Hindawi
Security and Communication Networks
Volume 2024, Article ID 7382302, 13 pages
https://doi.org/10.1155/2024/7382302

WILEY | Hindawi

Research Article

Effective and Efficient Android Malware Detection and Category Classification Using the Enhanced KronoDroid Dataset

Mudassar Waheed and Sana Qadir

School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan

Correspondence should be addressed to Sana Qadir; sana.qadir@seecs.edu.pk

Received 19 December 2022; Revised 3 November 2023; Accepted 16 March 2024; Published 8 April 2024

Academic Editor: Luigi Catuogno

Android is the most widely used mobile operating system and responsible for handling a wide variety of data from simple messages to sensitive banking details. The explosive increase in malware targeting this platform has made it imperative to adopt machine learning approaches for effective malware detection and classification. Since its release in 2008, the Android platform has changed substantially and there has also been a significant increase in the number, complexity, and evolution of malware that target this platform. This rapid evolution quickly renders existing malware datasets out of date and has a degrading impact on machine learning-based detection models. Many studies have been carried out to explore the effectiveness of various machine learning models for Android malware detection. Majority of these studies use datasets that have compiled using static or dynamic analysis of malware but the use of hybrid analysis approaches has not been addressed completely. Likewise, the impact of malware evolution has not been fully investigated. Although some of the models have achieved exceptional results, their performance deteriorated for evolving malware and they were also not effective against antidynamic malware. In this paper, we address both these limitations by creating an enhanced subset of the KronoDroid dataset and using it to develop a supervised machine learning model capable of detecting evolving and antidynamic malware. The original KronoDroid dataset contains malware samples from 2008 to 2020, making it effective for the detection of evolving malware and handling concept drift. Also, the dynamic features are collected by executing the malware on a real device, making it effective for handling antidynamic malware. We create an enhanced subset of this dataset by adding malware category labels with the help of multiple online repositories. Then, we train multiple supervised machine learning models and use the ExtraTree classifier to select the top 50 features. Our results show that the random forest (RF) model has the highest accuracy of 98.03% for malware detection and 87.56% for malware category classification (for 15 malware categories).

1. Introduction

In most parts of the world, almost everyone uses a smartphone to access the Internet. These devices make it very convenient to perform a variety of tasks including sensitive banking transactions and exchanging private messages. They also store personal data such as credit card numbers and passwords.

The compromise of a smartphone and the sensitive data it handles can lead to very serious consequences for users and organisations. In fact, according to the 2023 Global Mobile Threat Report, there has been a dramatic increase in the number of mobile users and devices being targeted by phishing attacks and mobile malware [1]. The main motivation of this research is to help counter the threat of mobile malware by developing an effective and efficient Android malware detector.

Because the operating system of a smartphone is its main component, it must be taken into consideration when developing an effective malware detection solution. According to the latest statistics, Android dominates the global mobile OS scene with 67.56% of the market share [2]. The other OSes have significantly lower share of the market (iOS has 31.6% and the remaining $< 1\%$ is composed of BlackBerry,

Windows, and Symbian). The lower price and open source nature of Android are the main reasons behind this substantial market share. However, the downside of this popularity is that there more than 400,000 new Android malware variants being detected per month [3].

Malware usually enters a smartphone through the installation of an infected app. It can also enter from a third-party Website or through a phishing attack. To minimise the chances of an infection, repositories like Google Play Store implement security checks (such as Google Play Protect) during the publication stage of an app [4, 5]. However, some sophisticated types of malware are able to bypass these checks. There are also a few app repositories and third-party Websites that do not include sufficient security checks during the publication stage.

To evade security checks on an app repository, malware authors use techniques like repackaging. During repackaging, an existing legitimate app is reverse engineered and malicious code is inserted. The modified app is then repackaged and uploaded onto the app repository. Malware authors also use obfuscation, antidynamic techniques, update attacks, mimicry, and reflection attacks to evade detection. During an update attack, malware is embedded into a legitimate app when an update is pushed. In obfuscation, sensitive methods or APIs are disguised in order to thwart analysis of the malware. To frustrate dynamic analysis, antidynamic techniques (that avoid executing malicious actions in the presence of a virtual or emulated environment) are used.

The explosive increase in malware targeting Android devices has made it imperative for researchers to automate the process of malware detection by using machine learning (ML) models. These models are trained on datasets compiled from output generated by the static or dynamic analysis of malicious and benign apps. The models trained with datasets compiled using only static analysis techniques are not effective against behaviour-modifying malware [6]. If dynamic analysis techniques are used, they should be carried out on a real device. Using an emulated environment instead of a real environment would result in a dataset that is limited because malware that use antidynamic techniques would not execute their malicious intentions in an emulated environment [7].

Another complication is the impact of changes in the Android platform and apps on malware detection. The Android ecosystem has evolved to add new functionality, maintain compatibility with new hardware and devices, and to incorporate improved security features. The negative consequences of such rapid transformation are the emergence of several forward and backward compatibility problems. The differences in behaviour and characteristics of Android apps across different versions and years have been explored in [8]. These frequent changes have a significant impact on malware detectors because of a phenomenon known as concept drift [9]. The impact, in terms of performance, was investigated in detail in [10]. The researchers trained five state-of-the-art ML-based Android malware detectors and a dataset with samples for one year and tested them on samples for the following year. Their results show that the performance of all detectors deteriorated significantly (30%–90% in some cases) within the span of that single year.

The performance of malware detection models is also affected by the evolution of Android malware. A malware detector can detect malware from different periods only if it is trained on a dataset that contains samples that span across as many years as possible. While some datasets have increased their size (to a few hundred thousand samples), the majority have failed to even include a time frame for their malware samples. One unique dataset that includes a time frame for 17,697 malware samples from 2010 to 2019 is introduced in [11]. However, this dataset has two important limitations. It only includes certain types of dynamic features (run-time traces of system calls) and it has very few samples for some years [11].

It is clear that an effective and efficient malware detector requires the hybrid analysis of samples and the compilation of a balanced dataset with samples from the largest possible time span. Also, the dynamic features should be captured by executing the sample on a real device instead of an emulator or virtual environment. The Android malware detector developed in this research is trained on a very comprehensive dataset that meets all of the abovementioned requirements. Specifically, it is trained on a subset of the KronoDroid [12] dataset for the malware detection and category classification.

The main contributions of this paper are:

(1) The creation of an enhanced subset of the KronoDroid dataset. This dataset is improved by adding a category label for each malware sample. The KronoDroid dataset was selected because

   (i) It is recent (2021)
   (ii) It contains malware samples spanning almost all the years of the Android platform (2008–2020)
   (iii) It has both static and dynamic features. Also, the dynamic features were captured by running the samples on a real device.

   These characteristics make our model effective for detection of evolving and antidynamic malware.

(2) An effective and efficient supervised ML model for malware detection and category classification.

   Initially, four supervised ML models were trained using the enhanced subset of the KronoDroid dataset mentioned in contribution 1. The performance of these models was evaluated and the model with the best performance was selected. To minimize computational cost, only traditional ML algorithms were used and only the top 50 features were selected for training.

The rest of this paper is organised as follows. Section 2 discusses related work and Section 3 presents the research methodology. In Section 4, the results are analysed and discussed. A comparison of the developed malware detector with existing malware detectors is also included in this section. Section 5 provides the conclusion and future work.

2. Related Work

2.1. Approaches to Malware Detection. There are three main approaches for malware analysis. The static approach extracts features such as permissions, intent filters, strings, services, activities, dalvik opcodes, and metadata from an app without executing it. The dynamic approach extracts features from an app by running it in an emulated environment or on a real device. Examples of features extracted during dynamic analysis include system calls, binders calls, API calls, network usage, and memory usage. The hybrid approach combines the use of static and dynamic analysis.

After analysis, the collected features are compiled into a dataset that is used to train and test a ML model. The rest of this section reviews existing research on ML-based malware detection and category classification.

2.2. Review of Static Analysis Techniques for Malware Detection. In [13], a model based on the Cat-Boost algorithm was developed for malware detection and family classification using static features (permissions and intents). For benign apps, the Drebin dataset was used, but for malicious apps, a new dataset was compiled. The developed model had an accuracy of 97.40% for malware detection and 97.38% for family classification. The limitations of this model include its failure to detect some advanced evolving malware and its low detection rate for the Linux-Looter ransomware family.

In [14], a malware detection and categorisation model based on deep image learning was proposed. This model used static features such as activities, services, broadcast receivers/providers, intent actions, permissions, and metadata. For feature selection, the ExtraTree classifier was used. The main contribution of this work was the generation of a large dataset (i.e., CCS-CICAndMal2020) with 200,000 malicious apps from 12 malware categories and 191 malware families. The reported results include a detection accuracy of 93%. However, like [13], this work also relied on static features only.

In [15], pairs of permissions were extracted from the manifest file of malicious and benign apps and used to construct a graph. The graph was used to train a ML model. The reported detection accuracy was 95.44% but any malware that did not utilise permissions was not detected.

In [16], permissions and APIs (along with their relationships) were extracted for around 30,000 benign apps and 15,000 malicious apps. The resulting dataset, though comprehensive, was imbalanced and limited to malware until 2015.

A probabilistic discriminative model was developed in [17] to detect malicious samples based on decompiled source code and permission data. The dataset contained almost 11,000 samples (9% malicious and 91% benign), some of which were very old. The downside of this model was its inability to detect malware that used byte code encryption or obfuscation.

2.3. Review of Dynamic Analysis Techniques for Malware Detection. In [7], a dynamic analysis technique called Entropylyzer was developed. This work analysed the behaviour of malware samples by running them in an emulator. Malware from 12 categories and 191 families was analysed using six classes of features, and the extracted data were used to compile the CCS-CICAndMal2020 dataset. Shannon's entropy was employed for feature ranking and different ML models were trained. The overall accuracy reported for malware category classification and family identification was high. However, it is important to note that during dynamic analysis, some malware failed to run because it detected the presences of a virtual environment. This limitation can only be overcome if dynamic analysis is carried out on a real device.

Similarly, the authors in [18] also relied on an emulated dynamic malware analysis platform and used multiple types of features (e.g., system calls, binder calls, and composite behaviours). The proposed semisupervised deep neural network approach for malware category classification also achieved good results. Like [7], they state that some malware samples did not execute after detecting the presence of an emulated environment.

Droidbox [19], DroidMat [20], and AMAT [21] were developed after conducting dynamic analysis in an emulator or sandbox. The feature types obtained included permissions, intents, and API calls. Despite the excellent performance of these models, these studies failed to analyse malware with antiemulation mechanisms. In other words, their dataset did not include all the features that represent the true nature of the sample. It should be noted that the ability to detect the use of an emulator is not limited to malware. Other apps (e.g., Telephony Manager) are also able to detect the presence of an emulator using Android API like *methodTelephonyManager.getDeviceId()* [22].

2.4. Review of Hybrid Analysis Techniques for Malware Detection. The need to conduct dynamic analysis on a real device is also highlighted in the few malware detection studies that adopt hybrid analysis approach. In [23], a malware classification technique based on pseudolabel stacking auto encoders is proposed. Malware, from 5 categories, was executed in a virtual machine introspection (VMI)-based system. The proposed model detected and classified malware with an accuracy of 98.28%. Although this study uses an emulated environment, it provides evidence that the hybrid analysis approach provides higher accuracy than only static analysis or only dynamic analysis. In [24, 25], the use of hybrid analysis is further supported by its use for effective identification of resource misuse and malware detection.

2.5. Impact of Changes in Android Ecosystem on Malware Detection. The evolution of the Android platform and its app structure has been explored in several studies along with its impact on the detection of Android malware. Some of these studies also investigate how the evolution of Android malware affects the performance of malware detectors.

In [26], a toolkit was used to mine different app platforms and user characteristics with the aim of studying the health and sustainable security of different apps. Even though this study provides significant insight into the evolution of the Android platform, the approach used has its limitations in terms of practicality and feasibility. It requires continuous data mining, crawling, and autoupdating to achieve durable results.

In [27], the static and dynamic characterization of Android apps developed between the years 2010–2017 was carried out. The authors present important findings along with recommendations related to app structure, behaviour, and evolution. However, because the researchers only analyse benign apps, the recommendations cannot be generalized to malicious apps because malicious apps are very different from benign apps in terms of their metrics and behaviour.

In [28], a self-evolving detection solution, called Droidevolver, was developed (using a dataset consisting of API calls). The dataset used consists of 34,722 malicious samples from 2011 to 2016 and the final model achieved a *F*-measure of 95.27%. This score decreased by 1.06% per year (on average) until it reached 89.97% in 2016. Because Droidevolver is based on static analysis, it inherits the limitations of this approach. Furthermore, the most recent samples in its dataset are more than 6 years old.

In [29], the evolution of malware was explored using static analysis of code fragments (via a technique known as differential analysis). Most importantly, this work revealed alterations in several malware characteristics over time in order to evade detection. Because this study is based solely on static analysis, it inherits the limitations of this approach. It highlights the importance of including dynamic analysis and using carefully crafting feature engineering processes on a balanced and recent dataset to effectively detect evolving malware.

In [30], the dynamic evolutionary behaviour of benign and malicious Android apps on code-level execution was evaluated. The dataset used consists of 15,451 benign samples and 15,183 malware samples from 2010 to 2017. The main contribution of this research is the uncovering of several important metrics that can help differentiate between benign and malicious apps. These metrics could be used to develop a durable malware detection solution but its effectiveness depends on the stability of certain patterns in different versions of evolving malware. Malware patterns that are not represented by these metrics will not be detected by such a malware detection solution. In addition, the solution will be subject to the limitation of dynamic analysis being conducted in an emulated environment.

To summarise, it is clear that hybrid analysis is the most promising approach for ML-based malware detection and category classification. It is also evident that dynamic analysis should be conducted on a real device to capture features from malware capable of modify its behaviour in the presence of an emulator. Furthermore, to be effective, a ML model should be trained using a dataset that includes malware that dates back to the start of the Android platform. This is important for the detection of malware that evolves over time (referred to as concept drift). Formally, concept drift refers to the change in relationship between the input variable(s) and the target variable over time. This change has a negative impact on the accuracy of a trained model (as highlighted by Droidevolver), and because malware evolves very rapidly, it is important for a dataset to be extensive and to include both old and new samples. It is for these reasons that the malware detection and category classification solution developed in this paper uses the recent, comprehensive, and hybrid analysis-based KronoDroid dataset.

3. Methodology

The steps in the methodology are presented in Figure 1.

3.1. Acquisition of Dataset. Android apps use a variety of features or attributes for performing different actions. It is crucial to select the right set of features and to this end several Android datasets have been published. These datasets differ in the number of samples, the type of attributes, and the date of malware publication and collection. In this research, we use a subset of the KronoDroid dataset published in [31]. This dataset consists of 41,382 malicious apps and 36,755 benign apps. It also includes samples from the almost entire history of Android, starting from 2008 and ending in 2020. This is the first hybrid dataset to take the time variable into account in such detail. A total of 200 static features and 289 dynamic features are extracted by running each app on a real device. These features are mostly numeric (e.g., number of times a system call is invoked by an app or the number of permissions requested by an app). Also, each sample in the dataset is given two labels. The first label identifies the sample as either benign or malware and is relevant for malware detection.

Because there are only two possible classes, malware detection is a binary classification task. The second label refers to the name of the malware family. Because there are more than two possible malware families, detecting the family is a multiclass classification task. Similarly, detecting the category to which a malicious app belongs is also a multiclass classification task. It should be noted that the original KronoDroid dataset does not contain a label for the category of malware.

3.2. Data Preprocessing. In the exploratory data analysis (EDA) step, the data are examined to understand its structure and important attributes. We used Python libraries to determine the weight of different features and to check for missing values, null values, and outliers. The KronoDroid dataset includes a few non-numerical features such as hash values and date of publication. These non-numerical features have informative value and do not have any impact on the output. This was verified through feature selection as explained in the next section.

In the Data Cleaning and Data Integration stage, the Pandas data frame was used to remove (e.g., non-numerical features), merge, and finally generate a clean version of the dataset [32].

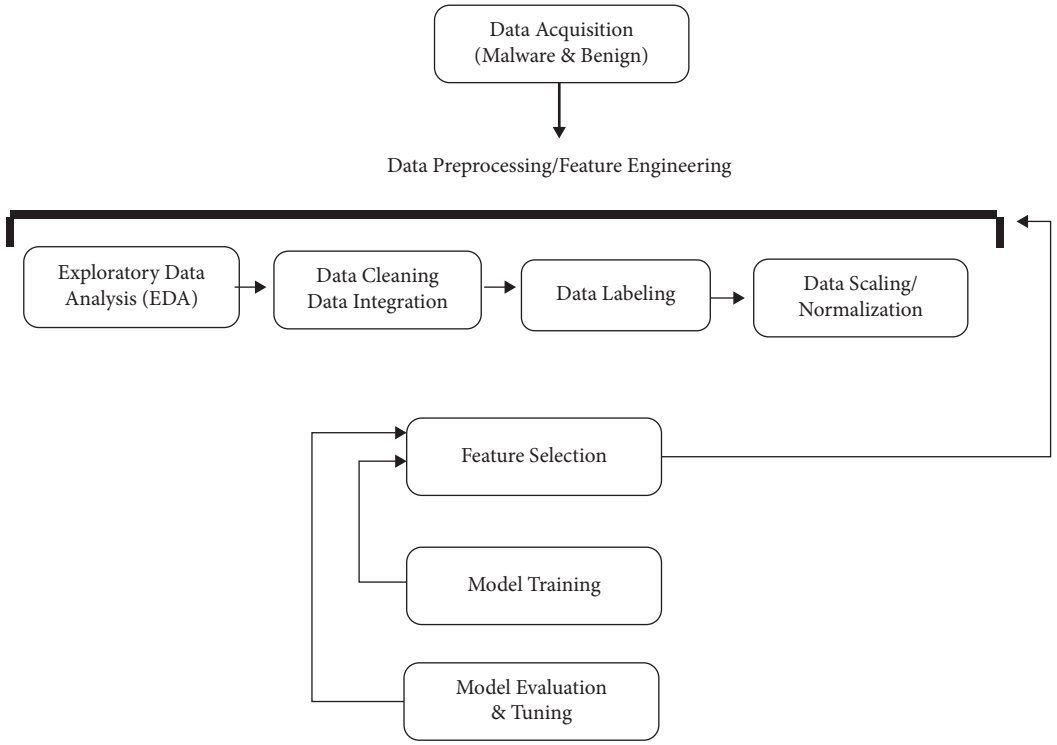

Figure 1: Methodology used to develop the malware detection model.

In the Data Labelling step, each malicious sample was labelled with the name of its malware category. The malware category label was obtained from VirusTotal [33], *F*-Secure [34], and FortiGuard [35] online repositories. Any discrepancy (e.g., malware labelled as Riskware by one repository and Adware by another repository) was resolved by cross-checking the label given to the sample in other datasets. If a sample is not found in any dataset, it was given the label assigned to it by the most number of repositories. In this way, up to 70% of the samples were labelled. The remaining 30% of the samples were not labelled and removed from the dataset. Their labelling is left for future work. The final modified and improved version of the dataset has been made publicly available on Github [36] for the research community. Table 1 presents the number of malware samples in each of the 14 categories (+1 unknown/blank category).

In the final Data Normalization/Scaling stage, the MinMax scaling technique was applied [37]. This technique is known to provide good results [38].

3.3. Feature Selection. Features selection techniques can be divided into the following three main types: filter methods, wrapper methods, and embedded methods. Wrapper and embedded methods are computationally expensive and not recommended for large dataset with wide dimensions [39]. From the different filter methods available, the ExtraTree classifier and the mutual information algorithm were selected because of their effectiveness and relevance for classification tasks [14, 40].

Table 1: Number of samples in each malware category.

	Malware category	Number of samples
1	Adware	11185
2	Trojan	6690
3	Trojan-SMS	6475
4	Riskware	5340
5	Trojan-spy	4281
6	Ransomware	1964
7	Trojan-banker	1681
8	Backdoor	1095
9	Scareware	1036
10	PUA	657
11	File-infector	436
12	Spyware	260
13	Trojan-dropper	62
14	Trojan/Riskware	54
15	Blank-category	166

Extra Trees builds multiple decision trees in parallel, where each tree is constructed using a random subset of the features and data points. During the tree construction process, it randomly selects feature splits, making it less prone to overfitting. By evaluating the importance of features across multiple trees, Extra Trees implicitly ranks the features based on their contribution to reducing impurity (Gini impurity for classification) and making accurate predictions. Mutual information is used to quantify the relationship between individual features (independent variables) and the target variable (dependent variable). It helps assess the relevance of each feature in predicting the target variable.

Table 2: 12 generated feature sets.

	Number of features	Feature selection algorithm
1	Top30	ExtraTree classifier
2	Top30 MinMax	Normalization and ExtraTree classifier
3	Top50	ExtraTree classifier
4	Top50 MinMax	Normalization and ExtraTree classifier
5	Top100	ExtraTree classifier
6	Top100 MinMax	Normalization and ExtraTree classifier
7	Top30	Mutual information
8	Top30 MinMax	Normalization and mutual information
9	Top50	Mutual information
10	Top50 MinMax	Normalization and mutual information
11	Top100	Mutual information
12	Top100 MinMax	Normalization and mutual information

Table 3: Accuracy of malware detection.

Features	RF (%)	DT (%)	KNN (%)	SVM (%)
Top30 ExtraTree classifier	95.99	**96.82**	91.53	84.28
Top30 mutual information	96.57	94.88	90.47	82.03
Top30 MinMax ExtraTree classifier	96.33	95.58	95.62	95.25
Top30 MinMax mutual information	96.63	94.94	95.28	91.06
Top50 ExtraTree classifier	97.72	95.88	86.66	68.66
Top50 mutual information	96.85	95.14	88.97	73.08
Top50 MinMax ExtraTree classifier	**97.98**	96.44	96.78	95.26
Top50 MinMax mutual information	96.84	95.21	95.66	92.03
Top100 ExtraTree classifier	97.72	95.58	89.90	69.17
Top100 mutual information	97.51	81.78	89.66	74.14
Top100 MinMax ExtraTree classifier	97.73	96.10	**96.85**	95.47
Top100 MinMax mutual information	97.55	82.55	96.77	**95.54**

Bold values are highest values/results of each classifier.

We tried many different features sets starting from top 5 features, top 10 features, and top 20 features but finally, 12 different feature sets with the best classification results were selected (see Table 2).

3.4. Model Training. We selected the following four supervised ML algorithms for this study: random forest (RF), decision tree (DT), K-nearest neighbour (KNN), and support vector machine (SVM). The dataset was split into two parts as follows: 70% for training and 30% for testing. Each algorithm was trained on each of the 12 feature set shown in Table 2.

3.5. Model Evaluation and Tuning. In this step, the performance of each model was evaluated using the metrics accuracy, precision, $F1$-score, and recall. The results are discussed in the next section.

4. Results

4.1. Initial Results. For malware detection, the accuracy of each model is shown in Table 3. Top50 MinMax ExtraTree classifier has the highest accuracy (97.98%) using RF. For malware category classification, the accuracy of each model is shown in Table 4. Again, Top50 MinMax ExtraTree classifier has the highest accuracy (87.24%) using RF. In both tables, the values in bold are the highest values for each model. These Top50 features include dynamic features (e.g., system calls) and static features (e.g., permission and intents).

The next highest accuracy for malware detection was obtained using Top100 MinMax ExtraTree classifier feature set and RF. For malware category classification, it was Top50 ExtraTree classifier and RF.

From Tables 3 and 4, we can also note that features sets with top 30 have the lowest accuracy. Features sets with top 100 have higher accuracy than those with top 30 but lower accuracy than feature sets with top 50. This is because the top 30 feature sets are missing some important features that have an impact on the output. These features are included in the top 50 features set which is the reason for their higher accuracy.

When top 100 feature sets are used (instead of the top 50 feature set), some features that have a low impact on the output are included. These features add noise and complexity to the classifier thereby reducing the accuracy. The heatmap in Figure 2 shows the importance of some top features for malware detection.

4.2. Model Tuning. The results of the RF model can be improved using hyperparameter tuning. Hyperparameters are classifier-specific parameters that control the learning rate during training and are set before a model is trained. Initially, default parameters were used. Then, hyperparameter tuning was carried out using GridSearchCV [41]. This is a Python library which facilitates the process of selecting the best parameters for a ML algorithm. There are also other techniques like random search but GridSearchCV

Table 4: Accuracy of malware category classification.

Features	RF (%)	DT (%)	KNN (%)	SVM (%)
Top30 ExtraTree classifier	85.13	82.55	77.47	43.53
Top30 mutual information	83.89	77.87	71.92	40.93
Top30 MinMax ExtraTree classifier	85.67	**83.48**	82.95	79.35
Top30 MinMax mutual information	84.01	77.78	79.75	60.53
Top50 ExtraTree classifier	87.06	81.78	68.28	39.91
Top50 mutual information	85.71	79.64	72.62	41.19
Top50 MinMax ExtraTree classifier	**87.24**	82.47	84.29	**81.10**
Top50 MinMax mutual information	85.70	80.35	81.69	61.48
Top100 ExtraTree classifier	86.99	81.78	72.36	41.18
Top100 mutual information	86.54	81.49	72.40	40.71
Top100 MinMax ExtraTree classifier	86.79	82.55	**84.72**	80.78
Top100 MinMax mutual information	86.81	82.35	83.70	77.30

Bold values are highest values/results of each classifier.

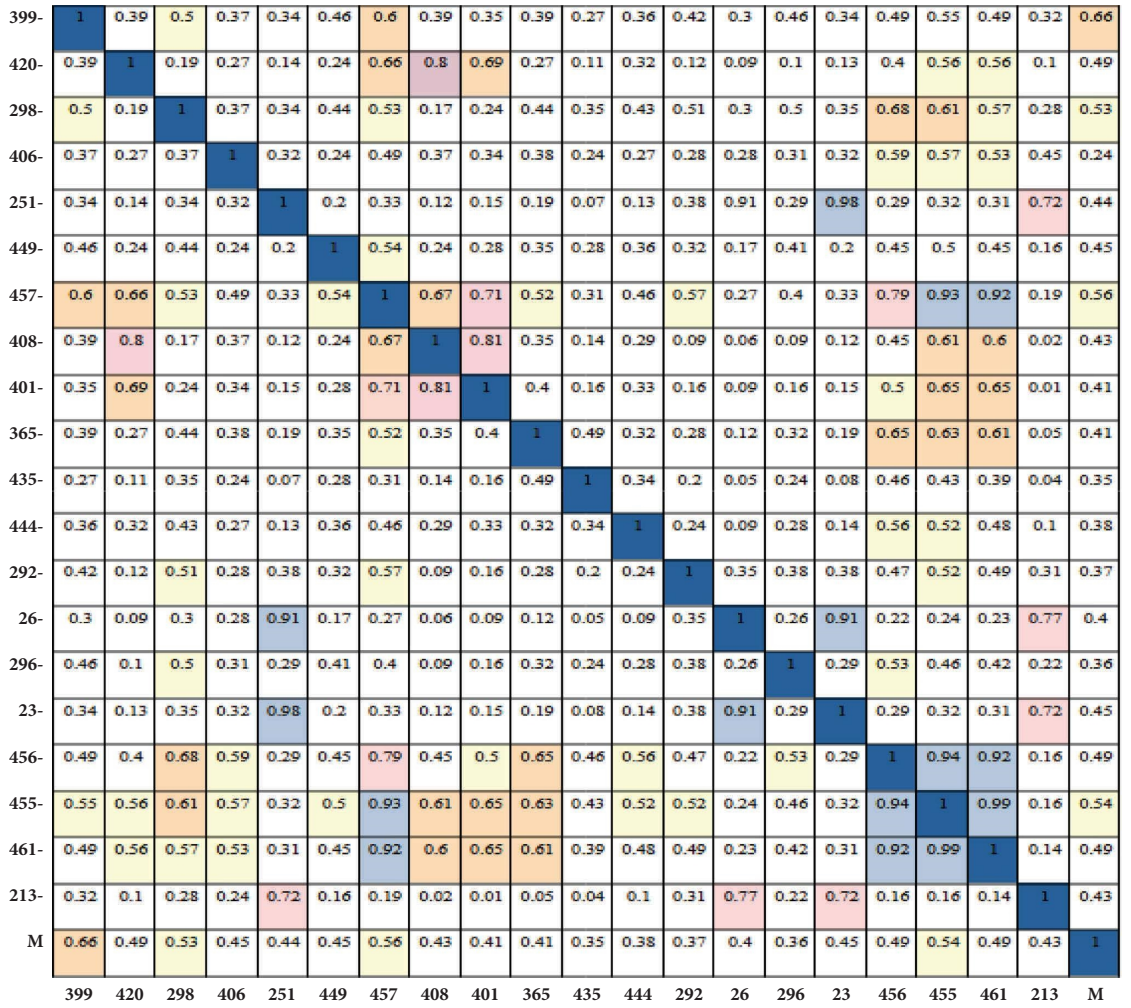

Figure 2: Heatmap of top 20 features.

implements the best technique for obtaining optimal hyperparameter values. The best hyperparameters selected by GridSearchCV for the four models are as follows:

(1) RF: (bootstrap = True, max_depth = 300, max_features = "log2")

(2) DT: (criterion = "gini," max_depth = 19)

(3) KNN: (n_neighbors = 1)

(4) SVM: (C = 20, kernel = "rbf")

The results for malware detection and category classification after hyperparameter tuning are summarised in Table 5. It can be clearly seen that RF still provides the best performance for both tasks. Its values are highlighted in bold

Table 5: Results after hyperparameter tuning.

Task	Metric	RF (%)	DT (%)	KNN (%)	SVM (%)
Malware detection	Accuracy	**98.03**	96.60	97.16	96.39
	Precision	**98.51**	97.12	97.24	97.65
	Recall	**97.73**	96.45	97.44	95.54
	*F*1-score	**98.12**	96.78	97.34	96.58
Malware category classification	Accuracy	**87.56**	83.09	84.92	84.78
	Precision	**90.14**	74.14	77.27	78.40
	Recall	**74.20**	70.87	73.67	71.34
	*F*1-score	**81.39**	72.47	75.20	74.70

Bold values are highest results/values of Random Forest classifier/detector.

text. Specifically, the accuracy of malware detection has increased to 98.03% and the accuracy of malware category classification has increased to 87.56%. The confusion matrix for RF is presented in Figure 3. The false positive rate and false negative rate are less than 0.1%.

4.3. Validation. The abovementioned models are validated using k-fold cross validation ($k = 5$) for malware detection and malware category classification. This validation process splits the dataset into 5 folds. In each iteration, $k - 1$ folds of the dataset are used for training and 1 fold is used for testing. This technique of training the model on different chunks of dataset and testing it on remaining chunk is carried out to validate the accurate performance of the model by avoiding model overfitting and underfitting. The results are summarised in Table 6 and clearly demonstrate the validity of the models.

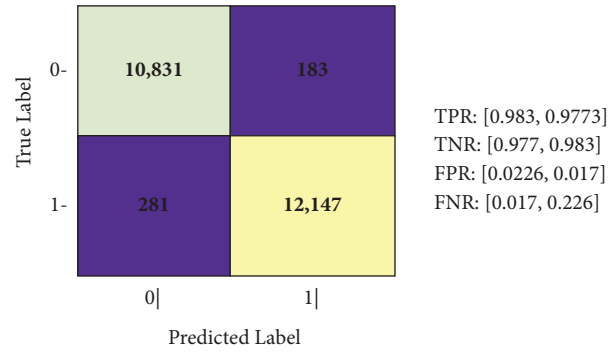

Figure 3: Confusion matrix of RF binary detection final results.

Table 6: Results for 5-fold cross validation.

	RF (%)	DT (%)	KNN (%)	SVM (%)
Malware detection	97.97	96.60	97.05	96.42
Malware category classification	87.20	82.99	84.78	84.25

5. Discussion

Multiple ML and DL solutions have been proposed and they are briefly discussed here for the purpose of comparison. The comparison is also summarised in Table 7 and the values in bold emphasizes aspects of this research that are important in comparison toexisting research.

The selection of a high-quality dataset is essential for the effectiveness of any proposed solution, especially for handling concept drift. Sustainable performance (across years) of five different ML-based malware detectors was evaluated in [47]. The results show a drastic decrease in the performance by all malware detectors including DroidSeive [45]. DroidSeive had good detection and family classification results (99.82% and 99.26%, respectively) but after its 7-year evaluation period, it had the lowest sustainable performance (with an accuracy of 34.59%).

The popularity of emulators for dynamic analysis is also a notable trend and more than half of the studies shown in Table 7 use an emulator. In [46], a dynamic malware classification technique called Droidcats is developed. The dynamic features were captured using an emulator. As pointed out by several studies, antidynamic malware can detect an emulated environment and chose not to execute malicious behaviours or intents. Therefore, any dataset compiled using an emulator will not have a comprehensive profile of such sophisticated malware. Despite this limitation, their technique achieved good results with an accuracy of 97.4% for malware detection and 97.8% for malware categorization. Another limitation of this research is that the malware samples used are more than 5 years old (samples date from 2009 to 2017).

DL-Droid [44] is one of the few models trained using data extracted from dynamic analysis carried out on a real device. The dataset used in this research consists of 30,000 samples and the developed DL model reported an accuracy of 98.5% for malware detection.

Interestingly, their list of top 20 hybrid features contains features that also are part of our top 50 feature set, e.g., RECEIVE_SMS, RECEIVE_BOOT_COMPLETED, SEND_SMS, and READ_PHONE_STATE. This shows that hybrid features are ideal for malware detection and despite malware evolution they persist.

One limitation of this research is that it does not take into account the timeframe of malware samples and is, therefore, not effective for detection of malware that evolves its behaviour with time.

Feature selection also plays a key role in training machine learning classifiers. In [48], the developed malware detection model achieved an accuracy of 96.3% when trained using only 27 features (permissions) selected by a regression-

Table 7: Related work on malware detection and category classification using different datasets.

Source	Approach and year	Name/Year of dataset	Time frame	Features and approach	Dynamic	Malware detection	Category classification
[42]	Semisupervised learning (2018)	2016	N/A	Hybrid feature selection: aggregate information	Emulator	Accuracy: 91.23%	N/A (dynamic accuracy: 80.3%)
[38]	Deep artificial Neural network (2021)	CICANDMAL2019	N/A	Hybrid feature selection: N/A	Real device	Static accuracy: 93.40%	4 categories (static accuracy: 92.5%) (dynamic accuracy: 80.3%)
[43]	Machine learning (2018)	Updroid	2014–2018	Hybrid feature selection: N/A	Emulator	Detection as categorization	Accuracy: 96.37%
[44]	Deep learning (2020)	DL-driod dataset 2019	N/A	Hybrid Feature ranking: InfoGain	Real device	Accuracy: 98.5%	N/A
[18]	Deep neural network with pseudolabel (2020)	CICMALDroid2020	2017-2018	Dynamic feature selection: N/A	Emulator	Detection as categorization	4 categories (*F*1-score: 97.8%)
[7]	Machine learning (2021)	CCS-CICAndMal2020	N/A	Dynamic feature selection: N/A	Emulator	Detection as categorization	12 categories (precision: 98.4%)
[45]	Machine learning (2017)	Drebin,Mcafee Praguard	N/A	Static feature selection: mean decrease impurity (MDI)	N/A	99.82%	99.26% into families
[46]	Machine learning (2018)	Drebin, Genome VirusShare	2009–2017	Dynamic feature selection: N/A	Emulator	97.4%	97.8% into families
[47]	Machine learning (2020)	VirusShare AndroZoo	2010–2017	Dynamic feature importance: Top100	Emulator	*F*1: 92.88% (same year) *F*1: 71.81% (across year)	N/A
[48]	Machine learning (2021)	APKPure, random dataset	N/A	Static, feature selection: LR based	N/A	Accuracy: 96.3%	N/A
[49]	Machine learning (2021)	APKPure, VirusShare	N/A	Static, feature selection: filter-based	N/A	*F*-measure: 95%	N/A
[50]	Machine learning (2022)	Mendeley repository	N/A	Dynamic, features selection: embedded BFE	Emulator	*F*-measure: 99%	N/A
[51]	Machine learning (2022)	Multiple repositories	N/A	Dynamic, feature selection: rough set analysis (RSA) and principal component analysis (PCA)	Emulator	Detection rate: 98.8%	N/A
[52]	Machine learning (2023)	AndroZoo and Drebin	N/A	Static, feature selection: wrapper based (DDQN)	N/A	Detection rate: 95.6%	N/A
[23]	Deep neural network with pseudo label stack auto encoder (2022)	CICMALDroid2020	2017-2018	Hybrid feature selection: N/A	Emulator	Accuracy: 98.28%	5 categories (precision: 98.4%)
[53]	Ensemble (random forest)	KronoDroid	2008–2020	Hybrid, features selection: Chi-squared	Real and emulated device	Accuracy: 95% Precision: 95%	N/A
Our	Machine learning (random forest)	Subset of 2020 KronoDroid	**2008–2020**	Hybrid, filter-based features selection	**Real device**	Accuracy: **98.03%**	15 categories (accuracy: **87.6%**)

The values in bold emphasizes aspects of this research that are important in comparison to existing research.

based feature selection technique. The Android malware detection solution developed by [49] achieved a *F*-measure of 95% using random forest classifier using only 20 features (permissions). These features were selected using a filter-based feature selection technique. In another study [50], BFEDroid was proposed. The researchers attained a *F*-measure of 99% using a new embedded feature selection-based detection technique called embedded BFEDroid and LSSVM (with radial basis function kernel and principal component analysis). Their solution used a dataset compiled from the Mendeley repository that consists of 5,000 subsets. It utilizes dynamic features such as permissions and API calls. In a similar study, FSDroid, a supervised machine learning malware detection model was developed by implementing LSSVM (least square support vector machine) with RBF (radial basis kernel function) [51]. The features were extracted by performing dynamic analysis on an emulator. This model achieved 98.8% detection rate using the RSA (rough set analysis) features subset selection technique and PCA (principal component analysis) for feature ranking. Lastly, in [52], a supervised machine learning malware detection model (called DroidRL) that uses a wrapper-based feature selection method was developed. The model achieved 95.6% accuracy for malware detection using the random forest classifier and a reduced subset of only 24 features from a dataset of 5,000 benign and 5,560 malware samples. None of these five studies use hybrid approach, which makes it difficult to compare their detection rate with ours. The only new study to use hybrid approach is [53]. It uses random forest (with chi-squared feature selection-based method) to achieved the best detection rate of 95%. Overall, these studies demonstrate the significant role of using suitable feature selection techniques in malware detection models.

The authors conclude that the following are essential requirements for the development of an effective ML-based malware detector [44]:

(i) A balanced dataset (with old and new samples),

(ii) Dynamic analysis on real devices, and

(iii) Carefully crafted features sets.

A similar conclusion is reached in [29]. To meet abovementioned requirements, the model developed in this paper uses the KronoDroid [31] dataset. This dataset is almost balanced (41,382 malware samples and 36,735 benign samples) and contains both dynamic and static features. The dynamic features were extracted using dynamic analysis conducted on both an emulated setup and a real device. The use of real devices means that antidynamic malware was successfully analysed. The dataset also includes samples from 2008 to 2020, making it effective against malware that changes its behaviour over time (concept drift). KronoDroid's hybrid features and the large time span of its samples make it effective for the detection of evolving malware and require minimal retraining. Furthermore, using only 50 hybrid features lowers the computational cost of retraining the model.

To the best of our knowledge, no other malware detection solution has combined and achieved the following:

(i) The ability to detect malware that can changes its behaviour when run in an emulated environment

(ii) Handling concept drift for the detection of malware that evolves with time

(iii) Accuracy of 98.03% for malware detection and 87.56% for malware category classification.

6. Conclusion

In this research, a malware detection and category classification model for advanced and evolving Android malware is developed. The model uses supervised ML and is trained using an enhanced subset of the KronoDroid dataset. The KronoDroid dataset includes malware samples from the entire history of Android and is ideal for a model that can handle concept drift. It also contains features extracted from both the static and dynamic analysis of malware. In addition, dynamic analysis is conducted by running the malware on a real device. The trained model is, therefore, effective for the detection of antidynamic malware capable of bypassing or modifying its behaviour in an emulated environment.

One shortcoming of the KronoDroid dataset is that it does not include labels for malware categories. We added malware category labels to a subset of this dataset with the help of multiple online antimalware repositories. This enhanced dataset was used to train random forest (RF), decision tree (DT), K-nearest neighbour (KNN), and support vector machine (SVM) classifiers. Also, the ExtraTree classifier and mutual information algorithms were used for feature selection. The performance of the trained models was evaluated using metrics such as accuracy, precision, *F*1-score, and recall.

The results show that the highest accuracy was obtained using RF (with Top50 MinMax features selected using the ExtraTree classifier) for both malware detection (98.03%) and malware category classification (87.56%). MinMax scaling selects the features which have highest impact on output. Initially, multiple models were trained using different subsets of top features such as top 30, top 50, and top 100. Because top 50 provided the best results, it was validated using 5-fold cross validation. The selection of optimal number of top features not only enhanced the results but also reduced the computational overhead. Compared to existing solutions, this makes our proposed model more suitable for adoption in a production environment.

To summarise, the main contributions of this paper are as follows:

(i) A subset of the KronoDroid dataset enhanced by adding malware category labels.

(ii) An effective supervised ML model (RF with Top50 MinMax ExtraTree classifier features) with 98.03% accuracy for malware detection and 87.56% accuracy for malware category classification (for 15 categories).

(iii) A comparison with related work that shows the closest work to our RF model [38] has 4.63% lower accuracy for malware detection. For malware

category classification, it only includes four categories compared to our 15 categories. Their model achieves 4.94% higher accuracy for static detection but 7.26% lower accuracy for dynamic detection. The comparison also shows that DL-based solutions like [44] have almost the same accuracy but our proposed solution has the following advantages:

(1) It is effective for detecting antidynamic and evolving malware because the dataset includes hybrid features and malware samples from almost entire timeline of the Android platform (2008–2020).
(2) It is effective for malware category classification because it uses a more comprehensive and reliable dataset with 15 categories.

In future, our enhanced dataset could be improved by adding more samples to ensure each malware category is balanced (currently, the malware categories have unequal number of samples as shown in Table 1). This should enhance the accuracy of the model for malware category classification. Also, additional steps could be taken to validate the malware category label. Around 30% of the malware samples were assigned different labels by different repositories (e.g., Trojan by one antimalware repository and Riskware by another other repository). We believe that the accuracy of malware category classification could be improved through this validation process.

Although our detection model does its best to minimize retraining requirement and computational cost (by using samples from the longest possible time span and including hybrid features for the detection of evolving and emerging malware), its effectiveness, to some extent, still depends on the stability of patterns in future. Therefore, the timeframe for retraining and the stability of the model with respect to evolving patterns should also be investigated.

Data Availability

The malware dataset (KronoDroid) used in this research work is from previously reported research work, which have been cited. The modified and improved version of data is publicly available at Github semw/kronodroid_improved_hybrid_detection_v2 in the form of csv files.

Disclosure

The authors conducted this research while affiliated with the School of Electrical Engineering and Computer Science (SEECS) at the National University of Sciences and Technology (NUST). This work was completed as part of Master's (MS) thesis at National University of Sciences and Technology (NUST) and is not part of any funded research project.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] Zimperium, "2023 global mobile threat report-zimperium," 2023, https://www.zimperium.com/global-mobile-threat-report/.
[2] F. Laricchia, "Mobile OS marketshares 2009-2022," 2022, https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.
[3] Statista Research Department, "Development of Android malware worldwide 2016-2020," 2022, https://www.statista.com/statistics/680705/global-android-malware-volume/.
[4] Google, "Google play protect," 2022, https://support.google.com/googleplay/answer/2812853.
[5] Source android, "Application security," https://source.android.com/docs/security/overview/app-security.
[6] E. Masabo, K. S. Kaawaase, J. Sansa-Otim, J. Ngubiri, and D. Hanyurwimfura, "A state of the art survey on polymorphic malware analysis and detection techniques," *ICTACT Journal On Soft Computing*, vol. 8, 2018.
[7] D. S. Keyes, B. Li, G. Kaur, A. H. Lashkari, F. Gagnon, and F. Massicotte, "Entroplyzer: android malware classification and characterization using entropy analysis of dynamic characteristics," in *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, pp. 1–12, IEEE, Hamilton, Canada, May 2021.
[8] S. Nielebock, P. Blockhaus, J. Krüger, and F. Ortmeier, "Androidcompass: a dataset of android compatibility checks in code repositories," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pp. 535–539, IEEE, Madrid, Spain, May 2021.
[9] D. Hu, Z. Ma, X. Zhang, P. Li, D. Ye, and B. Ling, "The concept drift problem in android malware detection and its solution," *Security and Communication Networks*, vol. 2017, Article ID 4956386, 13 pages, 2017.
[10] X. Fu and H. Cai, "On the deterioration of learning-based malware detectors for android," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 272-273, IEEE, Montreal, Canada, May 2019.
[11] W. Li, X. Fu, and H. Cai, "Androct: ten years of app call traces in android," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pp. 570–574, IEEE, Madrid, Spain, May 2021.
[12] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm, "KronoDroid: time-based hybrid-featured dataset for effective android malware detection and characterization," *Computers and Security*, vol. 110, Article ID 102399, 2021.
[13] H. Bai, N. Xie, X. Di, and Q. Ye, "Famd: a fast multifeature android malware detection framework, design, and implementation," *IEEE Access*, vol. 8, pp. 194729–194740, 2020.
[14] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, "Didroid: android malware classification and characterization using deep image learning," in *2020 The 10th international conference on communication and network security*, pp. 70–82, Tokyo, Japan, November 2020.
[15] A. Arora, S. K. Peddoju, and M. Conti, "Permpair: android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968–1982, 2020.
[16] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "Malpat: mining patterns of malicious and benign android apps via permission-related apis," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 355–369, 2018.

[17] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 400–412, 2015.

[18] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 515–522, IEEE, Calgary, Canada, August 2020.

[19] P. Lantz, "An android application sandbox for dynamic analysis," *Master, Lectrical and Information Technology*, Lund university, Lund, Sweden, 2011.

[20] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: android malware detection through manifest and api calls tracing," in *2012 Seventh Asia joint conference on information security*, pp. 62–69, IEEE, Calgary, Canada, August 2012.

[21] Dunkelheit, "AMAT-android malware analysis toolkit," 2022, http://dunkelheit.com.br/amat/index.html.

[22] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *International conference on detection of intrusions and malware, and vulnerability assessment*, pp. 252–276, Springer, Berlin, Germany, July 2017.

[23] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, "Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder," *Journal of Network and Systems Management*, vol. 30, no. 1, pp. 22–34, 2022.

[24] A. I. Ali-Gombe, B. Saltaformaggio, J. R. Ramanujam, D. Xu, and G. G. Richard, "Toward a more dependable hybrid analysis of android malware using aspect-oriented programming," *Computers and Security*, vol. 73, pp. 235–248, 2018.

[25] R. Surendran, T. Thomas, and S. Emmanuel, "A tan based hybrid model for android malware detection," *Journal of Information Security and Applications*, vol. 54, Article ID 102483, 2020.

[26] H. Cai, "Embracing mobile app evolution via continuous ecosystem mining and characterization," in *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, pp. 31–35, Seoul, Republic of Korea, July 2020.

[27] H. Cai and B. Ryder, "A longitudinal study of application structure and behaviors in android," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2934–2955, 2021.

[28] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: self-evolving android malware detection system," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 47–62, IEEE, Stockholm, Sweden, June 2019.

[29] G. Suarez-Tangil and G. Stringhini, "Eight years of rider measurement in the android malware ecosystem," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 107–118, 2022.

[30] H. Cai, X. Fu, and A. Hamou-Lhadj, "A study of run-time behavioral evolution of benign versus malicious apps in android," *Information and Software Technology*, vol. 122, Article ID 106291, 2020.

[31] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm, "Kronodroid: time-based hybrid-featured dataset for effective android malware detection and characterization," *Computers & Security*, vol. 110, Article ID 102399, 2021.

[32] Pandas, "Pandas-Python data analysis library," 2022, https://pandas.pydata.org/.

[33] Virus Total, "Virus total," 2022, https://www.virustotal.com/gui/home/search.

[34] F-Secure, "Total protection for your life online," https://www.f-secure.com/en.

[35] Fortiguard Labs, "Fortiguard," 2022, https://www.fortiguard.com/search?q=TrojanSMS.Stealer.

[36] M. Waheed and S. Qadir, "Kronodroid improved dataset," 2022, https://github.com/semw/kronodroid_improved_hybrid_detection_v2.git.

[37] Scikit-Learn, "Sklearn.preprocessing.MinMaxScaler," 2022, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[38] S. I. Imtiaz, S. U. Rehman, A. R. Javed, Z. Jalil, X. Liu, and W. S. Alnumay, "Deepamd: detection and identification of android malware using high-efficient deep artificial neural network," *Future Generation Computer Systems*, vol. 115, pp. 844–856, 2021.

[39] S. Biswas, M. Bordoloi, and B. Purkayastha, "Review on feature selection and classification using neuro-fuzzy approaches," *International Journal of Applied Evolutionary Computation*, vol. 7, no. 4, pp. 28–44, 2016.

[40] A. H. E. Fiky, A. E. Shenawy, and M. A. Madkour, "Android malware category and family detection and identification using machine learning," 2021, https://arxiv.org/abs/2107.01927.

[41] Scikit-Learn, "Sklearn GridSearchCV," https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[42] A. Atzeni, F. Diaz, A. Marcelli, A. Sánchez, G. Squillero, and A. Tonda, "Countering android malware: a scalable semi-supervised approach for family-signature generation," *IEEE Access*, vol. 6, pp. 59540–59556, 2018.

[43] K. Aktas and S. Sen, "Updroid: updated android malware and its familial classification," in *Nordic Conference on Secure IT Systems*, pp. 352–368, Springer, Berlin, Germany, August 2018.

[44] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Dl-droid: deep learning based android malware detection using real devices," *Computers and Security*, vol. 89, Article ID 101663, 2020.

[45] 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 *Proceedings of the seventh ACM on conference on data and application security and privacy*, pp. 309–320, Scottsdale, AZ, USA, March 2017.

[46] H. Cai, N. Meng, B. Ryder, and D. Yao, "Droidcat: effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2019.

[47] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *ACM Transactions on Software Engineering and Methodology*, vol. 29, no. 2, pp. 1–28, 2020.

[48] D. Ö. Şahin, O. E. Kural, S. Akleylek, and E. Kılıç, "A novel permission-based android malware detection system using feature selection based on linear regression," *Neural Computing & Applications*, vol. 35, no. 7, pp. 4903–4918, 2023.

[49] D. Ö. Şahin, O. E. Kural, S. Akleylek, and E. Kılıç, "A novel android malware detection system: adaption of filter-based feature selection methods," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 2, pp. 1243–1257, 2023.

[50] C. Chimeleze, N. Jamil, R. Ismail et al., "Bfedroid: a feature selection technique to detect malware in android apps using machine learning," *Security and Communication Networks*, vol. 2022, Article ID 5339926, 24 pages, 2022.

[51] A. Mahindru and A. L. Sangal, "Fsdroid:-a feature selection technique to detect malware from android using machine learning techniques: fsdroid," *Multimedia Tools and Applications*, vol. 80, no. 9, pp. 13271–13323, 2021.

[52] Y. Wu, M. Li, Q. Zeng et al., "Droidrl: feature selection for android malware detection with reinforcement learning," *Computers and Security*, vol. 128, Article ID 103126, 2023.

[53] S. Aurangzeb and M. Aleem, "Evaluation and classification of obfuscated android malware through deep learning using ensemble voting mechanism," *Scientific Reports*, vol. 13, pp. 3093–3111, 2023.