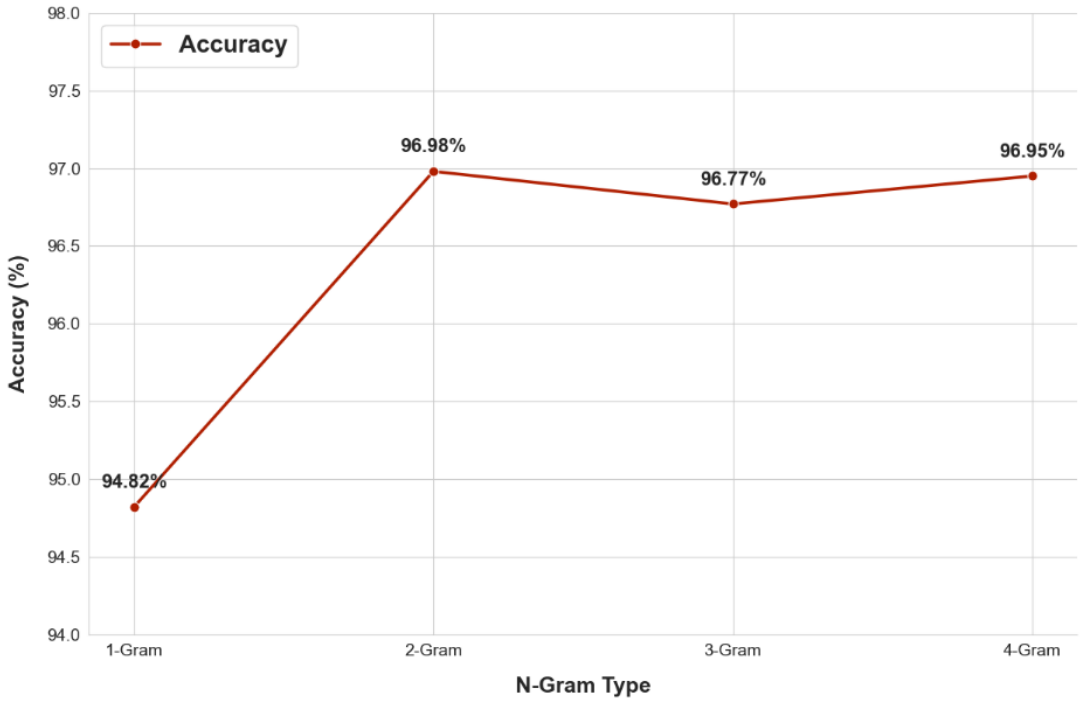

Figure 5. Comparison of classification performance metrics across N-gram orders.

In contrast, the 1-gram representation attained the lowest overall performance, registering an accuracy of 94.82 percent. Although 1-gram features exhibit computational efficiency and possess the smallest feature dimensionality, they are limited in their capacity to represent sequential dependencies among system calls. Consequently, significant behavioral contexts related to malware execution may not be adequately captured when examining only individual system call frequencies.

Interestingly, the performance of higher-order N-gram representations, specifically the 3-gram and 4-gram models, did not surpass that of the 2-gram model, despite their ability to encode more complex contextual information. The 3-gram model achieved 96.77 percent accuracy, while the 4-gram model reached 96.95 percent. This suggests that an increase in N-gram order does not necessarily lead to improved detection performance. Although higher-order N-grams provide more detailed execution contexts, they significantly increase feature dimensionality and sparsity. Table 3 illustrates that the number of generated features increased dramatically from 3,065 in the 2-gram representation to 76,330 in the 4-gram representation. Such high-dimensional sparse feature spaces may introduce redundant and infrequently occurring behavioral patterns, thereby reducing the discriminative effectiveness of the features.

The experimental results indicate that moderate contextual representation through 2-gram features achieves the best balance between behavioral richness and feature sparsity. This suggests that high-order N-gram representations may lead to diminishing returns because of dimensional explosion and sparse feature distribution, even with a robust classifier like XGBoost.

Table 4 confirms that the performance differences between 1-gram and all higher-order N-gram representations are statistically significant according to both the paired t-test and Wilcoxon signed-rank test ($p < 0.05$), indicating the weaker discriminative capability of 1-gram features under full-feature conditions. While 2-gram significantly outperforms 3-gram despite a small accuracy difference (0.21%), no significant differences are observed between 2-gram and 4-gram or between 3-gram and 4-gram ($p > 0.05$). These findings suggest that 2-gram provides the most effective balance between contextual richness and feature dimensionality, whereas higher-order N-grams offer diminishing performance gains.

Table 4. Paired t-test and Wilcoxon signed-rank test results for inter-order N-gram accuracy comparisons.

Comparison	Accuration Diff. (%)	Paired t-Test (p-Value)	Wilcoxon (p-Value)	Significance ($\alpha<0.05$)
1-Gram vs 2-Gram	2.16	0.00015	0.00260	Yes
1-Gram vs 3-Gram	1.95	0.00028	0.00506	Yes
1-Gram vs 4-Gram	2.13	0.00032	0.00506	Yes
2-Gram vs 3-Gram	0.21	0.03058	0.02181	Yes
2-Gram vs 4-Gram	0.03	0.64213	0.59363	No
3-Gram vs 4-Gram	0.18	0.34484	0.33289	No

Table 5. Comparative accuracy performance across different N-gram orders under various feature subset.

Feature Subset	1-Gram	2-Gram	3-Gram	4-Gram
Top-3	80.02	65.60	63.95	67.13
Top-5	81.52	70.02	69.29	72.46
Top-10	87.62	75.21	75.39	78.32
Top-20	90.88	83.65	81.91	83.16
Top-30	92.53	87.53	83.11	84.42
Top-50	93.93	90.82	84.48	85.51
Top-75	93.72	92.01	87.16	86.98
Top-100	94.08	92.65	90.33	88.19
Full Features	94.82	96.98	96.77	96.95

Performance Comparison of N-Gram Orders Under Feature Reduction

The leakage-free experimental results presented in Table 5 and Figure 6 reveal distinct performance patterns across different N-gram orders under varying feature subset sizes. Under full-feature conditions, the 2-gram representation achieved the highest classification accuracy of 96.98%, confirming that moderate sequential contextual information provides the most effective behavioral representation for Android malware detection.

However, a different trend emerged under aggressive feature reduction. Across all reduced feature subsets, from Top-3 to Top-100, the 1-gram representation consistently outperformed higher-order N-grams, demonstrating superior robustness in low-dimensional feature spaces. Even with the addition of the Top-75 scenario, this pattern remained unchanged. The 1-gram model improved from 80.02% accuracy in the Top-3 scenario to 94.08% in Top-100, approaching its full-feature performance of 94.82%.

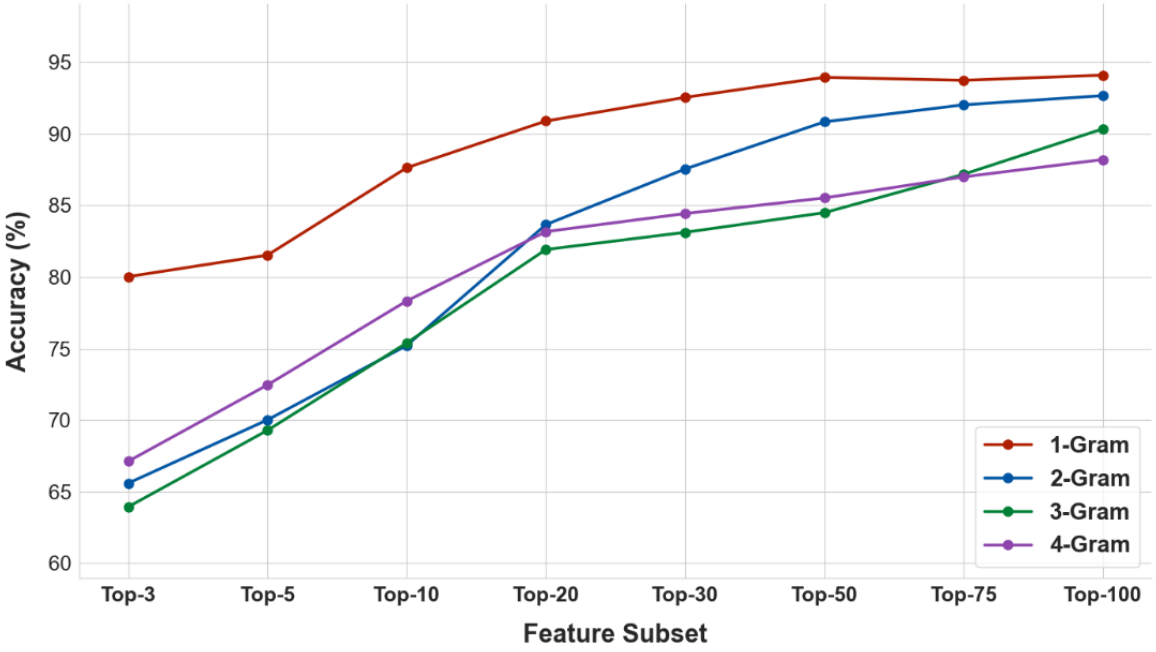

Figure 6. Comparative accuracy across different N-gram orders under aggressive feature reduction.

This behavior is largely explained by feature-space characteristics. 1-gram representations produce compact and dense feature distributions based on system call frequencies, whereas higher-order N-grams generate much larger and sparser feature spaces. As a result, discriminative information in 2-gram, 3-gram, and 4-gram representations becomes fragmented across many unique feature combinations, making them more vulnerable to aggressive feature reduction. This effect is particularly evident in the Top-3 and Top-5 scenarios, where higher-order N-grams exhibit substantial performance degradation.

Nevertheless, the performance gap narrows as more features are retained. The recovery observed in 3-gram and 4-gram representations between Top-75 and Top-100 suggests that larger feature subsets help restore fragmented contextual information. Overall, these findings highlight a trade-off between contextual richness and feature sparsity: 2-gram offers the best performance when sufficient features are available, whereas 1-gram remains more robust under severe dimensional constraints.

Computational Efficiency Analysis

Table 6 shows the percentage of feature reduction for each subset scenario compared to the full feature set, while Table 7 details the execution times for different N-gram representations. The results indicate that significant feature reduction enhances computational efficiency across all N-gram orders. Additionally, there is a notable relationship between computational overhead and feature dimensionality, especially in higher-order N-gram representations utilizing the complete feature set.

Table 6. Percentage of feature space reduction across N-gram orders and top-K scenarios.

N-Gram	Top-3 (%)	Top-5 (%)	Top-10 (%)	Top-20 (%)	Top-30 (%)	Top-50 (%)	Top-75 (%)	Top-100 (%)	Num. of Features
1-Gram	98.13	96.88	93.75	87.50	81.25	68.75	53.13	37.50	160
2-Gram	99.90	99.84	99.67	99.35	99.02	98.37	97.55	96.74	3,065
3-Gram	99.98	99.97	99.95	99.89	99.84	99.74	99.60	99.47	18,879
4-Gram	99.996	99.993	99.987	99.974	99.961	99.934	99.90	99.87	76,330

Table 7. Computational cost of N-gram representations across feature reduction scenarios (in seconds).

N-Gram	Top-3	Top-5	Top-10	Top-20	Top-30	Top-50	Top-75	Top-100	Full
1-Gram	0.35	0.89	0.74	1.18	0.56	0.71	0.82	1.55	11.95
2-Gram	0.46	0.47	0.51	0.5	1.21	0.7	0.85	1.09	165.05
3-Gram	2.43	2.04	2.19	3.81	3.36	2.21	2.31	4.02	684.96
4-Gram	9.49	7.81	8.00	7.99	8.08	8.14	9.05	8.28	3000.78

As shown in Table 6, higher-order N-gram representations exhibit substantially greater reductions in feature dimensionality compared to lower-order representations. For example, the 4-gram representation initially contained 76,330 features and achieved a 99.90 percent reduction under the Top-75 configuration and 99.87 percent under Top-100. Similarly, the 3-gram and 2-gram representations achieved reductions of 99.60 percent and 97.55 percent in the Top-75 scenario, respectively. In contrast, the 1-gram representation originally consisted of only 160 features, resulting in a relatively smaller reduction of 53.13 percent for Top-75 and 37.50 percent for Top-100. These results confirm that increasing the N-gram order leads to substantial feature-space expansion and higher sparsity, thereby making feature reduction increasingly important for computational tractability.

All computational efficiency experiments were conducted on a machine with an Intel Core i5-3320M processor (2.60 GHz), 16 GB RAM, and 1 TB SSD running Windows 10 Pro. For full-feature experiments, execution time includes only XGBoost training and prediction. In Top-K scenarios, execution time additionally includes the feature selection process, comprising Chi-Square computation, feature ranking, and feature subset extraction. Dataset loading, preprocessing, cross-validation split generation, statistical testing, and result export were excluded to ensure fair comparison across scenarios.

The impact of dimensionality reduction on computational efficiency is evident in Table 7. Under reduced feature subsets, execution times remained relatively low across all evaluated N-gram orders, even for high-order representations. For instance, the 4-gram representation required only 9.05 seconds under the Top-75 configuration and 8.28 seconds under Top-100, despite originating from more than 76,000 features. Similar trends were observed for the 3-gram model, which required 2.31 seconds for Top-75 and 4.02 seconds for Top-100. In contrast, substantial computational overhead emerged under full-feature conditions. The 1-gram representation required 11.95 seconds for classification, whereas the 2-gram, 3-gram, and 4-gram representations required significantly longer execution times of 165.05 seconds, 684.96 seconds, and 3000.78 seconds, respectively. These results demonstrate that high-dimensional feature spaces substantially increase computational cost during model training and inference.

When comparing classification performance in Table 5 with execution time in Table 7, a clear trade-off between detection effectiveness and computational efficiency becomes evident. Higher-order N-gram representations generally achieved strong classification performance under full-feature conditions but incurred substantially higher computational costs. For example, the 4-gram representation achieved 96.95 percent accuracy but required 3000.78 seconds for execution, whereas the 2-gram representation achieved a slightly higher accuracy of 96.98 percent with a significantly lower execution time of 165.05 seconds. Under reduced-feature settings, the Top-75 scenario provides an attractive balance between performance and efficiency. In particular, the 2-gram representation retained 92.01 percent accuracy while reducing execution time to only 0.85 seconds. These findings highlight that aggressive feature reduction substantially improves scalability and practical deployment feasibility, especially for high-dimensional N-gram representations, while preserving acceptable detection performance in lightweight malware detection systems.

Behavioral Characteristics of Dominant N-Gram Features

To facilitate behavioral interpretation, dominant system call features were identified using global Chi-Square ranking on the complete dataset, rather than the fold-wise ranking used during classification. This ranking was used solely for post-hoc analysis to identify globally discriminative behavioral patterns. The selected dominant system calls were then categorized into functional groups based on Linux system call semantics using the Linux system call references [33] and the Linux manual pages [34]. Each system call was assigned to a primary behavioral category, including file access, memory management, process control, resource management, and protection mechanisms, according to its dominant operational role within the operating system.

The Top-3 dominant features in each N-gram representation, as shown in Table 8, illustrate the increased contextual richness associated with higher N-gram orders. In 1-gram representations, the main

features are individual system calls like `setrlimit`, `fchmod`, and `timerfd`. These individual calls offer limited behavioral context as they stand alone, and although they may indicate suspicious activities pertaining to resource management and protection mechanisms, they fail to capture the execution dependencies among system calls.

Higher-order N-gram representations capture complex execution relationships. For instance, dominant 2-gram features such as readlinkat write and read setpriority indicate local behavioral transitions related to file manipulation and process control. Such patterns may reflect malware behaviors involving file inspection, privilege adjustment, stealth scheduling, or persistence maintenance, which are commonly observed in spyware and trojan-like malware. Additionally, dominant 3-gram and 4-gram features reveal longer execution workflows involving memory management, file access, protection mechanisms, and process control. These results highlight the ability of higher-order N-grams to represent richer behavioral contexts relevant to Android malware activities.

Table 8. Dominant system call N-gram features and their behavioral characteristics.

N-Gram	Rank	Dominant N-Gram Feature	χ^2 Score	Functional Group
1-Gram	1	setrlimit	98.20	Information Maintenance
	2	fchmod	80.35	Protection
	3	timerfd	46.90	Information Maintenance
2-Gram	1	setrlimit clock_gettime	212.17	Information Maintenance
	2	readlinkat write	207.65	File Management
	3	read setpriority	201.12	File Management & Process Control
3-Gram	1	prctl clock_gettime faccessat	260.43	Process Control & File Management
	2	fstat64 mmap2 gettimeofday	258.12	File Management & Info. Maintenance
	3	timerfd_settime close fcntl64	244.78	Info. Maintenance & File Management
4-Gram	1	mmap2 madvise madvise fstat64	288.54	File Management
	2	mprotect mprotect faccessat openat	277.12	Protection & File Management
	3	mmap2 fstat64 mmap2 gettimeofday	265.40	File Management & Info. Maintenance

Interestingly, the Chi-Square scores of dominant features increase with higher N-gram orders, suggesting that longer system call sequences demonstrate stronger discriminative capability and richer contextual behavioral information. However, this increased contextual richness also leads to sparse and fragmented feature distributions, causing important behavioral patterns to be distributed across many unique execution sequences. This observation is consistent with the classification results presented in previous sections, where higher-order N-grams showed strong performance under full-feature conditions but degraded more noticeably under aggressive feature reduction. In contrast, 1-gram representations remain dense and compact, preserving essential behavioral information even in restricted feature subset scenarios.

4. CONCLUSION

This study systematically investigated the impact of aggressive feature reduction on system call N-gram representations for Android malware detection using Chi-Square feature selection and XGBoost classification. Experimental results showed that under full-feature conditions, the 2-gram representation achieved the highest detection accuracy of 96.98%, indicating that moderate sequential dependencies between adjacent system calls provide valuable behavioral information. However, under aggressive feature reduction scenarios ranging from Top-3 to Top-100 features, 1-gram representations consistently outperformed higher-order N-grams, demonstrating greater robustness in highly constrained feature spaces due to their denser and more compact feature distributions. Statistical significance analysis using paired t-test and Wilcoxon signed-rank test further confirmed that the performance differences between 1-gram and higher-order N-gram representations were statistically significant. In contrast, although higher-order N-grams provide richer contextual information and higher statistical relevance, their discriminative patterns become increasingly fragmented in sparse high-dimensional spaces, making them more sensitive to extreme feature reduction. Furthermore, aggressive feature selection substantially improved computational efficiency, reducing execution time from 3000.78 seconds for full-feature 4-gram representations to only 8.28 seconds under the Top-100 scenario. These findings highlight the trade-off between contextual richness, feature sparsity, detection performance, and computational efficiency, while demonstrating the feasibility of lightweight system call-based Android malware detection using highly reduced feature subsets.

REFERENCES

[1] Kaspersky, "Kaspersky report: Attacks on smartphones increased in the first half of 2025." Accessed: May 15, 2026. [Online]. Available: https://www.kaspersky.com/about/press-releases/kaspersky-report-attacks-on-smartphones-increased-in-the-first-half-of-2025

[2] H. Haidros Rahima Manzil and S. Manohar Naik, "Detection approaches for android malware: Taxonomy and review analysis," *Expert Syst. Appl.*, vol. 238, p. 122255, Mar. 2024, doi: 10.1016/j.eswa.2023.122255.

[3] "What is Heuristic Analysis?," What is Heuristic Analysis? Accessed: Jan. 29, 2021. [Online]. Available: https://usa.kaspersky.com/resource-center/definitions/heuristic-analysis

[4] S. Jain, A. Arora, and D. Kumar, "Hybrid analysis in android malware detection: A systematic and comparative review," *Comput. Sci. Rev.*, vol. 62, p. 100980, Nov. 2026, doi: 10.1016/j.cosrev.2026.100980.

[5] A. Museeb, Y. Hamed, Y. Baashar, A. M. J. Kanaan-Jebna, A. A. Hamza, and R. Sokkalingam, "Android Malware Detection Using API Calls and Permissions With Random Forest Classifier," *IEEE Access*, vol. 14, pp. 6464–6480, 2026, doi: 10.1109/ACCESS.2026.3651861.

[6] W. Wang *et al.*, "Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019, doi: 10.1109/ACCESS.2019.2918139.

[7] A. Razgallah, R. Khoury, S. Hallé, and K. Khanmohammadi, "A survey of malware detection in Android apps: Recommendations and perspectives for future research," *Comput. Sci. Rev.*, vol. 39, p. 100358, 2021, doi: 10.1016/j.cosrev.2020.100358.

[8] A. Davarasan, J. Samual, K. Palansundram, and A. Ali, "A Comprehensive Review of Machine Learning Approaches for Android Malware Detection," *J. Cyber Secur. Risk Audit.*, pp. 39–60, 2024, doi: 10.63180/jcsra.thestap.2024.1.5.

[9] P. Bhat, S. Behal, and K. Dutta, "A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning," *Comput. Secur.*, vol. 130, p. 103277, 2023, doi: 10.1016/j.cose.2023.103277.

[10] R. Casolare, C. De Dominicis, G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, "Dynamic Mobile Malware Detection through System Call-based Image representation," *J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl.*, vol. 12, no. 1, pp. 44–63, 2021, doi: 10.22667/JOWUA.2021.03.31.044.

[11] G. Canfora, F. Mercaldo, E. Medvet, and C. A. Visaggio, "Detecting Android malware using sequences of system calls," *3rd Int. Workshop Softw. Dev. Lifecycle Mob. DeMobile 2015 - Proc.*, pp. 13–20, 2015, doi: 10.1145/2804345.2804349.

[12] T. Islam, S. S. M. M. Rahman, Md. A. Hasan, A. S. Md. M. Rahaman, and Md. I. Jabiullah, "Evaluation of N-Gram Based Multi-Layer Approach to Detect Malware in Android," *Procedia Comput. Sci.*, vol. 171, 2020, doi: 10.1016/j.procs.2020.04.115.

[13] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of Android Malware Detection Based on System Calls," in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, New Orleans Louisiana USA: ACM, Mar. 2016, pp. 1–8. doi: 10.1145/2875475.2875487.

[14] A. Singh, M. Tanha, Y. Girdhar, and A. Hunter, "Interpretable Android Malware Detection Based on Dynamic Analysis," in *10th International Conference on Information Systems Security and Privacy*, 2024, pp. 195–202. doi: 10.5220/0012415800003648.

[15] X. Zhang *et al.*, "An Early Detection of Android Malware Using System Calls based Machine Learning Model," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, Vienna Austria: ACM, Aug. 2022, pp. 1–9. doi: 10.1145/3538969.3544413.

[16] Z. Mas'ud, S. Sahib, M. F. Abdollah, S. R. Selamat, and Y. Robiah, "An Evaluation of N-gram System Call Sequence in Mobile Malware Detection," *ARPN J. Eng. Appl. Sci.*, vol. 11, no. 5, 2016.

[17] P. Brown, A. Brown, M. Gupta, and M. Abdelsalam, "Online Malware Classification with System-Wide System Calls in Cloud IaaS," in *2022 IEEE 23rd International Conference on Information Reuse and Integration for Data Science (IRI)*, San Diego, CA, USA: IEEE, 2022. doi: 10.1109/IRI54793.2022.00042.

[18] M. Z. Mas'ud, S. Sahib, Siti Rahayu Selamat, and C. Y. Huoy, "A Comparative Study on Feature Selection Method for N-gram Mobile Malware Detection," *Int. J. Netw. Secur.*, vol. 19, no. 5, Sep. 2017, doi: 10.6633/IJNS.201709.19(5).10.

[19] R. Thangaveloo, W. W. Jing, C. K. Leng, and J. Abdullah, "DATDroid: Dynamic analysis technique in android malware detection," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 10, no. 2, pp. 536–541, 2020, doi: 10.18517/ijaseit.10.2.10238.

[20] A. Ananya, A. Aswathy, T. R. Amal, P. G. Swathy, P. Vinod, and S. Mohammad, "SysDroid: a dynamic ML-based android malware analyzer using system call traces," *Clust. Comput.*, vol. 23, no. 4, pp. 2789–2808, 2020, doi: 10.1007/s10586-019-03045-6.

[21] N. A. B. M. Ariff, M. Z. Mas'ud, N. Bahaman, E. Hamid, and N. A. Anuar, "Ensemble Method for Mobile Malware Detection using N-Gram Sequences of System Calls," *Int. J. Commun. Netw. Inf. Secur. IJCNIS*, vol. 13, no. 2, 2022, doi: 10.17762/ijcnis.v13i2.4937.

[22] L. Xu, D. Zhang, M. A. Alvarez, J. A. Morales, X. Ma, and J. Cavazos, "Dynamic Android Malware Classification Using Graph-Based Representations," *Proc. - 3rd IEEE Int. Conf. Cyber Secur. Cloud Comput. CSCloud 2016 2nd IEEE Int. Conf. Scalable Smart Cloud SSC 2016*, pp. 220–231, 2016, doi: 10.1109/CSCloud.2016.27.

[23] K. Deepa, G. Radhamani, P. Vinod, M. Shojafar, N. Kumar, and M. Conti, "Identification of Android malware using refined system calls," *Concurr. Comput. Pract. Exp.*, vol. 31, no. 20, pp. 1–24, 2019, doi: 10.1002/cpe.5311.

[24] X. Chang, J. Chen, S. Cai, R. Xia, and S. Wang, "A Systematic Survey of Android Malware Detection Through Learning-Based Technology," 2025, *SSRN*. doi: 10.2139/ssrn.5519478.

[25] A. Jyothish, A. Mathew, and P. Vinod, "Effectiveness of machine learning based android malware detectors against adversarial attacks," *Clust. Comput.*, vol. 27, no. 3, pp. 2549–2569, Jun. 2024, doi: 10.1007/s10586-023-04086-8.

[26] S. Mahdavifar, D. Alhadidi, and Ali. A. Ghorbani, "Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder," *J. Netw. Syst. Manag.*, vol. 30, no. 1, p. 22, Jan. 2022, doi: 10.1007/s10922-021-09634-4.

[27] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible android malware detection and family classification using network-flows and API-calls," in *Proceedings - International Carnahan Conference on Security Technology*, 2019. doi: 10.1109/CCST.2019.8888430.

[28] V. Sihag, M. Vardhan, and P. Singh, "A survey of android application and malware hardening," *Comput. Sci. Rev.*, vol. 39, p. 100365, 2021, doi: 10.1016/j.cosrev.2021.100365.

.

[29] P. Faruki, R. Bhan, V. Jain, S. Bhatia, and N. El Madhoun, "A Survey and Evaluation of Android-Based Malware Evasion Techniques and Detection Frameworks," *J. Inf.*, vol. 14, no. 7, pp. 1–38, 2023, doi: https://doi.org/10.3390/info14070374.
[30] J. Song, "Hunting potential C2 commands in Android malware via Smali string comparison and control flow analysis," Virusbulletin, Berlin, 2025.
[31] D. R. Arikkat *et al.*, "Network Traffic-Based Malware Detection for Android Devices," in *The 4th Italian Conference on Big Data and Data Science,* N. Bena, Ed., Turin, Italy: CEUR Workshop Proceedings, 2025.
[32] AndroidDev, "Monkey." Accessed: Jan. 12, 2023. [Online]. Available: https://developer.android.com/studio/test/monkey?hl=id
[33] Geeksforgeeks, "Linux system call in Detail," Linux system call in Detail. Accessed: Jun. 22, 2026. [Online]. Available: https://www.geeksforgeeks.org/linux-unix/linux-system-call-in-detail/
[34] Linux Manual Page, "syscalls(2) — Linux manual page," syscalls(2) — Linux manual page. Accessed: Jun. 22, 2026. [Online]. Available: https://man7.org/linux/man-pages/man2/syscalls.2.html