Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.047530

ARTICLE

Outsmarting Android Malware with Cutting-Edge Feature Engineering and Machine Learning Techniques

Ahsan Wajahat[1], Jingsha He[1], Nafei Zhu[1], Tariq Mahmood[2,3], Tanzila Saba[2], Amjad Rehman Khan[2] and Faten S. Alamri[4,*]

[1]Faculty of Information Technology, Beijing University of Technology, Beijing, 100124, China

[2]Artificial Intelligence and Data Analytics (AIDA) Lab, CCIS Prince Sultan University, Riyadh, 11586, Saudi Arabia

[3]Faculty of Information Sciences, University of Education, Vehari Campus, Vehari, 61100, Pakistan

[4]Department of Mathematical Sciences, College of Science, Princess Nourah bint Abdulrahman University, Riyadh, 84428, Saudi Arabia

*Corresponding Author: Faten S. Alamri. Email: fsalamri@pnu.edu.sa

Received: 08 November 2023 Accepted: 11 February 2024 Published: 25 April 2024

ABSTRACT

The growing usage of Android smartphones has led to a significant rise in incidents of Android malware and privacy breaches. This escalating security concern necessitates the development of advanced technologies capable of automatically detecting and mitigating malicious activities in Android applications (apps). Such technologies are crucial for safeguarding user data and maintaining the integrity of mobile devices in an increasingly digital world. Current methods employed to detect sensitive data leaks in Android apps are hampered by two major limitations they require substantial computational resources and are prone to a high frequency of false positives. This means that while attempting to identify security breaches, these methods often consume considerable processing power and mistakenly flag benign activities as malicious, leading to inefficiencies and reduced reliability in malware detection. The proposed approach includes a data preprocessing step that removes duplicate samples, manages unbalanced datasets, corrects inconsistencies, and imputes missing values to ensure data accuracy. The Minimax method is then used to normalize numerical data, followed by feature vector extraction using the Gain ratio and Chi-squared test to identify and extract the most significant characteristics using an appropriate prediction model. This study focuses on extracting a subset of attributes best suited for the task and recommending a predictive model based on domain expert opinion. The proposed method is evaluated using Drebin and TUANDROMD datasets containing 15,036 and 4,464 benign and malicious samples, respectively. The empirical result shows that the Random Forest (RF) and Support Vector Machine (SVC) classifiers achieved impressive accuracy rates of 98.9% and 98.8%, respectively, in detecting unknown Android malware. A sensitivity analysis experiment was also carried out on all three ML-based classifiers based on MAE, MSE, R^2, and sensitivity parameters, resulting in a flawless performance for both datasets. This approach has substantial potential for real-world applications and can serve as a valuable tool for preventing the spread of Android malware and enhancing mobile device security.

KEYWORDS

Android malware detection; machine learning; SVC; K-Nearest Neighbors (KNN); RF

1 Introduction

Mobile apps have become essential tools for a wide range of smartphone functions such as making payments, purchasing tickets, and capturing photos. As of now, more than five million apps are accessible on both Android and iPhone platforms. Among these, Android is the predominant operating system (OS), commanding approximately 74% of the global market share. This positions Android as the most widely used OS for mobile devices worldwide [1]. Mobile apps can use various sensors on the device, such as GPS, camera, and microphone, as well as access personal data stored on the device, such as pictures, messages, and contacts [2]. However, unauthorized access to end-user data is a widespread problem, as various studies have shown [3,4]. Additionally, unauthorized access to end-user data has been a persistent issue. Even malicious apps removed from the Google Play Store can bypass security measures and reappear, putting users' personal information at risk [5]. It is essential to distinguish between legitimate and malicious apps to protect end-user data. Google Play Store can bypass security measures and reappear, putting users' personal information at risk. It is essential to distinguish between legitimate and malicious apps to protect end-user data.

Addressing these challenges the research community has proposed various solutions to mitigate the risks associated with malicious apps [6]. Two broad techniques for malware analysis are static analysis and dynamic analysis. Static analysis involves scrutinizing source code and binaries to identify unusual patterns [7]. Dynamic analysis entails running the software under in a controlled environment and monitoring its behavior and actions [8]. Machine learning (ML) algorithms provide a more proactive and efficient approach to detecting and combating Android malware as they continually learn and adapt to new and evolving threats. These algorithms can swiftly detect and classify malware by identifying patterns and anomalies in large datasets such as system calls, network traffic, API permissions, intents, and user behavior. ML-based models improve their accuracy over time, using supervised and unsupervised learning techniques, making them more effective at identifying new and previously unknown malware types. This proactive approach enables early detection and faster response to new threats, providing better protection against Android malware. Additionally, the ML-based approach can automate the malware detection and removal process, reducing the burden on security professionals and increasing the overall security system's efficiency. The primary goal of this study is to establish a robust framework utilizing ML-based algorithms, namely Support Vector Machine (SVC), Random Forest (RF), and K-Nearest Neighbors (KNN), for the identification and classification of Android malware. To accomplish this, the research team gathered two separate datasets containing benign and malicious Android apps, including Drebin and TUANDROMD, to evaluate the best algorithm for detecting Android malware accurately. These datasets cover various malware types and families, enabling the team to train and test their proposed models on a representative sample of the Android malware landscape. The study also implemented several feature selection techniques, such as Gain-ratio feature selection and chi-squared tests, to identify the most relevant features for malware detection. These features represent apps characteristics such as permissions, API calls, and intent. The proposed algorithms were then trained and tested using the selected features and evaluated using well-known metrics such as accuracy, precision, recall, F1-score, and AUC. The results of the experiments will provide valuable insights into the performance of these algorithms for Android malware detection, contributing to the development of more effective and efficient security solutions for the Android platform. Moreover, this research will serve as a benchmark for future studies in this area, providing a basis for comparison and further improvement.

1.1 Deficiencies in Existing Approaches

The increasing use of Android smartphones has led to a rise in Android malware and privacy breaches. This highlights the need for more effective detection technologies. Although current methodologies are advancing in scope, they still face significant limitations, such as high computational costs and a high rate of false positives, which hinder their practical application. There is a notable gap in the realm of data preprocessing for Android malware detection. Current techniques often fail to address important issues such as inconsistent data, missing values, and imbalanced datasets. These issues directly affect the reliability and accuracy of ML-based models. Additionally, the field has not fully utilized advanced feature selection and extraction methods, such as Gain-ratio feature selection and chi-squared tests. These methods are crucial in improving classifier performance by identifying key risk variables. In this domain, there is a significant opportunity to innovate by applying advanced ML-based techniques. However, these techniques have not been extensively utilized. The research aims to bridge these gaps by proposing a comprehensive framework. The framework includes novel data preprocessing techniques, sophisticated feature selection and extraction methods, and the deployment of robust ML algorithms. This study aims to improve Android malware detection and enhance mobile device security by evaluating it against established benchmarks. The research addresses critical gaps in the field.

1.2 Motivation and Novel Contribution

The main contribution of said investigation is illustrated as under:

- The study introduces innovative data preprocessing techniques to manage inconsistencies, address missing values, reduce noise, address imbalanced data issues, and discretize malware datasets, enhancing data quality and reliability for analysis.
- The study suggests utilizing Gain-ratio feature selection and extraction methods to improve classifier performance by eliminating deterministic risk variables, a significant step towards optimizing feature selection for improved analytical outcomes.
- This study focuses on creating a robust ML-based system for efficient Android malware prediction, utilizing advanced techniques to prompt user responses and reduce the harmful effects of malware, a crucial aspect of the research.
- The proposed model undergoes a thorough evaluation, benchmarking it against current models, providing a comprehensive assessment of its efficacy and competitiveness in the field using established benchmarks.
- The study conducts a comprehensive sensitivity analysis to understand the model's resilience under various conditions, thereby enhancing its performance and reliability.

The following is the paper's structure: Section 2 provides an overview of the related literature. Section 3 provides the data preparation and feature engineering for the cutting-edge algorithms used in this study. Section 4 provides the background about the cutting-edge algorithms used in this study. Section 5 highlights the findings of the study. Section 6 provides an interpretation and evaluation of the results. Finally, Section 7 concludes the study by summarizing its key findings, highlighting its contributions, and offering recommendations for future work.

2 Related Work

This section discusses the use of ML-based techniques for detecting Android malware, focusing on static, dynamic, and hybrid analysis methods. Each method has its limitations, with static analysis

being less resource-intensive but more susceptible to obfuscation, dynamic analysis being more time-consuming but more accurate, and hybrid analysis combining both methods for comprehensive analysis. Understanding these limitations is crucial for improving ML-based malware detection effectiveness.

2.1 Static Analysis Using Cutting-Edge Algorithm

Recent studies have utilized ML techniques to detect malware in Android apps. DroidSieve uses static analysis to detect obfuscated malware by extracting n-gram features from the app's code and applying a decision tree algorithm. Singh et al. use an ensemble of convolutional neural networks to extract features and classify the app as malware or benign. Shatnawi et al. proposed a novel technique using feature selection and reduction techniques for classification [9]. Furthermore, a study combined static and dynamic analysis using a deep learning-based model called AndroDeep, which extracts features from the code and applies a CNN for classification as malicious or non-malicious [10]. Lashkari et al.'s [11] study proposed a method for detecting Android malware using classification algorithms, utilizing feature selection techniques to classify apps as malware or benign. The study evaluated the effectiveness of different ML algorithms and feature selection techniques in detecting Android malware using static analysis on real-world malware and benign apps [3]. McDonald et al. studied the effectiveness of ML-based algorithms like RF, SVC, Gaussian Naive Bayes, and K-Means in classifying benign and malicious apps using permissions from manifest files. RF achieved the highest accuracy at 81.53% and recall rate of 85.85%. Commercial antivirus tools like Virus Total showed limited effectiveness in detecting unknown malware [12]. Khariwal et al. introduced R-MFDroid, an Android malware detection technique using seven static features from manifest files. The technique uses ML-based methods like SVC, XGBoost, RF, and Neural Network for classification. The Information Gain method is used to select top 25 features from .apk files. The SVC achieves an impressive accuracy rate and F1-score of 97% [13]. Kabakus and colleagues used a Convolutional Neural Network classifier on three datasets: Drebin, Androzoo, and VirusShare, analyzing features like permissions, API calls, and intents. Their study achieved a 90% accuracy rate, proving the effectiveness of their approach in classifying data [14].

2.2 Dynamic Analysis Using Cutting-Edge Algorithms

Many detection methods for Android malware have incorporated dynamic features with ML-based to enhance their detection accuracy. Zhou et al. developed a deep learning-based method for Android malware detection, which achieved an impressive accuracy rate of up to 99.4% when tested on a real-world dataset containing both malicious and benign apps [15]. Zhang et al. presented a study in which an automated Android malware detection system uses dynamic analysis to collect system call sequences and a CNN ensemble to classify the sequences. The proposed method achieves up to 99.95% accuracy on a dataset [16]. Similarly, another study has proposed an Android malware detection method that uses graph convolutional networks and dynamic analysis to classify apps based on their system call sequences. The proposed method achieves up to 98.8% accuracy [17]. Yang et al. introduced an ML-based approach for Android malware detection that utilizes system call traces obtained from dynamic analysis. The proposed method achieves up to 98.5% accuracy on the Drebin dataset of malware and benign apps [18]. Sun et al. presented an Android malware detection method that uses dynamic analysis and ensemble learning to classify apps based on their system call traces. The proposed method achieves up to 98.5% accuracy [19]. Furthermore, Li et al. proposed a hybrid framework for Android malware detection that combines dynamic analysis and ML techniques. The proposed framework achieves up to 97.7% accuracy. These studies demonstrate the effectiveness of incorporating

dynamic analysis and ML-based techniques in detecting Android malware, achieving high accuracy when tested on real-world malware and benign apps datasets [20]. Padmavathi et al. developed a dynamic analysis framework for Android apps, utilizing hidden malware data points. They used various unsupervised ML models for classification, using internal and external validation metrics. The k-means clustering model was found to be the most effective, with an accuracy rate of 88% [21]. Islam et al. developed a dynamic analysis method for android malware detection using weighted voting and ensemble ML techniques. They conducted extensive experiments, incorporating various ML-based classifiers and the Recursive Feature Elimination method, achieving a 95% accuracy rate in app classification [22]. Mahindru et al.'s study utilized a dynamic analysis method to classify Android apps based on API calls, including permissions, app ratings, and download numbers, with an accuracy rate of 98.8%, excluding malware and benign apps [23].

2.3 Hybrid Analysis Using Cutting-Edge Algorithms

Recent studies on Android malware detection have introduced a new method called Malconv, which uses hybrid analysis and convolutional neural networks. It achieves a high accuracy of 98.16% on a testing dataset and outperforms existing classifiers, making it effective in handling obfuscated and polymorphic malware [24]. Andro-Predictor is a real-time mobile malware detection system that uses a hybrid feature approach, combining system call sequences, API calls, and permissions. Its lightweight and efficient design makes it a promising solution for real-time malware detection on mobile devices [25]. The study Androdialysis presents a hybrid analysis framework for detecting Android malware, combining static and dynamic techniques. Experiments on 9,840 apps showed a high detection rate of 98.7% and low false positive rate [26]. A study presents a hybrid framework for Android malware detection that uses static and dynamic analysis. It uses permission, API call, and system call information to represent app behavior. A decision tree-based model classifies static features, while a deep neural network is trained on dynamic analysis features [27]. A study suggests a hybrid Android malware detection system that utilizes both static and dynamic analysis features, utilizing an ML-based model for high detection rates [28]. A hybrid approach to Android malware detection, combining static and dynamic analysis features, is proposed, utilizing an ML-based model for high accuracy [29]. Surendran et al. introduced TAN, a hybrid approach for Android malware analysis, which combines static and dynamic feature sets, including API calls and permissions, and uses an L2 regularization technique for classification, achieving a 97% accuracy rate [29]. Ullah et al. introduced TrojanDetector, it is a malware detection system that analyzes Android apps using a combination of static features (application-level and package-level) and dynamic features (user-level). The authors utilized traditional ML algorithms, including SVV, DT, LR, and RF, to classify these apps. The SVC algorithm was the top performer, achieving an impressive 96.64% accuracy rate in categorizing malicious apps [30]. Ahsan et al. developed an adaptive semi-supervised hybrid technique for detecting Android malware, combining CNN and LSTM deep learning algorithms. Their study, using the Drebin dataset, achieved a 99% accuracy rate, proving its effectiveness [31].

3 Data Preparation and Feature Engineering

This section aims to thoroughly understand the methodology used to create accurate and robust malware detection systems by providing a comprehensive overview of these critical aspects.

3.1 Android Dataset (Malware and Benign Apps)

The study utilized two commonly used datasets, namely Drebin [2] and Tezpur University Android Malware Dataset (TUANDROMD) [32] to conduct the experiments. Table 1 displays the percentage distribution of classes within the datasets, which serve as benchmarks for Android malware detection. The Tezpur University Android Malware Dataset, also known as TUANDROMD, is a collection of Android apps commonly used for research purposes in the field of cybersecurity [32]. Researchers at Tezpur University in India created a dataset containing over 5,000 Android apps, including both malicious and benign ones. TUANDROMD, a malware detection tool, has been used in studies to evaluate techniques, analyze malware characteristics, and understand its behavior on Android devices. In 2014, researchers [2] downloaded apps from the Google Play store and discovered many that were malicious. The study used the Drebin Dataset, which contains 5,560 malicious apps from 179 malware families, to represent malicious apps for experiments. To validate the app's benignness, the researchers downloaded over 8,000 Android apps from the Google Play store and used Virus Total's services to identify 5,721 purely benign apps. The remaining apps were discarded, resulting in a final dataset of 5,553 malware apps and 5,721 benign apps. The study aimed to identify the most harmful apps. The volume of both datasets is presented in Table 1.

Table 1: The volume of datasets

Datasets	Total volume	Training	Testing
Drebin	15,036	12,029	3,008
TUANDROMD	4,464	3,571	893

3.2 Data Preprocessing

The study analyzed data integrity in the Drebin dataset and the Tezpur University Android Malware Dataset (TUANDROMD), focusing on discretization, imputed missing values, and unbalanced data. It suggested removing low-quality variables and using data-cleansing procedures to identify and correct inconsistencies in the datasets. Identifying and removing duplicate samples from the datasets was a crucial step in the data preparation phase, achieved using the approach in the following formula. Let P and H be the package name and SHA256 hash value of each sample, respectively. The set of unique samples can be obtained by selecting the samples $s \in S$ such that s has a unique combination of P and H, as illustrated in Eq. (1).

$$\text{Unique} = s | s \in S, \forall s' \in S, (s / = s' \wedge Ps = Ps' \wedge Hs = Hs') \Rightarrow s' \in / \text{Unique} \quad (1)$$

We also handled missing feature values using mean or median imputation based on the distribution of the feature values. This can be expressed as $xf,\ i = mean\ (xf,\ j \mid xf,\ j = missing)$ or $xf,\ i = median\ (xf,\ j \mid xf,\ j = missing)$. These steps helped improve the quality and reliability of the Drebin and TUANDROMD datasets for training ML-based models.

3.3 Mini-Max Normalization Method

The min-max normalization method is a statistical technique that converts numerical data into a common scale, typically between 0 and 1, to ensure equal contribution of each feature to the analysis. It subtracts the minimum value of a feature from each data point and divides the result by the feature's range, resulting in a minimum value of 0. The minimax normalization technique is defined as in Eq. (2).

$$X_{norm} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{2}$$

where X is the original data point, X_{min} is the minimum value of the feature in the dataset, X_{max} is the maximum value of the feature in the dataset, and X_{norm} is the normalized value of X. Minimax normalization is particularly useful in ML, where features with large scales or ranges can dominate the analysis and produce inaccurate results. By applying Minimax normalization to each feature in the dataset, we can ensure that all features have an equal contribution to the analysis, regardless of their original scale or range.

3.4 Feature Selection

The feature selection process is a crucial step in classifiers to improve accuracy and reduce computational burden. It is particularly important when dealing with large datasets with numerous features and missing values. Research indicates that permission, API calls, and data flow are significant features for detecting malicious behavior [4,33,34]. The study utilized dex2jar to convert APK files to JAR files, identifying crucial features for malware detection. These include permissions, API calls, and intents, which are essential components of the Android operating system. Permissions restrict app access to necessary resources, intent allows communication between app components, and API calls provide access to system functionality and third-party libraries. The study utilized the Apktool tool to disassemble Android applications, producing files like AndroidManifest.xml and resources.arc. These files are crucial for analyzing features used in malware detection models. The AndroidManifest.xml file contains crucial information about the app's package name, components, permissions, and features. The study also compiles lists of permissions, API calls, and intents from the official Android documentation. These master lists [35] and intent lists [36] serve as valuable resources for developing ML-based models capable of accurately identifying malware in Android apps. The process involves decompiling each app and using a parser to create a comprehensive inventory of permissions, API calls, and intents. This data is integrated to generate three distinct feature vectors: Android Permission, Android API Calls, and Android Intent, which accurately represent the permissions, API calls, and intents used in the dataset, enabling the development of robust models. The study used Gain-ratio feature selection to identify the most informative features in a dataset. It calculated the gain ratio by dividing data based on a feature by its intrinsic information. Gain measures entropy reduction, while intrinsic information measures the amount needed to represent that feature's values. The gain of splitting the data on a feature X is given in Eq. (3).

$$Gain\,(X) = Entropy\,(\mathrm{P}arent) - \sum_{v \in Values(X)} \frac{|X_v|}{|Parent|} Entropy\,(X_v) \tag{3}$$

where Parent is the original dataset, X is the feature being considered for splitting, Values (X) is the set of all distinct values of X, and $|X_v|$ is the number of instances in Parent that have the value v for feature X. Entropy is a measure of the impurity of a dataset and is calculated as follows in Eq. (4).

$$Entropy\,(D) = \sum_{i=1}^{|c|} p_i \log_2 (p_i) \tag{4}$$

where D is a dataset, C is the set of possible class labels, and P_i is the proportion of instances in D that belong to class i. The intrinsic information of a feature X is given by in Eq. (5).

$$Intrinsic\,(X) = -\sum_{v \in Values(X)} \frac{|X_v|}{|Parent|} \log_2 \left(\frac{|X_v|}{|Parent|} \right) \tag{5}$$

The gain ratio of a feature X is defined in Eq. (6).

$$GainRatio\,(X) = \frac{Gain\,(X)}{Intrinsic\,(X)} \tag{6}$$

The Gain-ratio feature selection method is a method that selects features with a higher gain ratio, based on their intrinsic information, which is effective in handling both discrete and continuous data types and avoids bias towards features with numerous distinct values.

3.5 Feature Vector Extraction Approaches

The study uses two methods to rank features based on their ratings, without requiring a training process. These methods typically provide a score for each characteristic, which is then used to rank the features. The Chi-Squared test was used to calculate the statistical significance of the association between Permission, API calls, Intent features, and the target variable, malware detection. The Chi-Squared feature extraction is illustrated Eq. (7).

$$x^2 = \sum \frac{(O - E)^2}{E} \tag{7}$$

where X^2 is the Chi-Squared test statistic $\sum$ is the sum over all cells in the contingency table O is the observed frequency in a cell E is the expected frequency in a cell under the null hypothesis to identify the essential features for detecting Android malware and improve the accuracy of the detection model, a contingency table was created with the Permission, API calls, and Intent features as rows and the target variable (malware detection) as columns. The study used a Chi-Squared test formula to calculate the Chi-Squared test statistic and its associated p-value for malware detection. Observed frequencies were determined for each cell based on assigned feature values, and expected frequencies were calculated under the null hypothesis. Features with a high Chi-Squared test statistic and low p-value were selected for inclusion in the final feature set, enhancing the accuracy of the Android malware detection model.

3.6 Feature Selection Importance

The study utilized advanced feature selection techniques to enhance Android malware detection. The process involved refining Drebin and TUANDROMD, addressing data imbalances, and eliminating low-quality variables. The study also used Min-Max normalization to standardize numerical data, ensuring equal contribution to ML-based models. The focus was on permissions, API calls, and intents. Tools like dex2jar and Apktool were used to extract features from Android apps, targeting components essential for identifying malicious behavior. The study utilized Gain-ratio feature selection and the Chi-Squared test to identify informative features, enhancing malware detection efficiency and enhancing the accuracy and reliability of machine learning-based models in identifying Android malware, thereby reducing entropy reduction and statistical significance. The heatmap depicted in the Figs. 1 and 2 illustrates the correlation between different features within the Drebin and TUANDROMD datasets.

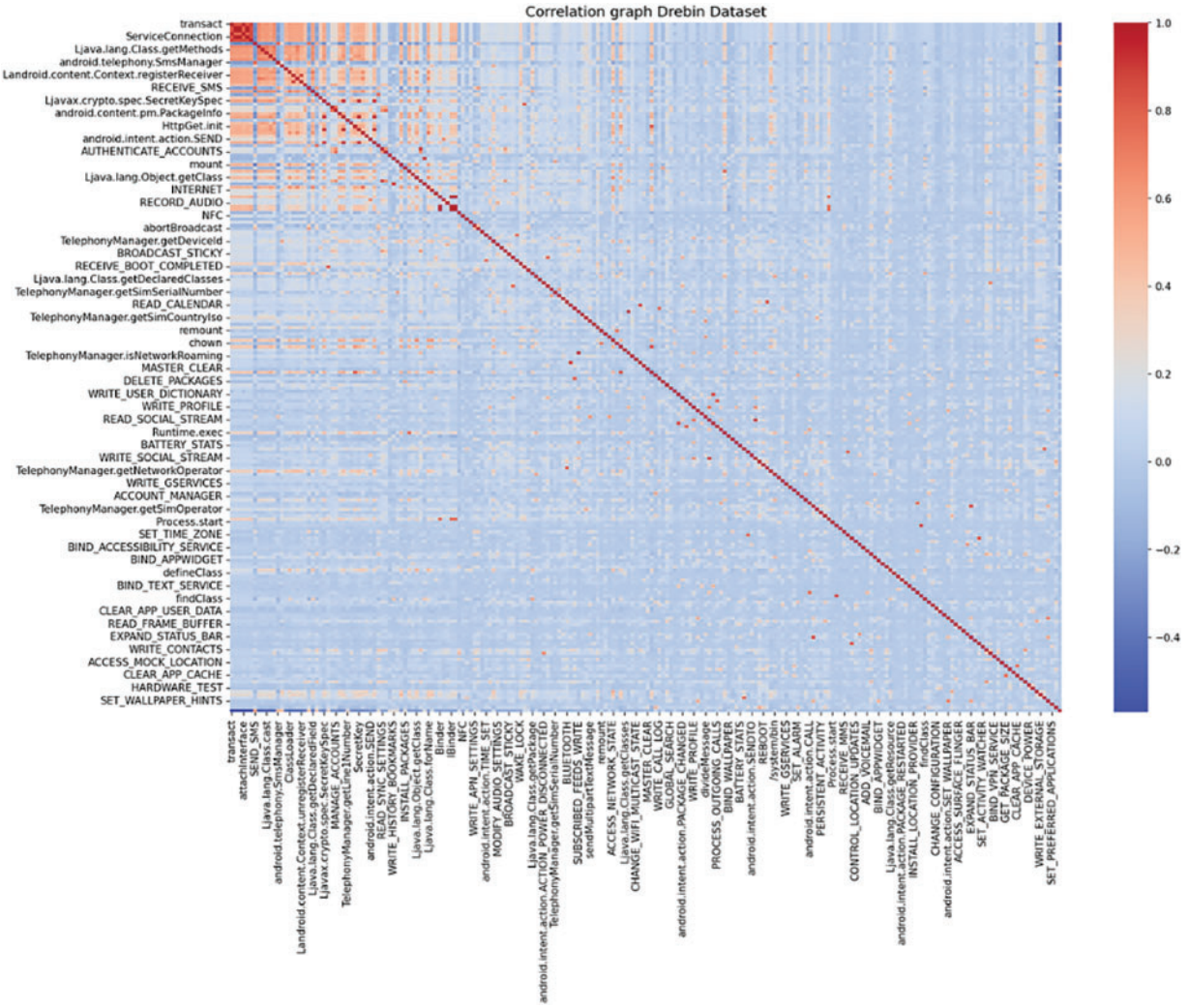

Figure 1: Distribution of key features in the drebin dataset

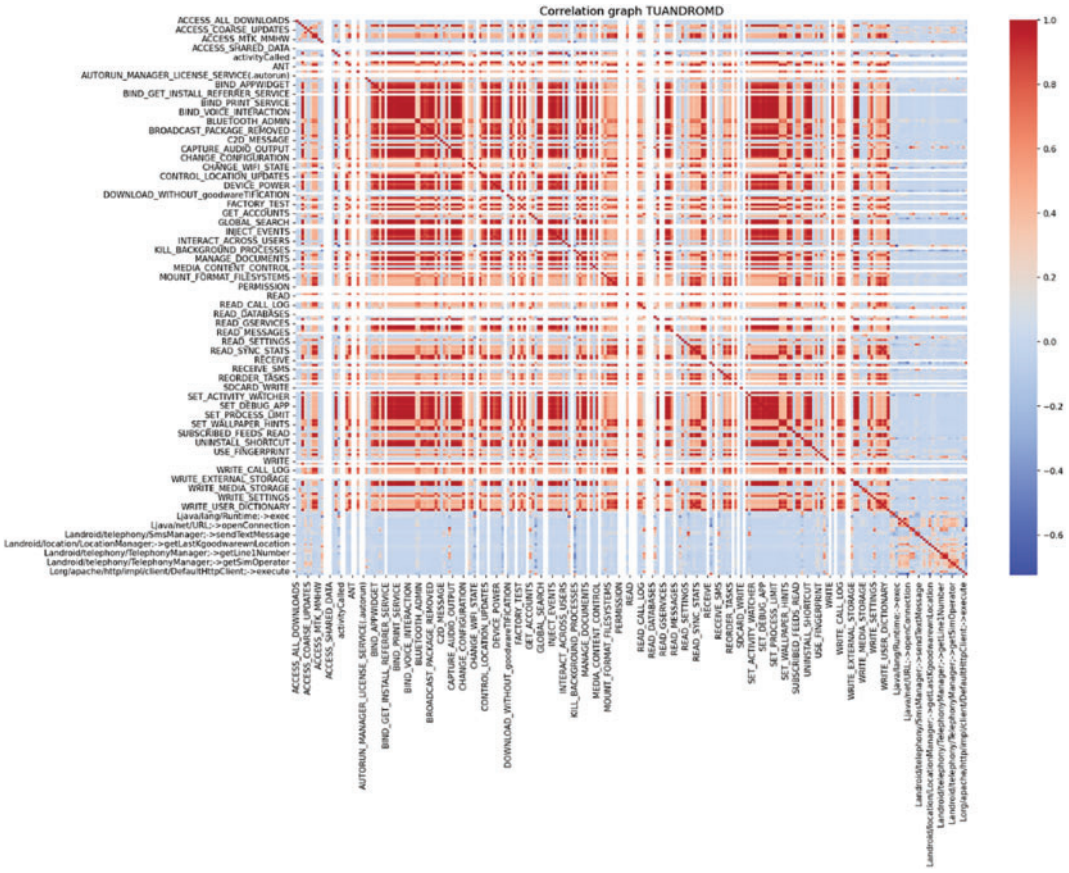

Figure 2: Distribution of key features in the TUANDROMD dataset

4 Performance Measurements

The study evaluates the effectiveness of our proposed algorithms in detecting Android malware using key metrics such as mean square error, Pearson's correlation coefficient, and root mean square error. It also considers essential classification metrics like accuracy, precision, recall, and AUC to assess the models' correctness, ability to minimize false positives, false negatives, and discriminative power.

5 Results

The proposed ML-based models were evaluated using Keras, TensorFlow library of Python, on standard Android malware datasets. A statistical analysis was conducted to assess the impact of these models, and the results were analyzed accordingly. Table 2 presents the platform utilized for detecting malware in Android apps.

Table 2: Optimal performance environmental requirements

Hardware	Software
RAM size 16 GB	Python version 3.7
G.P. U	Numpy version 1.24

(Continued)

Table 2 (continued)

Hardware	Software
	TensorFlow version 2.11 Keras version 2.11 Matplotlib version 3.4

5.1 Performance Analysis of the Proposed Algorithms

This study proposed three ML-based algorithms, namely SVC, KNN, and RF, to detect and categorize malicious Android apps. To test the effectiveness of the algorithms, this research used two well-known malware mobile datasets. Drebin dataset consists of over 15,036 Android apps classified as either malware or benign. The TUANDROMD dataset contains 4,464 malicious and benign apps to evaluate the detection of malware attacks on Android devices. These datasets have provided significant benchmarks for evaluating ML-based algorithms. However, the quality and variety of the training data and the algorithms' features and parameters might impair their ability to identify malicious apps. The suggested ML-based models have proven to be an effective method for detecting malicious Android apps.

The SVC approach demonstrated high efficiency and effectiveness in identifying malicious apps within the Drebin dataset while yielding good results in the TUANDROMD dataset. Detailed performance metrics for the SVC algorithm are presented in Figs. 3a–3f, and Table 3. The SVC approach was evaluated on the Drebin dataset, and the corresponding confusion metrics were analyzed. The SVC approach effectively detected malwares within the Drebin dataset, as evidenced by the very few FP and FN data. According to the outcomes, the majority of the 1,122 apps were labeled as malicious, indicating that the SVC classifier detected them as possibly posing a threat. Besides that, a proportion of the dataset 1851 was classified as TN and identified as benign apps, as depicted in Fig. 3a. This study reveals that within the TUANDROMD dataset, a substantial proportion of the standard data was categorized as TN, accounting for 169 of the dataset's apps. In contrast, 710 apps of the dataset were classified as TP and were identified as malicious, as depicted in Fig. 3b. The finding indicates that the SVC approach is proven to accurately classify all instances of malicious attacks as positive and all instances of average data as unfavorable, with fewer instances of misclassification.

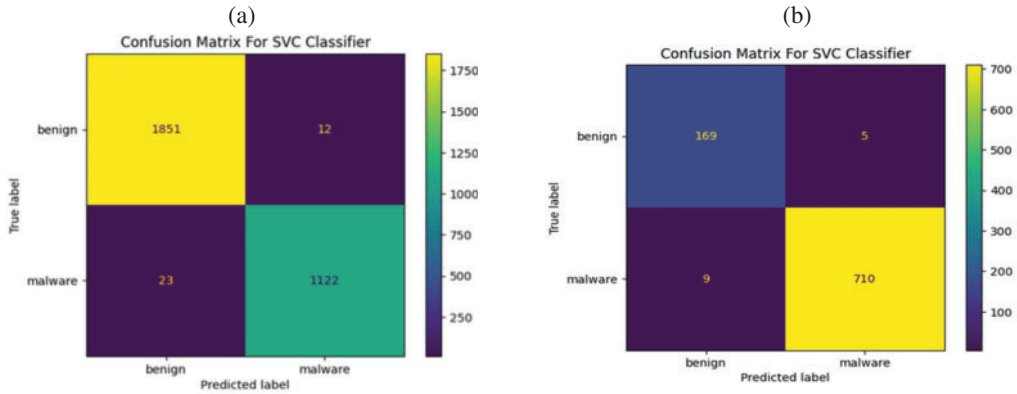

Figure 3: (Continued)

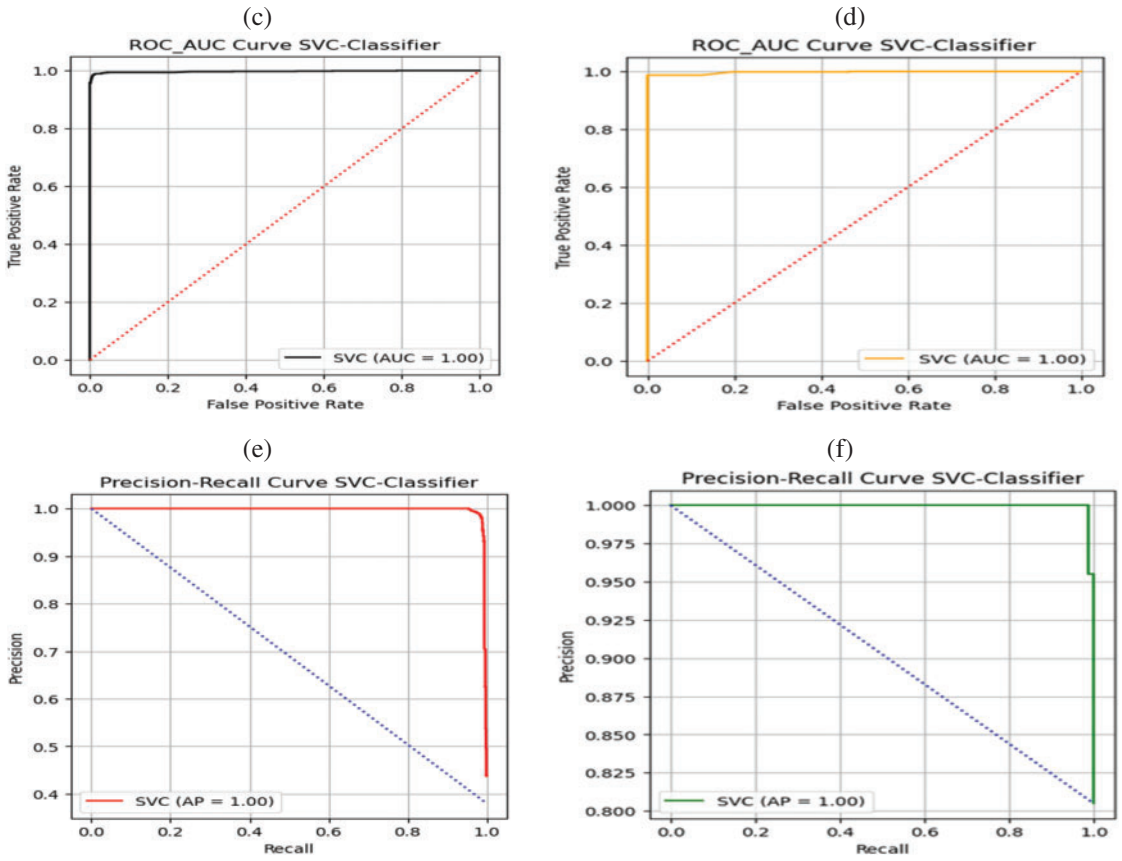

Figure 3: Performance comparison of SVC algorithm using Drebin and TUANDROMD datasets. Panels (a and b) show the confusion matrices, Panels (c and d) show the ROC-AUC, and Panels (e and f) show the precision and recall for Drebin and TUANDROMD, respectively

Table 3: SVC classifier performance on Drebin and TUANDROMD datasets

Datasets	Precision	Recall	F1-score	Accuracy
TUANDROMD	0.970	1.000	0.842	0.989
Drebin	0.981	0.978	0.935	0.986

The SVC could accurately classify malware and benign apps in both datasets, achieving a ROC-AUC value of 1.0 in each case, as represented in Figs. 3c, 3d. The proposed SVC achieved a ROC-AUC value of 1.0, suggesting it performed exceptionally well distinguishing between malware and benign apps. The assessment findings have confirmed the efficacy and reliability of the SVC approach. In particular, the low percentage of FP reveals that the SVC approach accurately classified normal data as negative without misclassifying them as potentially malicious. Figs. 3e, 3f depict the evaluation results of the proposed SVC algorithm based on both datasets for Android malware detection yielding 100 percent precision and recall. The graph illustrates the model's performance in recognizing Android malware and limiting FP and negatives. These findings significantly boost the effectiveness and potency of Android malware recognition and avoidance efforts, indicating that the SVC model

may be an effective tool for combatting Android malware. Figs. 4a–4f provide an overview of the performance of the KNN algorithm for detecting malware in both datasets using a value of $k = 5$. The results indicate that the KNN method performed significantly better in the TUANDROMD dataset, achieving a high accuracy rate of 0.989. Besides the Drebin dataset, the accuracy rate was slightly lower at 0.986, as shown in Table 4.

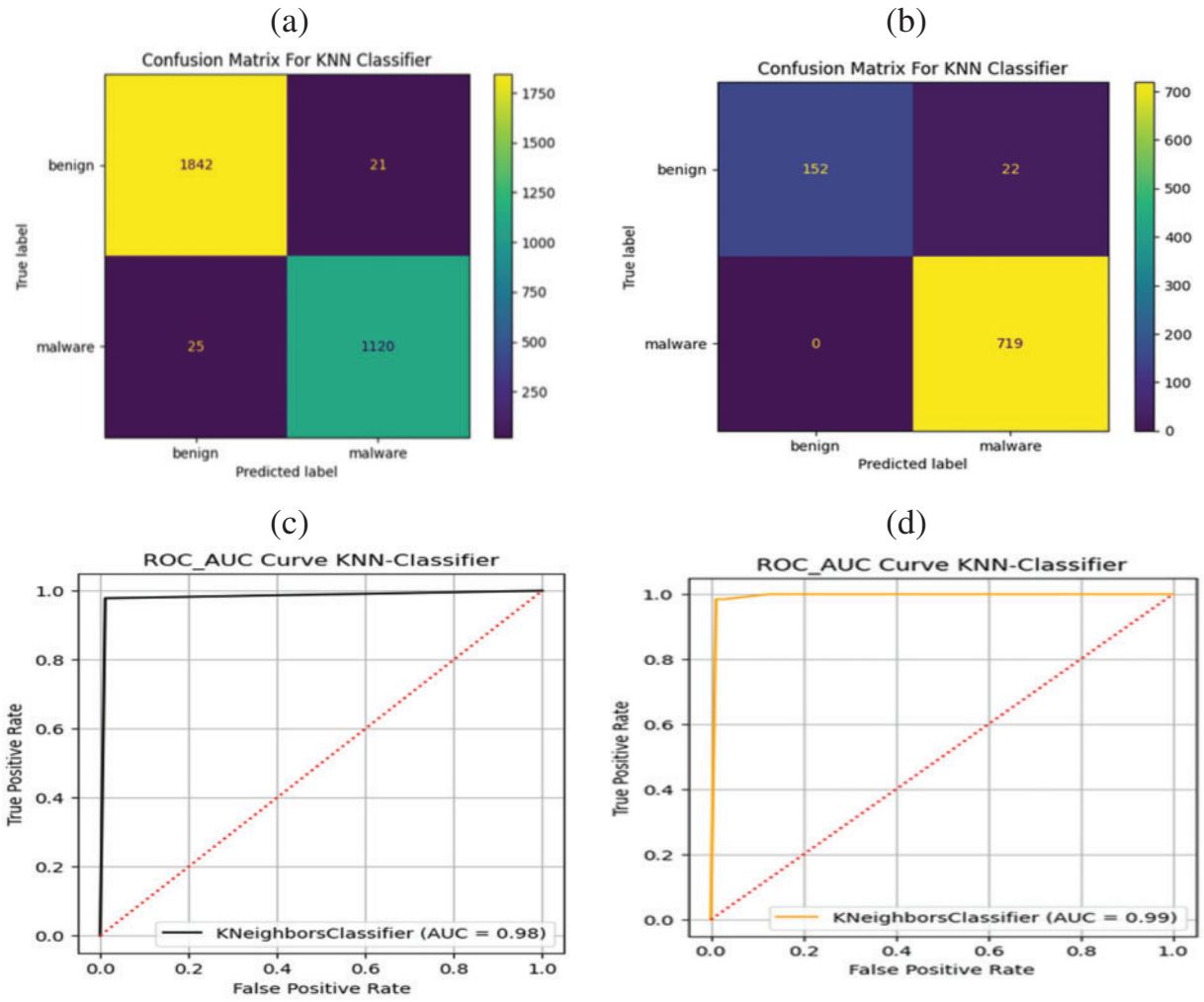

Figure 4: (Continued)

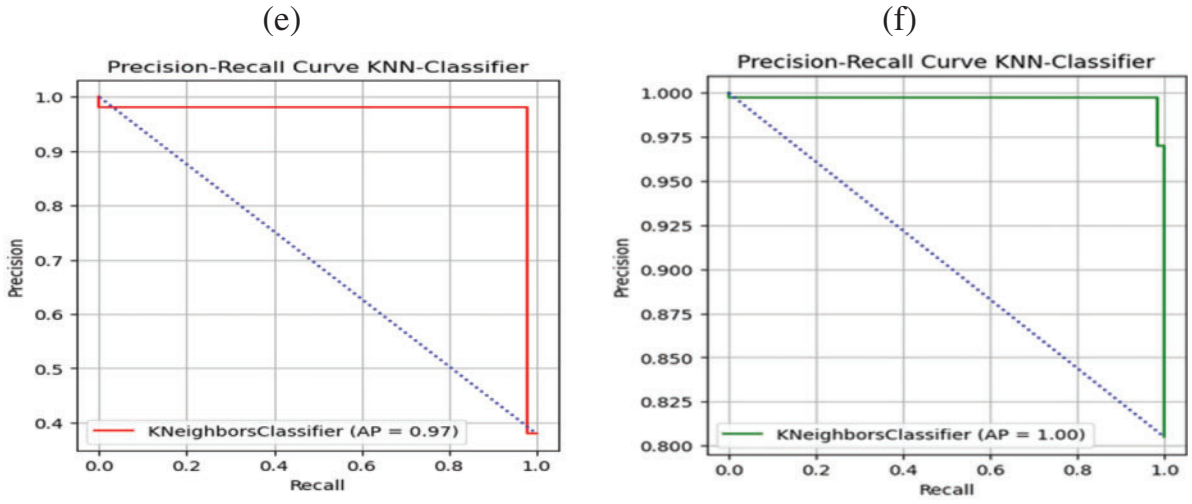

Figure 4: Performance assessment of KNN classifier, Panels (a and b) confusion matrices, Panels (c and d) ROC-AUC, and Panels (e and f) precision and recall for Drebin and TUANDROMD, respectively

Table 4: KNN classifier performance on Drebin and TUANDROMD datasets

Datasets	Precision	Recall	F1-score	Accuracy
TUANDROMD	0.970	1.000	0.842	0.989
Drebin	0.981	0.978	0.935	0.986

Figs. 4a, 4b present the confusion matrices for the KNN method. The KNN method achieved a higher TP rate in the Drebin dataset, classifying 1120 as TP malicious and 1842 as TN benign, with FP of less than 25, as illustrated in Fig. 4a in contrast, in the TUANDROMD dataset, 152 apps were correctly classified as TN benign apps, 719 were correctly classified as malware, and 0 were falsely identified as malicious FP, as illustrated in Fig. 4b. However, the overall accuracy of the KNN method was higher in the TUANDROMD dataset than in the Drebin dataset. The KNN model achieved ROC values of 0.98 and 0.99 for the Drebin and TUANDROMD datasets. Even though the suggested KNN model could not get a perfect score, the high ROC values indicate that it did very well distinguishing between malicious and benign apps. Figs. 4c, 4d present the ROC curve of the suggested KNN approach for both datasets, which provides further insights into the model's performance. Moreover, research findings indicate that the KNN model may identify Android malware well yet, the considerable investigation is needed to evaluate its resilience and scalability. KNN achieved a perfect score of 1 when tested on the TUANDROMD dataset for Android malware detection, as shown Fig. 4d indicating that the model could accurately identify all instances of malware in the dataset without any FP or negatives. Figs. 4e, 4f present the Precision and Recall graph of the KNN classifier. The model's accuracy and precision on this dataset suggest it may be suitable for recognizing similar malware in the future. The KNN model scored 0.97 on the Drebin dataset, significantly lower than the TUANDROMD dataset. The algorithm effectively identified malware in this dataset; however, there were some FP and negatives. This model performance gap may be due to the Drebin dataset's composition and features compared to the TUANDROMD dataset.

Figs. 5a–5f provide an overview of the performance of the RF algorithm for detecting malware in both datasets. RF approach demonstrated notably superior performance in detecting malware in the TUANDROMD dataset with a high accuracy level of 0.990. However, its accuracy rate in the Drebin dataset was slightly lower, measuring 0.988, which is presented in Table 5. The RF approach achieved a higher TP rate in the Drebin dataset, correctly categorizing 1117 data as TP malicious and 1855 as TN benign while misidentifying less than 28 data points as FP, are shown Fig. 5a on the other hand, in the TUANDROMD dataset, 174 apps were accurately categorized as TN benign apps, 710 were correctly classified as malware, and 9 were erroneously identified as malicious FP with 0 FN are shown in Fig. 5b.

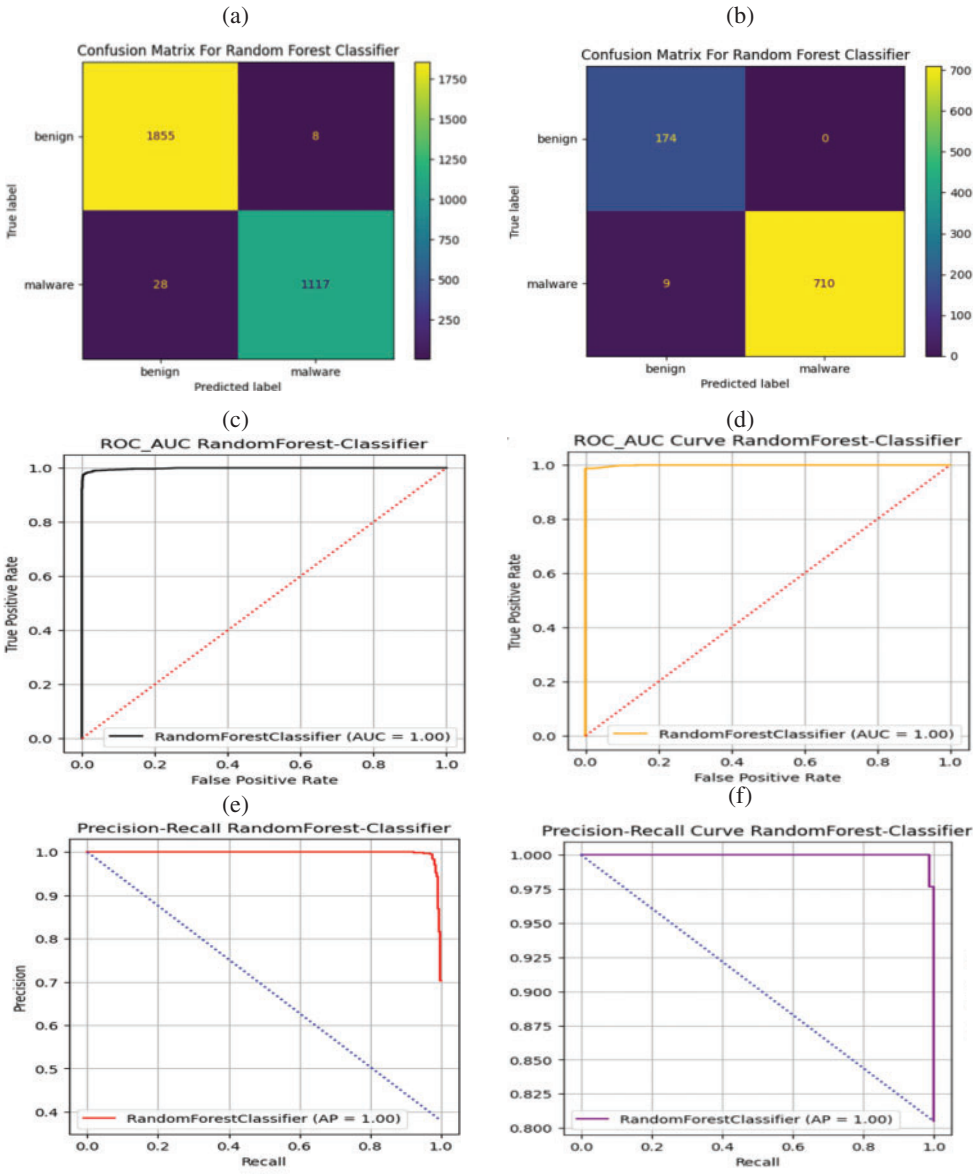

Figure 5: Performance analysis of RF classifier. Panel (a and b) confusion matrices, Panels (c and d) ROC-AUC, and Panels (e and f) precision and recall for Drebin and TUANDROMD, respectively

Table 5: RF classifier performance on drebin and TUANDROMD datasets

Datasets	Precision	Recall	F1-score	Accuracy
TUANDROMD	1.0	0.987	0.936	0.990
Drebin	0.993	0.976	0.949	0.988

However, the RF method demonstrated a higher overall accuracy in the TUANDROMD dataset than in the Drebin dataset. This study also evaluated the performance of the RF model in accurately classifying malware and benign apps. Our results demonstrated that the RF model achieved a perfect ROC value of 1.0 in both datasets, indicating its exceptional performance in accurately distinguishing between malware and benign apps, depicted in Figs. 5c, 5d. The RF model can be a practical approach for detecting Android malware, and further investigation is warranted to validate its robustness and scalability. Figs. 5e, 5f show the proposed model attained a perfect score of 1 for both precision and recall. This indicates that the RF model could accurately identify all instances of malware without any false positives or negatives. The high precision and recall values suggest that the RF model has a strong potential for detecting Android malware effectively.

5.2 Sensitivity Analysis Comparison of Proposed ML-Based Algorithm

This investigation proves helpful in identifying underlying patterns in the dataset by scrutinizing the input data. To determine the correlation between the input features and the classes, such as benign and malicious, this study utilized Pearson's correlation coefficient. The analysis revealed that certain features exhibited strong associations with the classes. Specifically, some features had significant relationships between the benign and malicious categories. To evaluate the prediction accuracy of our ML-based algorithms, this research employed several statistical metrics, including mean absolute error (MAE), mean squared error (MSE), and R-squared (R^2). The obtained prediction errors between the target class and the predicted values are presented in Table 6. The presented analysis provides valuable insights into the performance of the ML-based models, enabling us to identify and address any prediction errors. Table 6 shows the performance comparison between the proposed algorithms using two different datasets in terms of MAE, MSE, R^2-score, and sensitivity. In the TUANDROMD dataset, the RF algorithm exhibited excellent performance with an MAE and MSE of 0.010 and an R^2-score of 0.994, respectively, and a sensitivity of 100%. Similarly, in the Drebin dataset, the RF classifier performed well, achieving an MAE and MSE of 0.012, and R^2-score of 0.984, respectively, and a sensitivity of 0.996. On the other hand, the KNN classifier's performance was inferior to the RF classifier in both datasets. The KNN classifier, however, exhibited higher prediction errors and lower sensitivity than the other classifiers for both datasets. In the TUANDROMD dataset, the KNN showed an MAE and MSE of 0.025, an R^2-score of 0.985, respectively, and a sensitivity of 0.87.

In contrast, in the Drebin dataset, the KNN achieved an MAE and MSE of 0.015, an R^2-score of 0.980, respectively, and a sensitivity of 0.989. Finally, the SVC classifier also demonstrated good performance, achieving an MAE and MSE of 0.016 and an R^2-score of 0.990, respectively, and a sensitivity of 0.97 on the TUANDROMD dataset. Similarly, using the Drebin dataset, the SVC achieved an MAE and MSE of 0.012, an R^2-score of 0.985, respectively, and a sensitivity of 0.994. However, the SVC demonstrated superior performance by achieving the lowest prediction errors (MAE and MSE) and the highest R^2-score for both datasets. Furthermore, it exhibited the highest sensitivity for the TUANDROMD dataset, while the RF classifier had the highest sensitivity for

the Drebin dataset. Consequently, results indicate that the SVC algorithm is the most precise and dependable classifier for detecting Android malware in these datasets. Nonetheless, the RF classifier may perform better for specific datasets, like Drebin, where it achieved the highest sensitivity. While the KNN classifier may not be as precise as the other classifiers, it may still be a helpful option.

Table 6: The statistical analysis of the performance of ML-based models is carried out using the Drebin and TUANDROMD datasets

Dataset	Classifier	MAE	MSE	R^2-score
TUANDROMD	RF	0.0101	0.0101	0.9937
	KNN	0.0246	0.0246	0.9849
	SVC	0.0157	0.0157	0.9902
Drebin	RF	0.0120	0.0120	0.9841
	KNN	0.0153	0.0153	0.9799
	SVC	0.116	0.116	0.9846

6 Discussion

The employment of smartphones with novel functionalities and their correlated Android apps has escalated due to the rapid advancement of technology. Statista reported that 7,074 million smartphones will be used by 2024. This also challenges the researchers and developers of security mechanisms for these apps, originating in the new complexities and vulnerabilities of the Android apps that hackers can quickly exploit. Proper evaluation of app data concerning security is crucial, especially for Android apps in digital e-commerce, e-business, savings, and online banking. This is because such apps involve transmitting confidential and valuable information over mobile networks. To prevent security vulnerabilities in the network, ML-based algorithms are utilized for monitoring and detecting malicious attacks on Android apps. This research contributes significantly to cyber-security by developing an ML-based framework to identify signature database anomalies. As a result, the system is capable of detecting unknown malicious apps. The accuracy of our ML-based method for detecting Android malware is influenced by several key factors. The quality and diversity of the Drebin and TUANDROMD datasets provide a comprehensive training base. The chosen algorithms, including RF, SVC, and KNN, have inherent strengths in effective feature handling and classification capabilities, which significantly contribute to accuracy. Advanced feature selection techniques ensure the relevance and informativeness of the features used. Hyperparameter optimization of each algorithm optimizes performance, preventing overfitting or underfitting. Finally, the models' adaptability to the evolving nature of Android malware is essential for maintaining accuracy over time. These elements collectively enhance the models' ability to accurately detect malware. The KNN algorithm exhibited a high level of accuracy with a score of 0.989. The SVC and RF algorithms resulted in high accuracy in developing a robust smartphone security system against malware. The SVC model demonstrated an accuracy of 0.989, while the RF algorithm achieved 0.988 accuracies. Figs. 6a, 6b present the ROC-AUC comparison of the proposed classifiers and provides a visual representation of how well each classifier performed in distinguishing between positive and negative samples. The ROC-AUC comparison of the proposed classifiers shows that both SVC and RF achieved a perfect ROC-AUC score of 1.0 against both datasets. On the other hand, KNN performed slightly lower than SVC and RF with an AUC value of 0.99. The precision comparison of the proposed classifiers shows that both SVC and RF

achieved a perfect precision score of 100% against both datasets. On the other hand, KNN performed slightly lower than SVC and RF with a precision score of 0.97. These results indicate that SVC and RF classifiers effectively detect malware on both datasets. At the same time, KNN also provides good performance but with slightly lower accuracy, which can be observed visually in Figs. 6c, 6d.

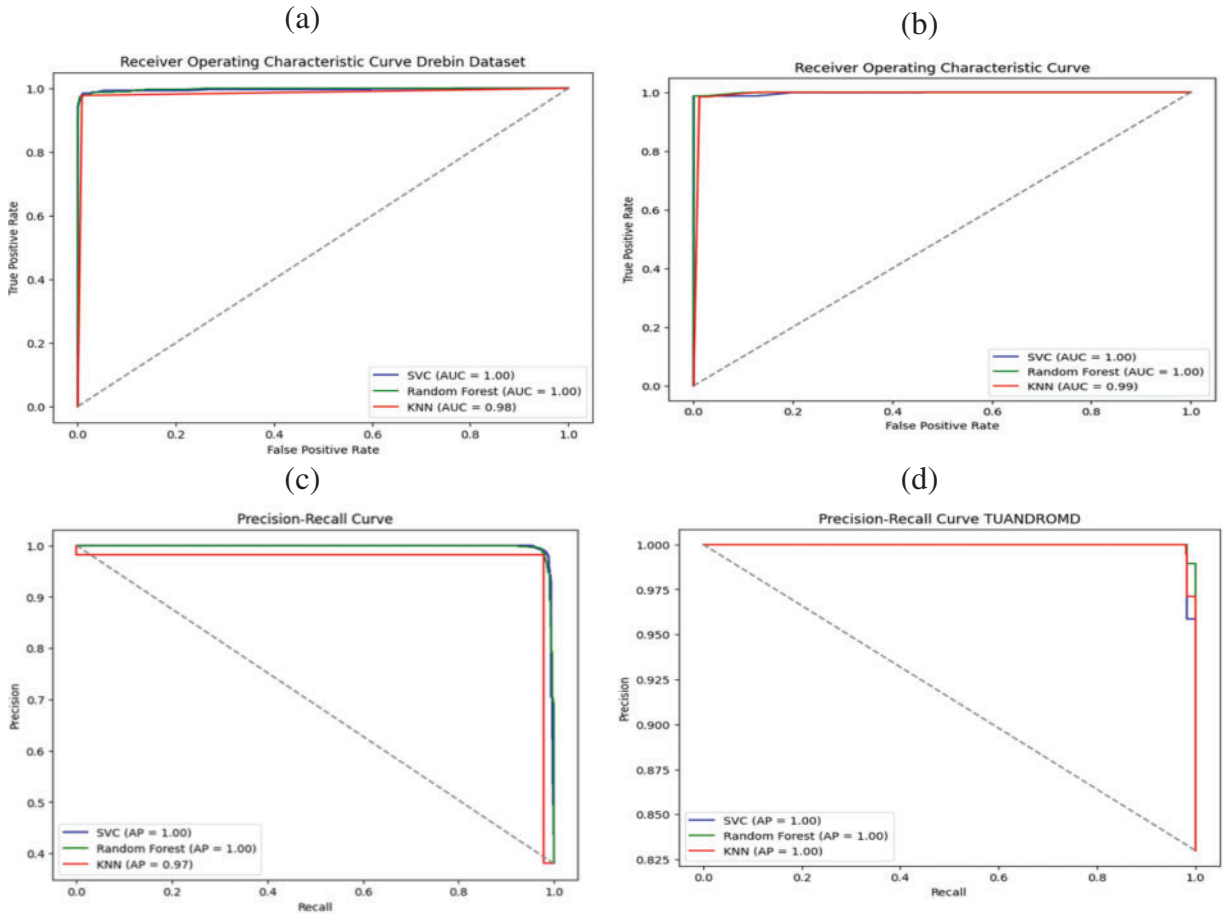

Figure 6: Performance analysis of SVC, KNN, and RF classifier: Panels (a and b) show the ROC-AUC, and Panels (c and d) show the precision and recall for Drebin and TUANDROMD, respectively

6.1 Comparative Evaluation Using Cutting-Edge Research

This study compares the proposed approaches with advanced ML and DL-based models designed to detect malware in Android apps for security purposes. The proposed scheme analyzes behavioral patterns within the dataset to identify malware. The results of our security framework are presented in Tables 7 and 8, and are compared to those from cutting-edge research utilizing both the Drebin and various other datasets. The study covers several models, such as CNN, RF, DT, and SVC. These models have shown a clear trend of improvement in effectiveness over time.

Table 7: The effectiveness of the proposed security system compared to existing ones was evaluated using the Drebin dataset

References	Datasets	Year	Model	ACC%
[37]	Drebin	2021	CNN	91
[38]	Drebin	2018	RF, J48, simple logistic, SMO	88–96
[39]	Drebin	2021	CBR, SVC, DT	95
[40]	Drebin	2019	RF with 100 decision trees	98.7
[41]	Drebin	2019	SCV	93.7
[42]	Drebin	2016	Random forest tree	97
[43]	Drebin	2021	CNN	98.2
Proposed	**Drebin**	**2023**	**SVC**	**98.6**
Proposed	**Drebin**	**2023**	**RF**	**98.6**

Table 8: The proposed security system's efficacy compared to existing ones using the different datasets

References	Datasets	Year	Model	ACC%
[44]	MalGenome, Kaggle, Androguard	2019	RF	93
[45]	Google play, Virus-Share, Mass-Vet	2018	LSTM	97.4
[46]	Genome, Intel-Security, MacAfee, Google play	2017	Deep CNN	87
[47]	MacAfee, MalGenome, Drebin	2017	Extra Trees	99
Proposed	**TUANDROMD**	**2023**	**SVC**	**98.9**
Proposed	**Drebin**	**2023**	**SVC**	**98.6**
Proposed	**TUANDROMD**	**2023**	**RF**	**98.9**
Proposed	**Drebin**	**2023**	**RF**	**98.8**

Table 7 shows the evolution of model accuracy in the Drebin dataset. The RF model, especially when combined with 100 decision trees as demonstrated by Kabakus et al. achieved an impressive 98.7% accuracy, surpassing earlier versions of the model. The proposed SVC and RF models achieved a high accuracy of 98.6%, surpassing other models such as CNN, which had an accuracy of 91% according to Millar et al. [37]. This represents a significant improvement in benchmark performance. The proposed system achieved a remarkably high level of accuracy in comparison to other systems utilizing the same Drebin Dataset. Table 8 compares the performance of models across various datasets, such as MalGenome, Kaggle, Androguard, and others. The models' performance varies significantly across different datasets. Xu et al. [45] achieved an accuracy of 97.4% using the Google Play and Virus-Share datasets. Another study proposed by McLaughlin et al. [46] achieved only 87% using a combination of different datasets. Moreover, another significant study is 'DroidSieve,' proposed by Suarez-Tangil et al. In this study, they employed an Extra Trees classifier, which achieved a high accuracy of 99%. It is crucial to note that while the DroidSieve study attained an accuracy of 99%, higher than that of the proposed study, their experiment was conducted using a limited set of only the 20 most prominent features. In contrast, our study utilized a much more comprehensive set of 241 features. This difference in the number of features considered is significant, as it suggests that

the proposed models in our study may offer a more robust and extensive analysis compared to the narrower focus of the DroidSieve study. Our proposed system demonstrated a remarkably high level of accuracy when compared with other systems using various datasets.

7 Limitation

The Android mobile OS has been equipped with ML-based algorithms to detect and prevent cyber threats. However, these methods have limitations, particularly in dealing with the ever-evolving landscape of modern cyber threats. The ML-based malware detection strategy relies on manual feature engineering, which is time-consuming and susceptible to human errors. This manual task can miss subtle and nuanced features of sophisticated malware, which are constantly evolving. This limitation hampers the ML models' ability to identify and adapt to advanced malware characteristics, limiting their effectiveness in real-world scenarios. Traditional ML algorithms can struggle with larger and more complex datasets, leading to stagnation or decline in algorithm performance. This is particularly critical in malware detection, where data is voluminous and highly intricate. While the use of ML algorithms to combat Android malware has shown promise, these limitations underscore the need for further refinement and evolution of the strategies.

8 Conclusion and Future Work

Cybercriminals are increasingly targeting cell phones due to their vulnerability to security breaches, particularly the Android OS. An emerging approach for detecting signature-based malicious attacks is through antivirus apps that utilize AI, ML-based algorithms to predict and identify new forms of malware. This study developed a security system using ML-based algorithms such as SVC, KNN, and RF. Two widely used datasets of Android malware apps, TUANDROMD and Drebin, were utilized for evaluation. The proposed system demonstrated promising results in identifying potential security threats in Android devices. A sensitivity analysis was conducted to evaluate the metrics MSE, RMSE, and R^2-score, which were used to measure errors between the predicted output and the target values during the validation phase. The RF algorithm was found to be the most accurate classifier in sensitivity analysis, with the lowest MAE and MSE values and the highest R^2-scores compared to KNN and SVC. The SVC method had the second-highest R^2-score after RF for the Drebin dataset, while the KNN method had the second-highest MAE and MSE values. The research findings have implications for real-time security systems for mobile apps, reducing the risk of malware attacks in critical applications involving sensitive data transactions. The distinct performance of the algorithms provides valuable insights for app developers and security teams in selecting the most suitable algorithm for their specific security needs. The reliability of SVC and RF in detecting malware makes them ideal choices for environments where precision is paramount.

Acknowledgement: This research was supported by Princess Nourah bint Abdulrahman University Researchers Supporting Project Number (PNURSP2024R346), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia and Beijing Natural Science Foundation (No. IS23054). The author would also like to thank Prince Sultan University, Riyadh Saudi Arabia for support of APC.

Funding Statement: This research was funded by Princess Nourah bint Abdulrahman University and Researchers Supporting Project Number (PNURSP2024R346), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Author Contributions: Ahsan and Jingsha, conceived the study and experimented. Nafei and Mahmood, designed the methodology. Saba and Alamri, reviewed, and revised the study. Rehman and Alamri, provided essential research resources. Ahsan and Nafei, contributed to the analyzed data. Rehman and Saba, conducted the proofreading of the study. All authors have read and agreed to the published version of the manuscript.

Availability of Data and Materials: This paper has previously been published as a preprint on Research Square at this link https://www.researchsquare.com/article/rs-2762659/v1.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Statista, "Number of apps available in leading app stores Q3 2022," 2022. Accessed: Jun. 26, 2023. [Online]. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores

[2] D. Arp, Q. Erwin, and K. R. Christian, "Privacy threats through ultrasonic side channels on mobile devices," *2017 IEEE Eur. Symp. Secur. Priv.*, vol. 1, no. 1, pp. 35–47, 2017. doi: 10.1109/EuroSP39530.2017.

[3] M. S. Akhtar and T. Feng, "Evaluation of machine learning algorithms for malware detection," *Sens.*, vol. 23, no. 2, pp. 946, 2023. doi: 10.3390/s23020946.

[4] P. Faruki, R. Bhan, V. Jain, S. Bhatia, N. El Madhoun and R. Pamula, "A survey and evaluation of android-based malware evasion techniques and detection frameworks," *Inf.*, vol. 14, no. 7, pp. 374, 2023. doi: 10.3390/info14070374.

[5] P. Buono and F. Balducci, "Visual discovery of malware patterns in android apps," in *Integrating Artificial Intelligence and Visualization for Visual Knowledge Discovery*, 2022, pp. 437–457. doi: 10.1007/978-3-030-93119-3_17.

[6] A. Habbal, M. K. Ali, and M. A. Abuzaraida, "Artificial intelligence trust, risk and security management (AI TRiSM): Frameworks, applications, challenges and future research directions," *Expert Syst. Appl.*, vol. 240, no. 1, pp. 122442, 2024. doi: 10.1016/j.eswa.2023.122442.

[7] G. Portokalidis and H. Asia, "Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," in *Proc. ACM SIGOPS/EuroSys European Conf. Comput. Syst.*, Leuven, Belgium, 2006, pp. 15–27. doi: 10.1145/1217935.1217938.

[8] A. D. Schmidt, J. H. Clausen, A. Camtepe, and S. Albayrak, "Detecting symbian OS malware through static function call analysis," in *2009 4th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Montreal, QC, Canada, 2009, pp. 15–22.

[9] A. S. Shatnawi and Y. Qussai Yassen, "An android malware detection approach based on static feature analysis using machine learning algorithms," *Procedia Comput. Sci.*, vol. 1, no. 1, pp. 653–658, 2022. doi: 10.1016/j.procs.2022.03.086.

[10] S. Thaler, V. Menkovski and M. Petkovic, "Deep learning in information security," arXiv preprint arXiv:1809.04332, 2018.

[11] A. H. Lashkari, A. H. Arash, A. K. Andi Fitriah, G. Hugo, K. F. Mbah and A. A. Ghorbani, "Towards a network-based framework for android malware detection and characterization," in *2017 15th Annu. Conf. Privacy, Secur. Trust (PST)*, Calgary, AB, Canada, 2017, pp. 233–23309. doi: 10.1109/PST.2017.00035.

[12] N. Herron, J. William Bradley Glisson, T. McDonald and R. K. Benton, "Machine learning-based android malware detection using manifest permissions," in *Proc. 54th Hawaii Int. Conf. Syst. Sci.*, 2021, pp. 1–6.

[13] K. Khariwal, R. Gupta, J. Singh and A. Arora, "R-MFDroid: Android malware detection using ranked manifest file components," *Int. J. Innov. Technol. Explor. Eng.*, vol. 10, no. 7, pp. 55–64, 2021.

[14] A. T. Kabakus, "DroidMalwareDetector: A novel Android malware detection framework based on convolutional neural network," *Expert Syst. Appl.*, vol. 206, pp. 117833, 2022. doi: 10.1016/j.eswa.2022.117833.

[15] M. Torres, C. I. Javier, and P. J. García-Nieto, "Machine learning techniques applied to cybersecurity," *Int. J. Mach. Learn. Cybern.*, vol. 10, pp. 2823–2836, 2019.
[16] M. Jaiswal, Y. Malik, and F. Jaafar, "Android gaming malware detection using system call analysis," in *2018 6th Int. Symp. Digit. Forensic Secur. (ISDFS)*, Antalya, Turkey, 2018, pp. 1–5. doi: 10.1109/ISDFS.2018.8355360.
[17] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal and Y. Xiang, "A survey of android malware detection with deep neural models," *ACM Comput. Surv.*, vol. 53, no. 6, pp. 1–36, 2020. doi: 10.1145/3417978.
[18] J. Gajrani, V. Laxmi, M. Tripathi, M. S. Gaur, A. Zemmari and M. Mosbah, "Effectiveness of state-of-the-art dynamic analysis techniques in identifying diverse android malware and future enhancements," *Adv. Comput.*, vol. 119, no. 1, pp. 73–120, 2020. doi: 10.1016/bs.adcom.2020.03.002.
[19] N. Ashraf, A. Masood, H. Abbas, R. Latif, and N. Shafqat, "Analytical study of hardware-rooted security standards and their implementation techniques in mobile," *Telecommun. Syst.*, vol. 74, pp. 379–403, 2020. doi: 10.1007/s11235-020-00656-y.
[20] H. N. Nguyen, F. Abri, V. Pham, M. Chatterjee, A. S. Namin and T. Dang, "MalView: Interactive visual analytics for comprehending malware behavior," *IEEE Access*, vol. 10, pp. 99909–99930, 2022. doi: 10.1109/ACCESS.2022.3207782.
[21] G. Padmavathi, D. Shanmugapriya and A. Roshni, "Performance analysis of unsupervised machine learning methods for mobile malware detection," in *2022 9th Int. Conf. Comput. Sustainable Glob. Dev. (INDIACom)*, New Delhi, India, 2022, pp. 201–206. doi: 10.23919/INDIACom54597.2022.9763180.
[22] R. Islam, M. I. Sayed, S. Saha, M. J. Hossain, and M. A. Masud, "Android malware classification using optimum feature selection and ensemble machine learning," *Internet Things Cyber-Phys. Syst.*, vol. 3, no. 1, pp. 100–111, 2023. doi: 10.1016/j.iotcps.2023.03.001.
[23] A. Mahindru and A. L. Sangal, "SemiDroid: A behavioral malware detector based on unsupervised machine learning techniques using feature selection approaches," *Int. J. Mach. Learn. Cybern.*, vol. 12, no. 1, pp. 1369–1411, 2021. doi: 10.1007/s13042-020-01238-9.
[24] I. Almomani, R. Qaddoura, M. Habib, S. Alsoghyer, A. AI Khayer and I. Aljarah, "Android ransomware detection based on a hybrid evolutionary approach in the context of highly imbalanced data," *IEEE Access*, vol. 9, no. 1, pp. 57674–57691, 2021. doi: 10.1109/ACCESS.2021.3071450.
[25] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in android," *J. Parallel Distrib. Comput.*, vol. 103, no. 1, pp. 22–31, 2017. doi: 10.1016/j.jpdc.2016.10.012.
[26] A. B. Yilmaz, Y. S. Taspinar, and M. Koklu, "Classification of malicious android applications using naive bayes and support vector machine algorithms," *Int. J. Intell. Syst. Appl. Eng.*, vol. 10, no. 2, pp. 269–274, 2022.
[27] A. Alzahem, W. Boulila, M. Driss, A. Koubaa, and I. Almomani, "Towards optimizing malware detection: An approach based on generative adversarial networks and transformers," in *Int. Conf. Comput. Collective Intell.*, Hammamet, Tunisia, 2022, pp. 598–610. doi: 10.1007/978-3-031-16014-1_47.
[28] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song and H. Yu, "SAMADroid: A novel 3-level hybrid malware detection model for android operating system," *IEEE Access*, vol. 6, no. 1, pp. 4321–4339, 2018. doi: 10.1109/ACCESS.2018.2792941.
[29] R. Surendran, T. Thomas, and S. Emmanuel, "A TAN based hybrid model for android malware detection," *J. Inf. Secur. Appl.*, vol. 54, no. 1, pp. 102483, 2020. doi: 10.1016/j.jisa.2020.102483.
[30] S. Ullah, T. Ahmad, A. Buriro, N. Zara, and S. Saha, "TrojanDetector: A multi-layer hybrid approach for trojan detection in android applications," *Appl. Sci.*, vol. 12, no. 21, pp. 10755, 2022. doi: 10.3390/app122110755.
[31] T. Shanshan *et al.*, "Mobile fog computing security: A user-oriented smart attack defense strategy based on DQL," *Comput. Commun.*, vol. 160, no. 1, pp. 790–798, 2020. doi: 10.1016/j.comcom.2020.06.019.
[32] P. Borah, D. K. Bhattacharyya, and J. K. Kalita, "Malware dataset generation and evaluation," in *2020 IEEE 4th Conf. Inf. Commun. Technol. (CICT)*, Chennai, India, 2020, pp. 1–6. doi: 10.1109/CICT51604.2020.9312053.

[33] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE Symp. Secur. Privacy*, San Francisco, CA, USA, 2012, pp. 95–109. doi: 10.1109/SP.2012.16.
[34] L. Li *et al.*, "IccTA: Detecting inter-component privacy leaks in android apps," in *2015 IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, Florence, Italy, 2015, pp. 280–291.
[35] H. Rathore, S. K. Sahay, R. Rajvanshi, and M. Sewak, "Identification of significant permissions for efficient android malware detection," in *Int. Conf. Broadband Commun., Netw., Syst.*, vol. 11, no. 1. 2020, pp. 33–52.
[36] M. Sewak, S. K. Sahay, and H. Rathore, "DeepIntent: Implicitintent based Android IDS with E2E deep learning architecture," in *2020 IEEE 31st Annu. Int. Symp. Pers., Indoor Mobile Radio Commun.*, London, UK, 2020, pp. 1–6. doi: 10.1109/PIMRC48278.2020.9217188.
[37] S. Millar, N. McLaughlin, J. M. del Rincon, and P. Miller, "Multi-view deep learning for zero-day android malware detection," *J. Inf. Secur. Appl.*, vol. 58, no. 1, pp. 102718, 2021. doi: 10.1016/j.jisa.2020.102718.
[38] A. Sinha, F. di Troia, P. Heller, and M. Stamp, "Emulation *vs*. instrumentation for android malware detection," *Digit. Forensic Investig. Internet Things (IoT) Devices*, 2021. doi: 10.1007/978-3-030-60425-7.
[39] Z. H. Qaisar and R. Li, "Multimodal information fusion for android malware detection using lazy learning," *Multimed. Tools Appl.*, vol. 81, pp. 12077–12091, 2022. doi: 10.1007/s11042-021-10749-8.
[40] A. T. Kabakus, "What static analysis can utmost offer for Android malware detection," *Inf. Technol. Control*, vol. 48, no. 2, pp. 235–249, 2019. doi: 10.5755/j01.itc.48.2.21457.
[41] S. Lou, S. Cheng, J. Huang, and F. Jiang, "TFDroid: Android malware detection by topics and sensitive data flows using machine learning techniques," in *2019 IEEE 2nd Int. Conf. Inf. Comput. Technol. (ICICT)*, Kahului, HI, USA, 2019, pp. 30–36. doi: 10.1109/INFOCT.2019.8711179.
[42] G. Meng *et al.*, "Semantic modeling of android malware for effective malware comprehension, detection, and classification," in *Proc. 25th Int. Symp. Softw. Test. Anal.*, Saarbrücken, Germany, 2016, pp. 306–317.
[43] L. N. Vu and S. Jung, "AdMat: A CNN-on-matrix approach to android malware detection and classification," *IEEE Access*, vol. 9, pp. 39680–39694, 2021. doi: 10.1109/ACCESS.2021.3063748.
[44] U. S. Jannat, S. M. Hasnayeen, M. K. B. Shuhan, and M. S. Ferdous, "Analysis and detection of malware in android applications using machine learning," in *2019 Int. Conf. Electr., Comput. Commun. Eng. (ECCE)*, Cox'sBazar, Bangladesh, 2019, pp. 1–7. doi: 10.1109/ECACE.2019.8679493.
[45] K. E. Xu, Y. Li, R. H. Deng and K. Chen, "DeepRefiner: Multi-layer android malware detection system applying deep neural networks," in *2018 IEEE Eur. Symp. Secur. Privacy*, London, UK, 2018, pp. 473–487.
[46] N. McLaughlin *et al.*, "Deep android malware detection," in *Proc. Seventh ACM Conf. Data Appl. Secur. Priv.*, Scottsdale, Arizona, USA, 2017, pp. 301–308. doi: 10.1145/3029806.3029823.
[47] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto and L. Cavallaro, "DroidSieve: Fast and accurate classification of obfuscated android malware," in *Proc. Seventh ACM on Conf. Data Appl. Secur. Priv.*, Scottsdale, Arizona, USA, 2017, pp. 309–320. doi: 10.1145/3029806.3029825.