FG-Droid: Grouping based feature size reduction for Android malware detection

Recep Sinan Arslan [Corresp. 1]

[1] Department of Computer Engineering, Kayseri University, Kayseri, Turkey

Corresponding Author: Recep Sinan Arslan
Email address: sinanarslanemail@gmail.com

Background. The number of applications prepared for use on mobile devices has increased rapidly with the widespread use of the Android OS. This has resulted in the undesired installation of Android application packages (APKs) that violate user privacy or are malicious. The increasing similarity between Android malware and benign applications makes it difficult to distinguish them from each other and causes a situation of concern for users. **Methods**. In this study, FG-Droid, a machine-learning based classifier, using the method of grouping the features obtained by static analysis, was proposed. It was created because of experiments with Machine learning (ML), DNN, RNN, LSTM, and GRU-based models using Drebin, Genome, and Arslan datasets. **Results**. The experimental results revealed that FG-Droid achieved a 97.7% area under the receiver operating characteristic (ROC) curve (AUC) score with a vector including only 11 static features and the ExtraTree algorithm. While reaching a high classification rate, only 0.063 seconds were needed for analysis per application. This means that the proposed feature selection method is faster than all traditional feature selection methods, and FG-Droid is one of the tools to date with the shortest analysis time per application. As a result, an efficient classifier with few features, low analysis time and high classification success was developed thanks to a unique feature grouping method.

1   **FG-Droid: Grouping based feature size reduction for Android malware detection**

3   Recep Sinan ARSLAN[1]

5   [1] Department of Computer Engineering, Kayseri University, Kayseri, Turkey

7   Corresponding Author:

8   Recep Sinan ARSLAN[1]

9   Kayseri University, Kayseri, 38000, Turkey

10   Email address: sinanarslanemail@gmail.com

12 Abstract

13   **Background**. The number of applications prepared for use on mobile devices has increased rapidly with
14   the widespread use of the Android OS. This has resulted in the undesired installation of Android
15   application packages (APKs) that violate user privacy or are malicious. The increasing similarity between
16   Android malware and benign applications makes it difficult to distinguish them from each other and
17   causes a situation of concern for users.
18   **Methods**. In this study, FG-Droid, a machine-learning based classifier, using the method of grouping the
19   features obtained by static analysis, was proposed. It was created because of experiments with Machine
20   learning (ML), DNN, RNN, LSTM, and GRU-based models using Drebin, Genome, and Arslan datasets.
21   **Results**. The experimental results revealed that FG-Droid achieved a 97.7% area under the receiver
22   operating characteristic (ROC) curve (AUC) score with a vector including only 11 static features and the
23   ExtraTree algorithm. While reaching a high classification rate, only 0.063 seconds were needed for
24   analysis per application. This means that the proposed feature selection method is faster than all
25   traditional feature selection methods, and FG-Droid is one of the tools to date with the shortest analysis
26   time per application. As a result, an efficient classifier with few features, low analysis time and high
27   classification success was developed thanks to a unique feature grouping method.

29   **Keywords**: Permission grouping, size reduction, Android OS, security, malware detection

31 Introduction

32   Android OS is a mobile platform that was prepared by a group of developers. It has dominated the mobile
33   operating system market for many years. According to 2021 statistics, it is used in more than 70% of
34   mobile devices, and together with the iOS operating system, it meets 98% of the entire market share [53].
35   Accordingly, the number of application downloads worldwide is increasing rapidly and this trend is
36   expected to continue [61]. There are many reasons for this widespread use of the Android operating
37   system. It has the appropriate functional infrastructure to access hardware resources. It is free and open
38   source platform and is equipped with a security framework that relies on the Linux kernel [1]. However,
39   since its security structure is based on the application layer [5], these devices become partially or
40   completely exposed to numerous security attacks, making them a routine target [4]. In addition, the
41   widespread use of mobile devices with Android OS, the adoption of these devices by end users and the
42   resulting of increasing market share, has caused it to become the target of cyber hackers, especially web-
43   based and application layer-based attacks [2, 3].
44   When malicious applications access user mobile devices, they can engage in a series of malicious
45   activities, such as obtaining confidential information, seizing more authorized user accounts, and

46 misusing the obtained certain level of security [4]. Google Play has set up a permission-based system to
47 control applications' access to confidential data. Users are asked to give permission before installation,
48 taking into account the resources of the application. Users must approve these permissions before
49 installation to use it. However, this-preapproval mechanism does not provide sufficient protection for
50 users, since users accept these permissions without detailed examination. As a result, users accept all
51 conditions in order to have free access to applications that offer the features they demand [6]. For these
52 reasons, there is a need to work on the security mechanism that Google Play provides for users.
53 Different types of malware detection mechanisms have been proposed to address this need in the security
54 mechanism. These are the signature-based approach [7, 8], behavior-based detection software [9–11], and
55 machine learning-based approaches [12–14]. Among these, machine learning and deep learning
56 architectures have gained more popularity recently. Because in this method, it is possible to obtain both a
57 dynamic learning and development process and good results against zero-day attacks. It is possible to
58 create ML models that are open to learning and development at the same speed for cyber attackers who
59 try to overcome the security mechanism with a new technique every day, and to produce promising
60 solutions for the detection of malware [15]. In machine learning based approaches, there is a dependency
61 on the features used as input, classifier and learning architectures. In this study, the size of the feature set
62 and the effects of the features it contains in terms of performance and efficiently in android malware
63 detection mechanisms are focused.
64 There are many manual and automatic feature extraction methods. These methods are basically divided
65 into 3 groups, as static features, dynamic features, and hybrid features [16]. Each of these features can be
66 used to detect different malicious activities. In addition, each feature can provide a distinctiveness in
67 detecting malicious applications. For this reason, different feature sets can be used in studies in this field,
68 and problem-specific feature vectors can be produced. The main purpose is to provide fastest and highest
69 classification success model with the best feature set.
70 The contributions of this work in the following way:
71 • Feature grouping based Android malware detection tool(FG-Droid) is a low runtime and highly
72 efficient machine learning model.
73 • The model groups permission-based features with a unique methodology and obtains only 11
74 static features for each application.
75 • The model has 97.7% classification success in the tests and only needs 0.063 seconds for analysis
76 per application. This value is one of the best values among the models with similar classification
77 success. This value was obtained without using the GPU.
78 • The model selects fewer features than traditional feature selection methods (chi2, f_class_if,
79 PCA) and requires less processing time while showing higher classification success than them.
80 • As a result, a model with high classification success and low analysis time with few features has
81 been revealed thanks to the proposed unique grouping.
82
83 The continuation of this study is organized as follows: In Section 2, Android application development
84 infrastructure and similar studies on this subject are analyzed separately for static and dynamic methos.
85 The methodology of FG-Droid is explained in detail in Section 3 and the experimental results are given
86 comparatively in Section 4. In the last part, a general evaluation of the study is made and suggestions are
87 made for future studies. In addition, a FG-Droid permission grouping table is given as Appendix-A before
88 the reference section.
89

90 **Materials & Methods**

91

92 **Literature Review**

93 In this section, recent studies related with Android malware detection, feature generation and selection,
94 and static, dynamic, and hybrid approaches are discussed.

95 **Static Analysis**

96 The features extracted as a result of the static analysis are basically obtained from the
97 Androidmanifest.xml in the APK package SMALI files. It is intended to extract features without actually
98 running applications and use them for classification. Permissions, application programming interface
99 (API) calls, intent filters, application metadata, function calls, and opcode are some of these features.
100 Among these, permission and API calls are more commonly used.

101 Permissions are one of the most researched and widely used features in malware analysis because they
102 form the basis of the android security architecture. Android applications are installed on mobile devices
103 after approval from the users. Since permission is the first obstacle for cyber hackers to reach their
104 malicious targets, many researchers [17–19] have carried out permission-based analysis studies. Arp et al.
105 [55] evaluated the permissions, API calls, hardware components, and intents in the application manifest
106 file together in their study in 2014. They classified the obtained static feature vector with support vector
107 machine (SVM). APK Auditor [54] is static analysis tool running on a central server with signature
108 database based on application analysis. When the system was tested with 6909 samples, it was able to
109 detect malware with 88% accuracy. The specificity value was 0.925. Syrris et al. [24] examined malware
110 detection based on machine learning and static analysis. The six best-known ML techniques were
111 evaluated in terms of both the classification rate and feature selection. In the experimental results, it was
112 shown that a high accuracy rate could be achieved with a lower dimensional feature set, as in FG-Droid.
113 Anastasia [25] is a classifier using machine learning (decision tree (DT), random forest (RF), SVM, etc.)
114 and a deep neural network. The static analysis tool was created to extract features such as intent, system
115 commands, permissions, and API calls. In the tests performed with a dataset consisting of 11,187 benign
116 and 18,677 malicious applications, a TP value of 97.3% was obtained and the f-score was 96.0%.
117 Droidmat [27] is another tool that uses permissions, intents, API calls, and services to classify android
118 APKs. Analysis was performed using the K-neighbors algorithm. As a result, 91.82% f1 score was
119 obtained. Drebin [29] works with a large dimensional static feature vector, since limited hardware
120 resources pose problems for dynamic analysis. The extracted feature vector was trained with SVM and a
121 classification success of 94% was achieved. It was stated that an average of 10 seconds was required for
122 the analysis.

123 API calls is another feature that can be obtained as a results of static analysis. Because applications need
124 API callas to communicate with the device. Therefore, API calls can be useful to understand the intent of
125 applications. Many researchers [20–23] have targeted malware detection using API calls. High
126 classification success has been achieved in studies with API calls, but there are approximately 32,000
127 different APIs on the Android platform [15]. This causes the feature vector to be very large size. In
128 addition, a very limited part of these features were used by applications. DroidAPIMiner [26] uses
129 package-level parameters in the feature vector to catch malicious API calls. As a result, a 2.2% FP rate
130 was achieved. MAMADroid [28], which analyzes the API usage behavior of applications, is a more
131 robust system for malware detection. It was reported that the model has shown a high success rate in tests
132 for many years. Taheri et al. [42] similarly used the feature vector produced by using API calls, intents,
133 and permission from training and prediction phases for 4 different classifiers. Similarity calculation was

134 made using Hamming distance and an accuracy value of 91% was achieved. The static analysis method
135 has been used in many other studies, such as DroidSieve [40] and DroidDet [41].
136 Although, static analysis is advantageous at certain points, is also has some limitations [33]. It would not
137 be possible to observe its behavior at runtime. Code analysis of complex software takes time. It may not
138 be possible to extract static features based on the source code inn encrypted and obfuscated applications.
139 A summary of all studies using the static analysis methodology used in this study is shown in Table-1.
140
141 **Dynamic analysis**
142 Dynamic features are extracted as a result of analyzing the situations in which Android applications
143 communicate with the operating systems or the network. System calls and network usage statistics are the
144 most basic features that give an idea about the APKs. In addition, processor and memory space usage
145 data, the status of the services running instantaneously, some statistical data about the device (battery
146 usage, screen-on time, etc.), and information about the addresses reached by the systems calls and
147 network packets are important features for classification. Droidscope [30] is an efficient and effective
148 dynamic android malware detection tool that works on Android devices by extracting three-layer
149 (hardware, operating system, and dalvik virtual machine) system calls, and performing semantic analysis
150 at the operating system and code level. MADAM [31] tracks kernel-level system calls and user-level
151 usage statistics and activities to be able to describe and classify the behavior of a mobile application.
152 ANDLANTIS [35] is a dynamic analysis tool that runs on a sandbox. It aims to detect malware by
153 analyzing system calls, footprints, and running behaviors. It requires 1 h for the analysis of 3000
154 applications. Chen et al. [39] proposed a semi-supervised classifier that works using dynamic API usage
155 logs. They made use of both labeled and unlabeled data to obtain application properties. They showed the
156 results comparatively by classifying with SVM and k-nearest neighbor (KNN). TaintDroid [62] is a
157 dynamic tracer tool. It uses Dalvik virtual machine to do this tracking. It tracks the usage of sensitive
158 resources such as the location, microphone, and camera. While revealing the traces of the applications due
159 to the virtual machine, it does not pose any danger on the real environment. This ensures that data leaks
160 are prevented and malicious software intentions are revealed.
161 Dynamic analysis-based approaches try to understand the intentions of mobile applications by analyzing
162 their behavior during running. For this reason, in some cases, it can show higher recognition success than
163 static analysis. However, in order to understand a suspicious behavior, it must be run at least once and
164 information must be collected during this time. This means both creating a security problem for the
165 device and additional processing time [32]. The presence of processor and memory limitations of mobile
166 devices complicates the applicability of dynamic analysis. However, running applications on an
167 emulator/sandbox may be sufficient for dynamic analysis. It was accepted that it is possible to gather
168 more information about the application by simulating the so-called user behavior [34]. Crowdroid [56] is
169 a behavior-based android malware detection tool that works in client-server architecture. All system calls
170 from the application are collected via the mobile device and sent to a cloud server for analysis. With K-
171 means, this data is processed and the application is classified.
172 A single feature can capture certain aspects of an application. However, using more than one feature
173 together can be more advantageous in malware detection. Various studies have been conducted with
174 hybrid feature structures using combinations of both static and dynamic features [36–38]. ProfileDroid
175 [43] evaluates both static and dynamic features such as Android permissions, features obtained as a result
176 of code analysis, and network usage statistics. Thus, a systematic study was aimed to establish a cost-
177 effective and consistent model.

178   **Hybrid Analysis**
179   The features obtained because of static and dynamic analysis alone can capture only one aspect of the
180   applications. For this reason, it is possible to make a more accurate analysis when more than one feature
181   group is used together. Thus, it is possible to detect malware with higher accuracy. NTPDroid [57] uses a
182   hybrid feature vector that combines permissions and network traffic. The FP-Growth algorithm is
183   proposed to obtain the commonly used thicknesses among applications. Experimental results showed that
184   it has an accuracy value of 94.25%. Arshad [58] proposed a model for Android malware detection using
185   static and dynamic analysis together. The model first extracts the requested permissions, used
186   permissions, application components and suspicious API calls by examining the APK file. Then the
187   application's network usage statistics obtain system calls as features. By combining these two feature
188   groups, the feature vector of the application is obtained and classified by SVM. Experimental results
189   showed high malware detection accuracy. OmniDroid [59] is a hybrid feature vector and machine
190   learning classification tool that combines static and dynamic features based on voting. In this study, the
191   dependencies of static and dynamic features are taken into account and the features go through an
192   evaluation mechanism when combining them into a single vector. AAsandbox [63] offers a two-stage
193   analysis approach. First, an image of the application is taken in offline mode and static and dynamic
194   analysis is performed on the sandbox. The entire system is hosted on the cloud server and suspicious
195   applications are detected. Tong et al. [64] proposed a different approach as a solution to the long
196   processing time problem of static analysis and the problem of dynamic analysis to consume a lot of
197   processing resources. The model can detect different types of malware much more efficiently than other
198   studies and achieves an average of 90% classification success.
199   In the hybrid analysis process, there are models that use static and dynamic features together while
200   creating the feature vector, and there are studies in which hybrid classifiers are used to classify a single
201   feature type. Yerima et al. [60] proposed a composite classification model to increase classification
202   accuracy. In the composition classifier, rule-based, function-based, tree-based, and probability-based
203   classifiers are used together. Thus, an average of 5% increase in success was achieved.
204   **Methodology**
205   In this section, the method used by the FG-Droid tool was explained in detail. The method includes the
206   stages of extracting features, grouping extracted features, updating feature values, and testing on sample
207   dataset.
208   **Android Application Structure**
209   Applications prepared for the Android OS are presented to users as a kind of compressed file with the
210   APK extension. The package includes files such as source code, manifest.XML, libraries, resources, DEX
211   file, and properties, as shown in Figure 1. Applications are developed in Java using the Android SDK.
212   Applications whose source code is completed are converted to Dalvik bytecode (DEX) together with
213   other required files. A manifest file is an XML that contains basic cookies for the applications, links to
214   external files, and libraries such as activities, receivers, and content providers. In addition, the
215   permissions needed to access device resources, target platform data are included in the manifest. External
216   resources such as videos, sounds, audios, images, and text files used by the application are packaged in
217   the APK file. As a result, a package is prepared that contains necessary files to execute the entire working
218   functions [51]. Prepared APK packages are offered to users via Google Play Store or third party
219   application distribution platforms. More details for Android application development are provided on the
220   developer page [50].
221   **Automatic Feature Extraction and Pre-processing**

222

223   The Android operating system works on an application-based. After the applications are prepared, they
224   are made into zip files, and their extensions are determined as APK. These applications, which have
225   similar file structures, can be run on the same operating system (Android OS). The application package
226   contains folders and files such as androidmanifest.xml and class.dex, resources.arsc, lib, res.
227   Androidmanifest.xml is required for applications to run and contains some information (version, API
228   level, hardware, software information etc.) and permissions declared by the application.

229

230   This study adopts a static analysis-based feature extraction approach for Android malware detection. A
231   series of steps were applied to extract the permission-based features in the Androidmanifest.xml file, as
232   shown in Figure 2. First, a dataset consisting of benign and malicious software was created. Applications
233   are opened using the Jadx decompiler. Thus, access to both the source code and the Androidmanifest.xml
234   file needed in this study was provided. A list of all permissions in the Androidmanifest.xml file is
235   required to determine the permissions requested by the applications and transfer them to the feature
236   vector. This study used a list consisting of 348 permissions in Android 11 API level 30.
237   The permissions in the manifest file and the permissions in the permission list were compared and
238   transferred to the feature vector with a value of 1 for used ones and 0 for unused ones. Thus, a feature
239   vector of 1x349 dimensions was obtained for each application. It should be noted that it is impossible to
240   open all applications healthily with reverse engineering. In this case, values such as NaN and space in the
241   vector may be present. These were checked, and related applications were filtered. There was no problem
242   in terms of malicious application labelling applications. However, it is critical to determine whether
243   benign apps are genuinely benign. For this reason, all benign applications were rechecked on Virustotal.
244   For this reason, high-speed and efficient feature extraction and preprocessing were carried out. Those who
245   did well in these tests were marked with B, and applications from the Drebin and genome dataset were
246   marked M. After these processes were completed for all applications, they were combined into a single
247   feature vector. As a result, a feature vector of 7266x349 dimensions was obtained. The resulting vector
248   was saved in a CSV file and converted into a usable form in the following steps. All these processes were
249   done with the exe file prepared in c# language.

250

251   Considering that there will be a need for high processing power in this large-sized vector in machine
252   learning models, it is subjected to feature grouping, the details of which we have given in the next section.
253   Since the values after the group are greater than 1, normalization is performed with the normal
254   distribution, and all values are scaled to the 0-1 range.

255

256   **Proposed Feature Grouping and Feature Selection Algorithm**
257   The mechanism of accessing certain components or performing functions was based on permissions in the
258   Android operating system. Android applications request permission to access and read information about
259   calendar, location information, contacts, storage, camera, microphone, various sensors. For example, it is
260   possible to access contact information on the device with the Android.permission.READ_CONTACT
261   permission. Permissions vary according to the device's feature and functional capacity.
262   In this study, instead of a feature vector containing all permissions used in Android architecture, a model
263   called FG-Droid has been developed, which can achieve high classification success with a very fast
264   training and testing time by using fewer features. For this purpose, a series of operations were carried out
265   within the flow chart shown in Figure-3.

266   In the first stage of the model, a dataset containing 7622 applications was created, the details of which are
267   given in Section 4. The permission-based features of these applications have been extracted. At this stage,
268   a training and test vector containing 349 features was created. In order to shorten the training and testing
269   times and to classify using less processing time, operations were carried out according to the proposed
270   algorithm for grouping the features and reduction the size. Thus, a tool was created in which it was
271   possible to classify with less features. The standardization process was performed using the normal
272   distribution on the feature vector obtained after this step. The resulting data set was divided into two
273   separate parts to be used in the training and testing phases. During the training phase, oversampling was
274   performed using the SVM-synthetic minority over-sampling technique (SMOTE) algorithm to resolve the
275   sample number imbalance between clusters, and training was carried out on deep neural network (DNN),
276   recurrent neural network (RNN), long short-term memory (LSTM), and gated recurrent unit (GRU)
277   networks together with machine learning algorithms (Rf, Extratree, gradient boosting, etc.). The resulting
278   trained models were tested with previously separated test data and the results were shown in comparison
279   with commonly used metrics.
280   As can be seen in Figure 3, the proposed feature grouping and selection method was carried out after the
281   application features are extracted. Thus, it was possible to work with a much lower-dimensional feature
282   vector not only in the training and testing phases, but also in all processing steps. The details of feature
283   grouping operations are as shown in Figure 4.
284   The determination of groups was based on basic read/write operations (CRUD) in computer systems and
285   operations specific to mobile devices (Broadcast, Control, Bind). These groups consisted of Access (A),
286   Modify (M), Set (S), Update (U), Write (W), Read (R), Get (G), Manage (Mn), Bind (Bd), Broadcast (B)
287   and Control (C). The Android permissions in each group are shown in Appendix A. Within the groups
288   determined in Appendix A, all of the features were first scanned and brought together on a group basis.
289   The values of these features were aggregated within each group, so that the existence of the permission
290   represents itself in the total. For example, instead of using 21 different features in model trainings
291   separately, they were combined under the Access (A) feature. Thus, 21 properties were represented only
292   by the Access (A) property. Although there are many permissions/features in the Android operating
293   system, very few of them were used in applications, so it is not necessary to use all of the features
294   separately during training and testing. In the Drebin dataset used in this study, the average number of
295   permissions requested for each application was 5. Similarly, the average number of permit requests in the
296   Genome and Arslan datasets was 6 and 5, respectively. In this case, a feature vector created with 5
297   numbers 1 and 344 numbers 0. Instead, all of the permissions were searched for each application, and
298   they were combined under 11 feature groups after grouping and aggregation. Thus, in the process starting
299   as 349 features, a feature vector with only 11 features was obtained for each application.
300   The feature grouping process was carried out using the code structure given in Figure 5. The feature
301   vector of 7622×349 dimensions taken as input was converted into a 7622×11 dimensional vector, thus
302   providing a much more efficient learning in training and testing processes. The effect of the obtained
303   vector on the results is shown in detail in Section 4.
304
Results
306   The performance of the proposed model is evaluated in terms of 1) the classification success and 2) the
307   time needed for training and prediction. A binary classification was determined as benign and malicious.
308   The tests were performed separately on machine learning techniques (RF, DT, LDA, etc.), DNN, RNN,

309 LSTM, and GRU networks, and the results were given in detail. Thus, it was desired to show the effect of
310 feature grouping and reduction process used by FG-Droid application on the results.
311

312 **Evaluation Metrics and Experimental setup**
313 The experimental environment and evaluation criteria are very important to analyze the performance of
314 the machine-learning model. For the experiment, 70% of the data was split for training and 30% for
315 testing. area under the receiver operating characteristic (ROC) curve (AUC), precision, recall, f-score, and
316 accuracy values were calculated to evaluate the performance of the proposed model. The calculation
317 equations of these metrics are shown below:

318 $$precision = \frac{TP}{TP + FP} \qquad (1)$$

319 $$recall = \frac{TP}{TP + FN} \qquad (2)$$

320 $$f - score = \frac{\frac{TP}{TP + FN} * \frac{TP}{TP + FP}}{\frac{TP}{TP + FN} + \frac{TP}{TP + FP}} \qquad (3)$$

321 $$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (4)$$

322

323 TP is the number of truly malicious samples from those predicted as malware, FP is the number of
324 samples that are predicted to be malware that are not truly malicious. FN is the number of samples that
325 are predicted to be benign that are not truly benign and TN is the number of truly benign samples from
326 those predicted as benign.
327 The computer system architecture used in the development of the FG-Droid tool is as shown in Table 2. It
328 is very important in calculations regarding training and test times. All comparative results were obtained
329 using the same infrastructure.
330 **Dataset**
331 In studies on Android malware detection, it is not possible to reach sufficiently large, homogeneously
332 distributed, and reliable datasets. In this study, Drebin [44] and Genome [45] datasets were used for
333 malware applications. The Drebin dataset contains 5560 samples from 179 different application groups. A
334 malicious dataset with 6660 samples was created by taking 1000 samples from the genome dataset.
335 Arslan [52] dataset was used for benign applications. In this dataset, the applications with the highest
336 number of downloads in Google Playstore were selected in various categories, and they were subjected to
337 security tests on virustotal.com [46] and a data set consisting of 960 applications that passed the test was
338 created. During the labeling phase of the datasets, Drebin and Genome were distributed as labeled, so no
339 action was taken on the malicious dataset. However, labeling in the benign dataset was made by us to be
340 used in this study. As a result, a dataset containing real-world applications and examples that will not
341 create noise for both malicious and benign applications has been created.
342 **Hyper-parameter tuning for best machine learning models**
343 In order to achieve successful classification performance in a test environment where each sample is
344 represented by 11 features, 10 different machine learning techniques and DNN, RNN, LSTM, and GRU
345 networks were designed and tested. At the end of these processes, hyper-parameter tuning was performed
346 for the all classifiers.. Both GridSearch and RandomSearch algorithms were used for the selection of the
347 best parameters, and the selection range of the parameters and selected parameters was as shown in Table
348 3. Thus, it was possible to obtain the best rates for all classifiers in the tests.
349
350

Results for Machine Learning

351
352 After the feature-grouping algorithm used in the development process of the FG-Droid, the results
353 obtained in the tests using machine-learning techniques were as shown in Table 4 for 10 different
354 classifiers.

355 As can be seen in Table 4, the accuracy rate is 90% and above, except for two algorithms, and the highest
356 classification rate was 92.5%. While a successful result such as 94.4% was obtained in the precision
357 value, recall, and f-score were 92%. As a result, a very high success of 97.9% was achieved in the AUC
358 score, which is the indicator of classification success for both classes. This showed that it is not enough
359 for permission-based applications to remove the disruptive features in android malware detection and it is
360 not necessary to consider each permission as a separate feature. It was possible to reach the 97.9%
361 success level using only 11 features of the application. The effect of the FG-Droid tool is not in the
362 feature selection, but in grouping the features and ensuring that each permission is represented under the
363 group. It was possible to prevent the loss of the distinctiveness of the feature by selecting the feature.
364 Thus, both the number of features were greatly reduced and the effect of many features on classification
365 was used under the group.

366

367 In order to improve the results, the cross-validation process on the model was performed using
368 "RepeatedStratifiedKFold" function. The n_splits, n_repeats and random state were chosen as 10, 3, and
369 123, respectively. The results obtained in repeated tests for all classifiers are shown in Figure 6.
370 Accordingly, XGB, ET, RF and DT algorithms are more successful than other classifiers at the average
371 classification rate.

372

Results for Deep Learning Models

373
374 During the development of the FG-Droid, tests were carried out using deep learning models. As stated
375 before, the number of features decreases considerably with the proposed grouping algorithm. The effect
376 of the algorithm in deep learning models, which need to use more features and more training examples in
377 its basic structure, is very important. These tests were carried out to observe whether the group-based
378 feature vector would have a negative effect on the classification performance in deep learning models.

379

380 As can be seen in Table 5, a lower classification success was achieved when compared to the machine
381 learning models, due to the reduction in the number of features. This decrease occurred for all of the
382 metrics. The highest level of success was achieved with the LSTM (100,100) model with 92.2% for
383 accuracy. The precision, recall, and f-score values were 94.4%, 91.6% and 93.9%, respectively. The
384 highest value of 92.5% was obtained in the AUC score, in which both classes were evaluated together.
385 The learning curves for the models with the highest classification rate for DNN, RNN, GRU, and LSTM
386 are shown in Figure 7. For all of the models, learning took place very quickly, with training reaching its
387 peak at 20 epochs. The test curve was parallel to the training curve. However, the learning rate slowed
388 down after 20 epochs. This slowdown was thought to be due to the need for more data to continue
389 learning. Repeating the tests with a larger dataset will allow FG-Droid to achieve a higher AUC.

390

Comparison Results and Best Classifier Results Details

391
392 In the FG-Droid development process, it was understood that the models in which the proposed algorithm
393 produced the most successful results were based on the machine learning algorithms. Machine learning
394 techniques have generally shown high success.

395

396 In order to evaluate the results with all classifiers, ROC curves were taken as shown in Figure-8.
397 Accordingly, 97.0% and above AUC values were obtained in Random forest, ET, DT, and XGBoost
398 algorithms. Obtaining a high classification value in different classifiers strengthens the widespread effect
399 of the proposed feature grouping approach. The highest value of 97.7% was obtained with the ET
400 classifier. This value represents high malware detection. Achieving high classification success with only
401 11 features is valuable.

402

403

404

405 **The effect of proposed feature grouping on learning and testing time**
406 The proposed feature grouping-based algorithm performed a very large feature vector size reduction and
407 feature selection from 349 features to 11 features. Thus, the amount of data was reduced by 30 times and
408 a much simpler feature vector was obtained. This had a serious impact on the training and test duration as
409 well as on the classification result. Having hardware limitations in mobile devices and the need for fast
410 and efficient tools are other advantages in choosing FG-Droid.
411 In the initial state of the created feature vector, in the case of using well-known feature selection methods
412 and in the tests made with the proposed feature grouping method, the required times for training and
413 testing were as shown in Table 6.

414

415 The results obtained when using known and feature selection methods, such as Extratree, randomforest,
416 chi2, f_classif, f_regression, and PCA, or without feature selection, are as shown in Table 6. The FG-
417 Droid tool reached a 97.7 AUC with only 11 features. The minimum number of features required to
418 obtain similar AUC values was 35. It was seen that the increase in the number of selected features had a
419 serious effect on both the training and prediction time. FG-Droid was on average 700% faster in the
420 training time than in the model without any feature selection, while it was approximately 80% faster in
421 the prediction time. On the other hand, the best results were obtained with chi2 among the traditional
422 feature selection methods, and the proposed model was 45% faster in the training time and 23% faster in
423 the testing time when compared to the chi2 method. As a result, by grouping the features with FG-Droid,
424 the effect of the feature on the classification was not lost and an efficient classification was made.
425 **Comparison with similar works and discussion**
426 A lot of work has been done in recent years on Android malware detection. These studies basically used
427 static analysis, dynamic analysis, or hybrid feature extraction methods. While some of these obtained
428 features contributed positively to the classification performance, some may have had no effect at all, and
429 some may have had a deteriorating effect. For this reason, it is beneficial to determine those features that
430 contribute positively to the result and to remove the others from the feature set. In this study, a feature
431 grouping method was proposed for the use of features extracted from static analysis in the classification.
432 FG-Droid used the feature vector consisting of grouped features and made classification with the
433 ExtraTree algorithm. Instead of selecting and removing features from the feature vector, the approach of
434 evaluating these features within the group was adopted. The Drebin malicious dataset, which has been
435 widely used for many years, was used in the tests for FG-Droid.
436 The results of recent studies and the test results of the FG-Droid are shown in Table 7.

437

When the studies were examined, it was understood that Drebin was widely used as a dataset, and in some studies, Androzoo and original datasets were studied. Permissions have generally been the most widely used feature in static analysis-based methodologies. These features were used in the training and testing processes of various classifiers. Classification successes vary between 91.0% and 99.0%. In the analysis time per application, the best value was 0.008. When the studies were compared, it was understood that the tool with the highest classification success and the best analysis time per application was the FG-Droid recommended in this study. In the study conducted in 2021 with an analysis time of 0.008, the classification performance remained at the level of 91.0%. The analysis time of FG-Droid of 0.063 seconds reveals that it is a very efficient model. The most important factor in the emergence of this efficient model is of course working with a low number of features. It works with a lower feature count than all the studies given in the table. In almost all studies, feature selection is avoided and the processing time is compromised in order to achieve high classification success. However, limited resources in mobile devices require consideration in efficiency. In this study, the feature selection approach was handled from a different perspective and instead of selecting features, a joint evaluation approach was adopted.

Conclusions

The increase in mobile devices using the Android operating system has caused them to be the target of cyber attackers. New types of malware are emerging every day, and new methods have been proposed as a precaution. FG-Droid uses a permission grouping-based approach to Android malware analysis. It has an AUC of 97.7% with 11 features for binary classification. Using this newly proposed algorithm, 349 features extracted from Android applications were grouped and reduced to 11 features. Thus, a much more efficient feature vector was revealed. Drebin and Genome malware datasets were used to observe the effect of the model. With the success of the classification, a very efficient model was created with the shortening of the training and prediction times. In the future, tests will be performed with datasets containing more samples to further increase the classification success. Analysis time per application is just 0.063 seconds, one of the best analysis times ever. In addition, since a very fast method has been developed, it is aimed to present it with a platform that will serve online. As a result, FG-Droid is expected to contribute positively to the security of Android smart devices.

References

[1] Palak Khanna, Amandeep Singh, "Google Android Operating System: A Review", International Journal of Computer Applications, 174(4), 1-4, 2016.

[2] https://www.businessofapps.com/data/android-statistics/

[3] https://securelist.com/it-threat-evolution-q1-2021-mobile-statistics/102547/

[4] Xinning Wang, Chong Li, "Android malware detection through machine learning on kernel task structures", Neurocomputing, 435, 126-150, 2021.

[5] Stephan Smalley, Robert Craig, "Security Enhanced Android Bringing Flexible MAC to Android", Network and Disctributed System Security Sypmpsium, 1-18, 2013.

[6] Sharfah Ratibah Tuan Mat, Mohd Faizal Ab Razak, Mohd Nizam Mohmad Kahar, Juliza Mohammad Arif, Ahmat Firdaus, "A Bayesian probability model for Android malware detection", ICT Express, 1-8, 2021.

481 [7] Vikas Sihang, Manu Vardhan, Ashawani Swami, Pradeep Singh, "Signature based malicious
482 behaviour detection in android", computer science, communication and security, 251-262, 2020.
483 [8] Zahoor-Ur Rahman, Sidra Nasim Khan, Khan Muhammed, Jong Weon Lee, Zhihan Lv, Sung
484 Wook Baik, Peer Azman Shah, Khalid Wan, Irfan Mehmood, "Machine learning-assisted
485 signature and heuristic-based detection of malwares in Android devices", Computer & Electrical
486 Engineering, 69, 828-841, 2018.
487 [9] Xin Su, Lijun Xiao, Wenjia Li, Xuchong Liu, Kuan-Ching Li, Wei Liang, "DroidPortrait: Android
488 Malware Portrait Construction Based on Multidimensional Behavior Analysis", Applied Sciences,
489 10(11), 1-20, 2020.
490 [10] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, Fabio Martinelli, "MADAM: Effective
491 and Efficient Behaviour-based Android Malware Detection and Prevention", IEEE Transactions
492 on Dependable and Secure Computing, 15(1), 1-15, 2018.
493 [11] Long Chen, Chunhe Xia, Shengwei Lei, Tianbo Wang, "Detection, Traceability, and
494 Propagation of Mobile Malware Threats", IEEE Access, 9,1-23, 2021.
495 [12] Long Nguyen Vu, Souhwan Jung, "Admat: a cnn-on-matrix approach to Android malware
496 detection and classification", IEEE Access, 9, 1-15, 2021.
497 [13] Nada Lachtar, Duha Ibdah, Anys Bacha, "Toward Mobile Malware detection through
498 convolution neural networks", IEEE Embedded Systems Letters, 13(3), 1-4, 2021.
499 [14] Wikas Sihang, Manu Wardhan, Pradeep Singh, Gauraw Choudhary, Seiil Son, "De-LADY:
500 Deep learning based Android malware detection using dynamic features", Journal of Internet
501 Services and Information Security(JISIS), 11(2), 34-45, 2021.
502 [15] Fau Ou, Jian Xu, "S^3Feature: A static sensitive subgraph-based feature for android malware
503 detection", Computer & Security, 112, 1-17, 2022.
504 [16] Francisco Handrick da Costa, Ismael Medeiros, Thales Menezes, Joao Victor da Silva,
505 Ingrid Lorraine da Silva, Rodrigo Bonifacio, Krishna Narasimhan, Marcio Ribeiro, "Exploring the
506 use of static and dynamic analysis to improve the performance of the mining sandbox approach
507 for android malware detection", Journal of Systems and Software, 183, 1-14, 2020.
508 [17] Recep Sinan Arslan, İbrahim Alper Doğru, Necaattin Barışçı, "Permission-based malware
509 detection system for Android using machine learning techniques", International journal of
510 software engineering and knowledge engineering, 29(1), 43-61, 2019.
511 [18] Saleh M. Shehata, Ahmed H. El Fiky, Mohammed Sh. Torky, Tamer H. Farag, Nesrin
512 Ahmed Abbas, "Android malware prevention on Permission based", International journal of
513 applied engineering research, 15(1), 5-11, 2020.
514 [19] Janai Thiyagarajan, A. Akash, Brindha Murugan, "Improved real-time permission based
515 malware detection and clustering approach using model independent pruning", IET Information
516 Security, 14(5), 531-541, 2020.
517 [20] Jaemin Jung, Hyunjin Kim, Dongjin Shin, Myeonggeon Lee, Hyunjae Lee, Seong-je Cho,
518 Kyoungwon Suh, "Android Malware Detection Based on Useful API Calls and Machine
519 Learning", IEEE First International Conference on Artificial Intelligence and Knowledge
520 Engineering (AIKE), 175-178, 2018.
521 [21] Abdurrahman Pektaş, Tankut Acarman, "Deep learning for effective Android malware
522 detection using API call graph embeddings", Soft Computing, 24(2), 1027-1043, 2020.

523   [22] Akanksha Sharma, Subrat Kumar Dash, " Mining API Calls and Permissions for Android
524   Malware Detection", International Conference on Cryptology and Network Security, 191-205,
525   2014.
526   [23]Moutaz Alazab, Mamoun Alazab, Andrii Shalanginov, Abdelwadood Mesleh, Albara Awajan,
527   "Intelligent mobile malware detection using permission requests and API calls", Future
528   Generation Computer Systems, 107, 509-521, 2020.
529   [24] Vasileios Syrris, Dimitris Geneiatakis, "On machine learning effectiveness for malware
530   detection in Android OS using statis analysis data", Journal of information security and
531   applications, 59, 1-22, 2021.
532   [25] Hossein Fereidooni, Mauro Conti, Danifeng Yao, Alessandro Sperdutti, "ANASTASIA:
533   Android mAlware detection using Static analySIs of Applications", 2016 8th IFIP International
534   Conference on New Technologies, Mobility and Security (NTMS), 1-5, 2016.
535   [26] Yousra Aafer, Wenliang Du, Heng Yin, "DroidAPIMiner: Mining API-Level Features for
536   robust malware detection in android", International conference on security and privacy in
537   communcation systems, 86-103, 2013.
538   [27] Dong-Jie Wu, Ching Hao Mao, Te En Wei, Hahn Ming Lee, Kuo Ping Wu, "DroidMat:
539   Android malware detection through manifest and api calls tracing", 7th Asia joing conference on
540   information security, 1-8, 2012.
541   [28] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano de Cristofaro, Gordon
542   Ross, Gianluca Stringhini, "MAMADROID: Detecting Android malware bu bulding markov chains
543   of behavioral models", Transactions on Privacy and Security, 22(2), 1-34, 2019.
544   [29] Asma Razgallah, Raphael Khoury, Sylvain Halle, Kobra Khanmohammadi, "A survey of
545   malware detection in Android apps: Recommendations and perspectives for future research",
546   Computer science review, 39, 1-17, 2021.
547   [30] Lok Kwong Yan, Heng Yin, "Droidscope seemlessly reconstructing the OS and Dalvik
548   semantic views for dynamic Android malware analysis", 21st usenix conference on security
549   symposium, 29-32, 2012.
550   [31] Gianluca Dini, Fabio Martinelli, Andrea Saracino, Daniele Sgandurra, "madam a multi level
551   anomaly detector for android malware", international conference on mathematical methods,
552   models and architectures for computer network security, 1-17, 2012.
553   [32]Neha Bala, Aemun Ahmar, Wenjia Li, Fernanda Tovar, Arpit Battu, Prachi Bambarkar,
554   "Droidenemey battling adversarial example attacks for Android malware detection", Digital
555   communications and networks, 1-10, 2021.
556   [33] Khaled Bakour, Halil Murat Ünver, "The Android malware static analysis: Techniques,
557   Limitations and Open challenges", 3rd international conference on computer science and
558   engineering(UBMK), 2018.
559   [34] Mohammed K. Alzaylaee, Suleiman Y. Yerima, Sakir Sezer, "DL-Droid deep learning based
560   android malware detection using real devices", Computer and security, 89, 1-11, 2020.
561   [35] Michael Bierma, Eric Gustafson, Jeremy Erickson, David Fritz, Yung Ryn Choe, "Andlantis
562   large-scale android dynamic analysis", Workshop on Mobile Security Technologies (MoST), 1-8,
563   2014.
564   [36] Roopak Surendran, Tony Thomas, Sabu Emmanuel, "A TAN based hybrid model for
565   android malware detection", Journal of Information Security and Applications, 54, 1-11, 2020.

566 [37] Alejandro Martin, Raul Lara-Cabrera, David Camacho, "Android malware detection through
567 hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the
568 OmniDroid dataset", Information Fusion, 52, 128-142, 2019.
569 [38] Fei Tong, Zhen Yan, "a hybrid approach for mobile malware detection in Android", Journal
570 of parallel and distributed computing, 103, 22-31, 2017.
571 [39] Li Chen, Mingwei Zhang, Chih Yuan Yang, Ravi Sahita, "Semi supervised classification for
572 dynamic android malware detection", Cyrtopgraphy and Security, 1-19, 2017.
573 [40] Guiller Suares Tangil, Santanu Dash, Mansour Ahmadi, Johannes Kinder, "Droidsieve fast
574 and accurate classification of obfuscated android malware", international conference on data
575 and application security and privacy, 1-13, 2017.
576 [41] Hui Juan Zhu, Zhu Hong You, Ze Xuan Zhu, Wei Lei Shi, Xing Chen, Li Cheng, "droiddet
577 effective and robust detection of android malware using static analysis along with rotation forest
578 model", neurocomputing, 272, 638-646, 2018.
579 [42] Rahim Taheri, Meysam Ghahramani, Reza Javidan, Mohammad Shojafar, Zahra
580 Pooranian, Moura Conti, "Similarity-based android malware detection using hamming distancec
581 of static binary features", Future generation computer systems, 105, 230-247, 2020.
582 [43] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, Michalis Faloutsos, "Profiledroid multi layer
583 profiling of android applications", 18th annual international conferencec on mobile computing
584 and networking, 137-148, 2012.
585 [44] https://www.sec.cs.tu-bs.de/~danarp/drebin/.
586 [45] http://www.malgenomeproject.org/.
587 [46] ttps://www.virustotal.com/gui/home/upload.
588 [47] Juliza Mohammed Arif, Mohd Faizal Ab Razak, Sharfah Ratibah Tuan Mat, Suryanti
589 Awang, Nor Syahidatul Nadiah Ismail, Ahmad Firdaus, " Android mobile malware detection
590 using fuzzy AHP", Journal of Information Security and Applications, 61, 1-11, 2021.
591 [48] Stuart Millar, Niall McLaughlin, Jesus Martinez del Rincon, Paul Miller, "Multiview deep
592 learning for zero-day Android malware detection", Journal of Information Security and
593 Applications, 58, 1-14, 2021.
594 [49] Nan Zhang, Yu-an Tan, Chen Yang, Yuanzhang Li, "Deep learning feature exploration for
595 Android malware detection", Applied soft computing, 102, 1-7, 2021.
596 [50] https://developer.android.com/docs
597 [51] Vasileios Syrris, Dimitris Geneiatakis, " On machine learning effectiveness for malware
598 detection in Android OS using static analysis data", Journal of Information Security and
599 Applications, 59, 1-22, 2021.
600 [52] Recep Sinan Arslan, İbrahim Alper Doğru, Necaattin Barışçı, "Permission-Based Malware
601 Detection System for Android Using Machine Learning Techniques", International Journal of
602 Software and Knowledge Engineering 29, 43-61, 2019.
603 [53] https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-
604 systems-since-2009/
605 [54] Abdullah Talha Kabakus, Ibrahim Alper Doğru, Aydın Çetin, "APK Auditor: Permission-
606 based Android malware detection system", Digital Investigation, 13, 1-14, 2015.
607 [55] Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K, Drebin: Effective and explainable
608 detection of android malware in your pocket, Proceedings of the Annual Symposium on Network
609 and Distributed System Security (NDSS), 2014.

610   [56] Burguera Iker, Zurutuza Urko, Nadjm-Tehrani, Simin. "Crowdroid: Behavior-Based Malware
611   Detection System for Android", SPSM 11, 15-26, 2011.
612   [57] A. Arora , S. K. Peddoju, "NTPDroid: A Hybrid Android Malware Detector Using Network
613   Traffic and System Permissions," 2018 17th IEEE International Conference On Trust, Security
614   And Privacy In Computing And Communications/ 12th IEEE International Conference On Big
615   Data Science And Engineering (TrustCom/BigDataSE), 808-813, 2018.
616   [58] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song and H. Yu, "SAMADroid: A Novel
617   3-Level Hybrid Malware Detection Model for Android Operating System", IEEE Access, 6, 4321-
618   4339, 2018.
619   [59] Alejandro Martín, Raúl Lara-Cabrera, David Camacho, "Android malware detection through
620   hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the
621   OmniDroid dataset", Information Fusion, 52, 128-142, 2019.
622   [60] S. Y. Yerima, S. Sezer and I. Muttik, "Android Malware Detection Using Parallel Machine
623   Learning Classifiers", 2014 Eighth International Conference on Next Generation Mobile Apps,
624   Services and Technologies, 37-42, 2014.
625   [61] https://www.statista.com/statistics/266488/forecast-of-mobile-app-downloads/
626   [62] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon
627   P. Cox, Jaeyeon Jung, Patrick McDaniel, Anmol N. Sheth, "TaintDroid: An Information-Flow
628   Tracking System for Realtime Privacy Monitoring on Smartphones", ACM Trans. Comput. Syst.
629   32, 2, 2014.
630   [63] T. Bläsing, L. Batyuk, A. Schmidt, S. A. Camtepe and S. Albayrak, "An Android Application
631   Sandbox system for suspicious software detection", 5th International Conference on Malicious
632   and Unwanted Software, 55-62, 2010.
633   [64] Fei Tong, Zheng Yan, "A hybrid approach of mobile malware detection in Android", Journal
634   of Parallel and Distributed Computing, 103, 22-31, 2017.

635

Figure 1

Figure 1 Structure of Android Apk [51]

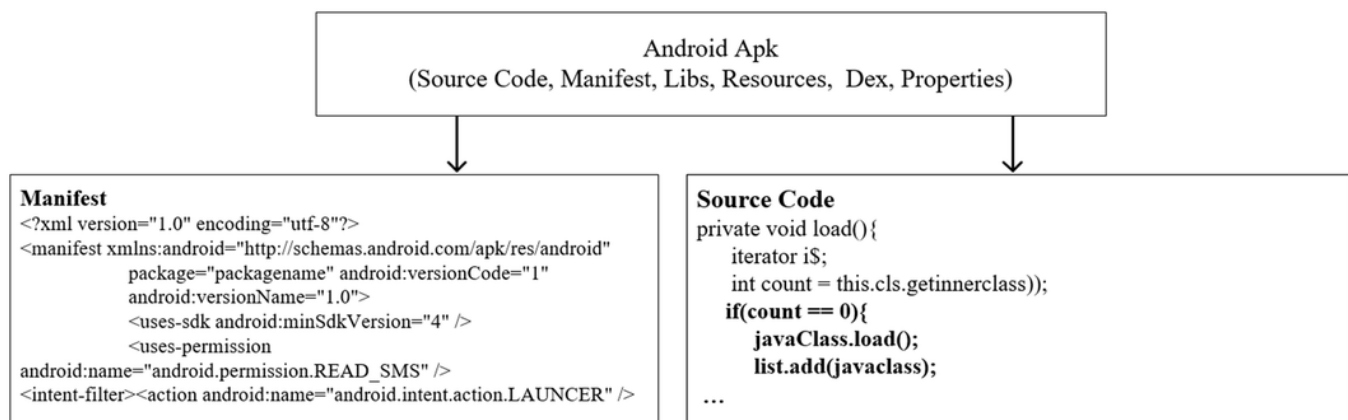



Figure 2

Figure 2 Automatic Feature Extraction and Pre-processing flowchart

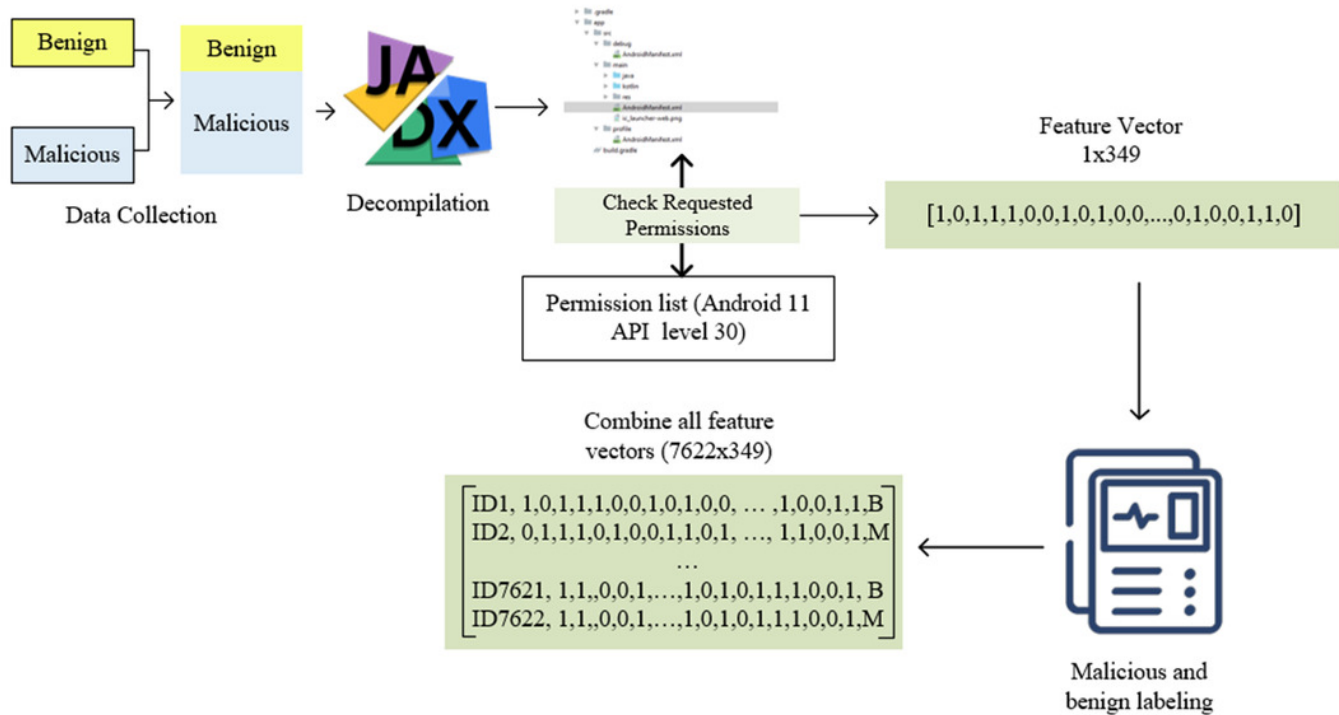



Figure 3

Figure 3 Flow-chart of proposed model

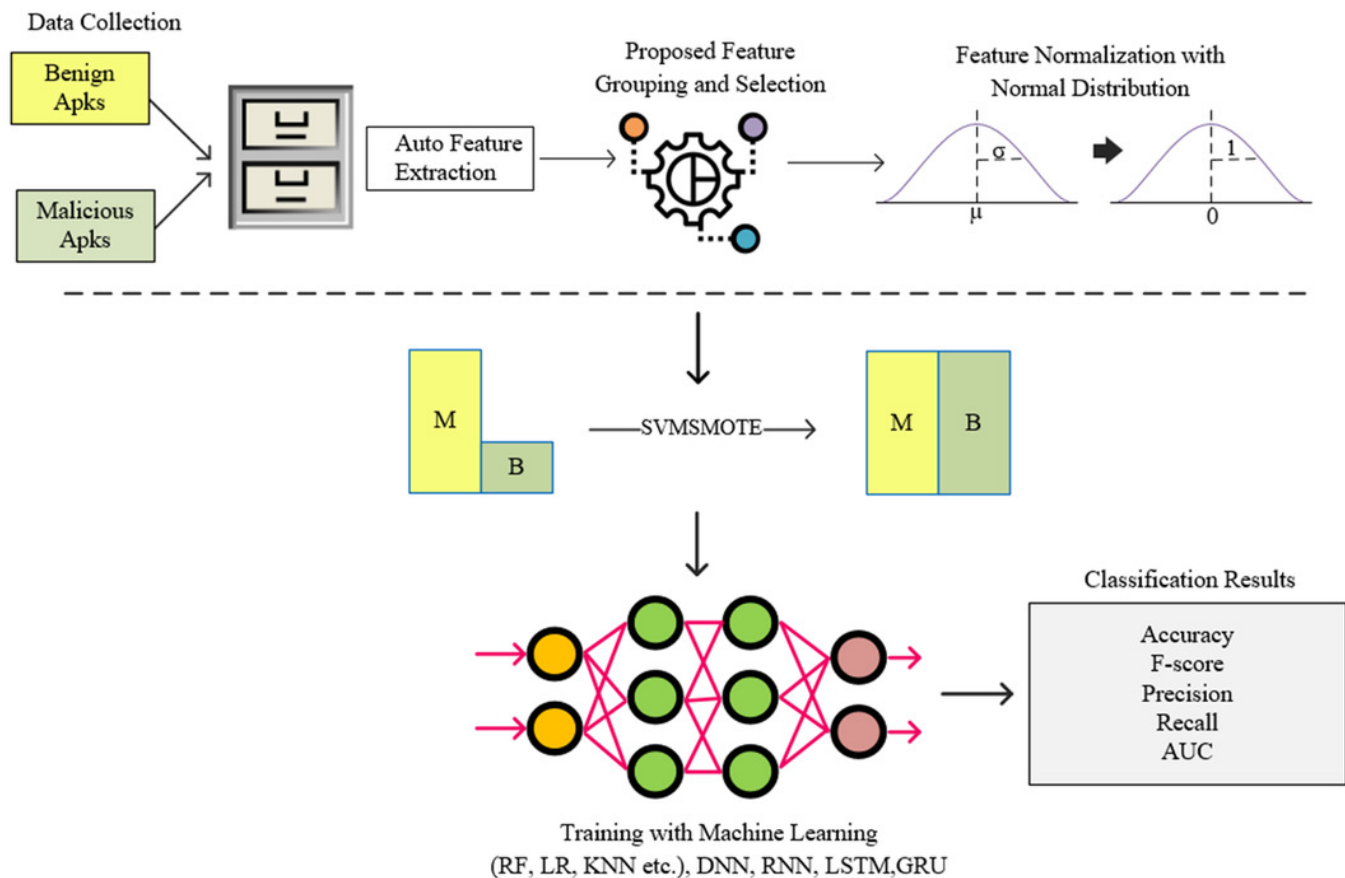



Figure 4

Figure 4 Proposed Feature Grouping and Selection Algorithm

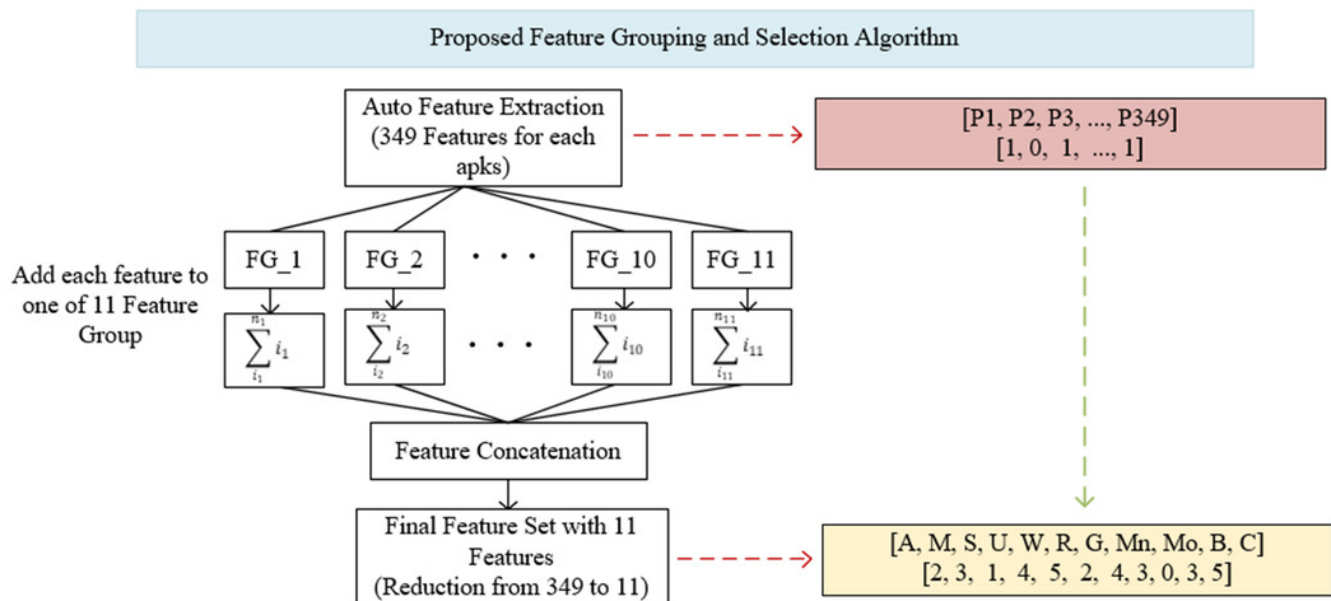



Figure 5

Figure 5 Pseudo-code of feature grouping and selection algorithm

```
Algorithm: Feature Grouping
Input:   originalfv : 2-dimensional array, fgroup: 2-dimensional array
Output: groupedfv: 2-dimensional array

row = len(originalfv)
col = len(originalfv[0])

for i ← 0 to row
    for j ← 0 to col
        for k ← 0 to 11 {
            if originalfv[i][j] in fgroup[k] then
                groupedfv[i][k] += originalfv[i][j];
        }
return groupedfv;
```

Figure 6

Figure 6 Cross-validation graph for all ML classifiers

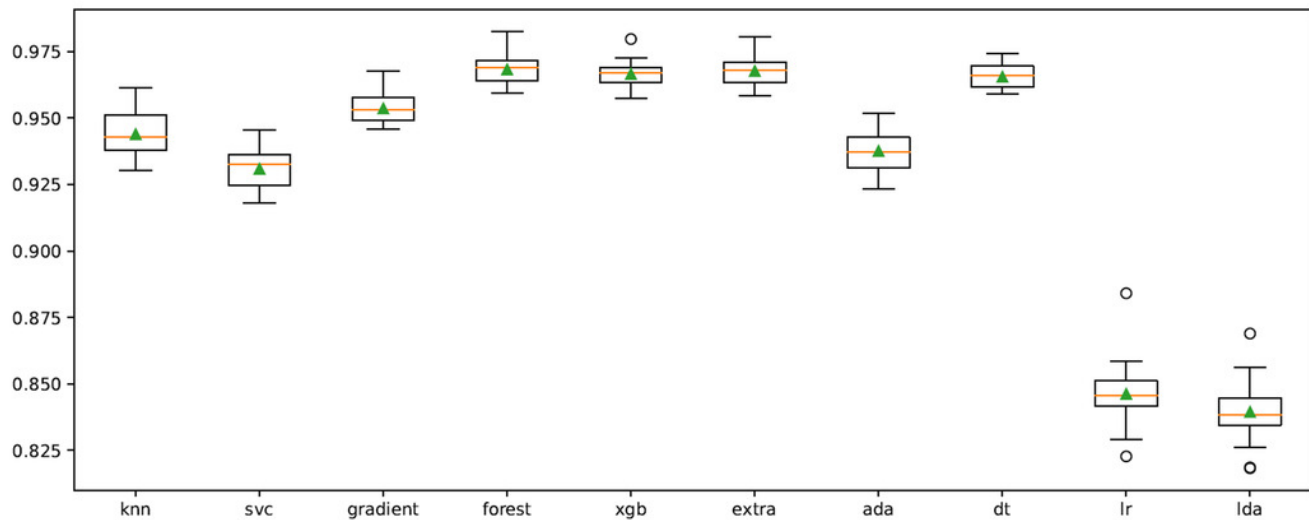
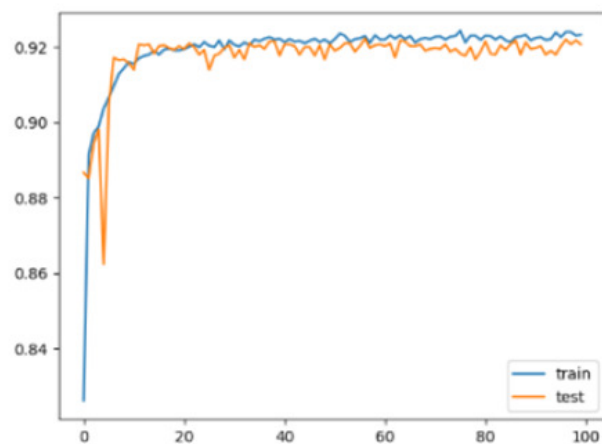
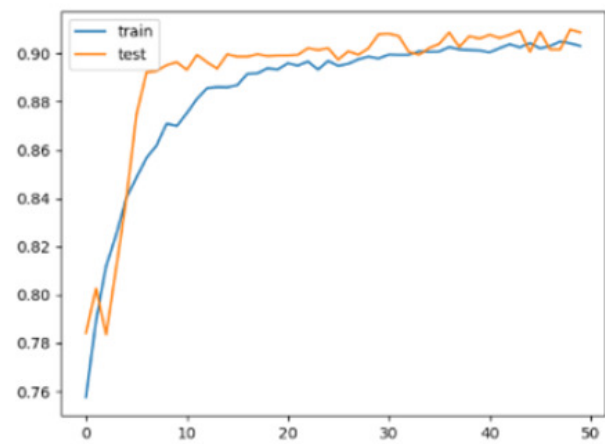



Figure 7

Figure 7 Learning Curves of Deep Learning Models

a) DNN(30,30,30,30)

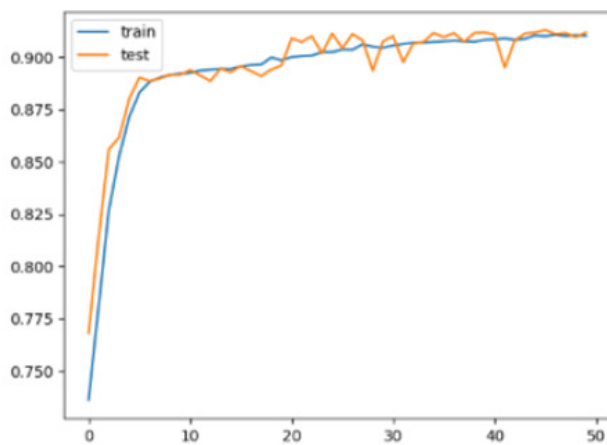
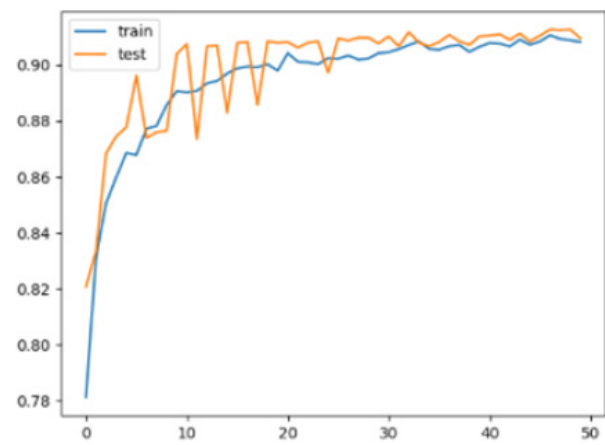
b) GRU(300,300,300)

b) RNN(30,30,30)

d) LSTM(300,300,300)

Figure 8

Figure 8 ROC Curve of all ML Algorithms with Feature Grouping

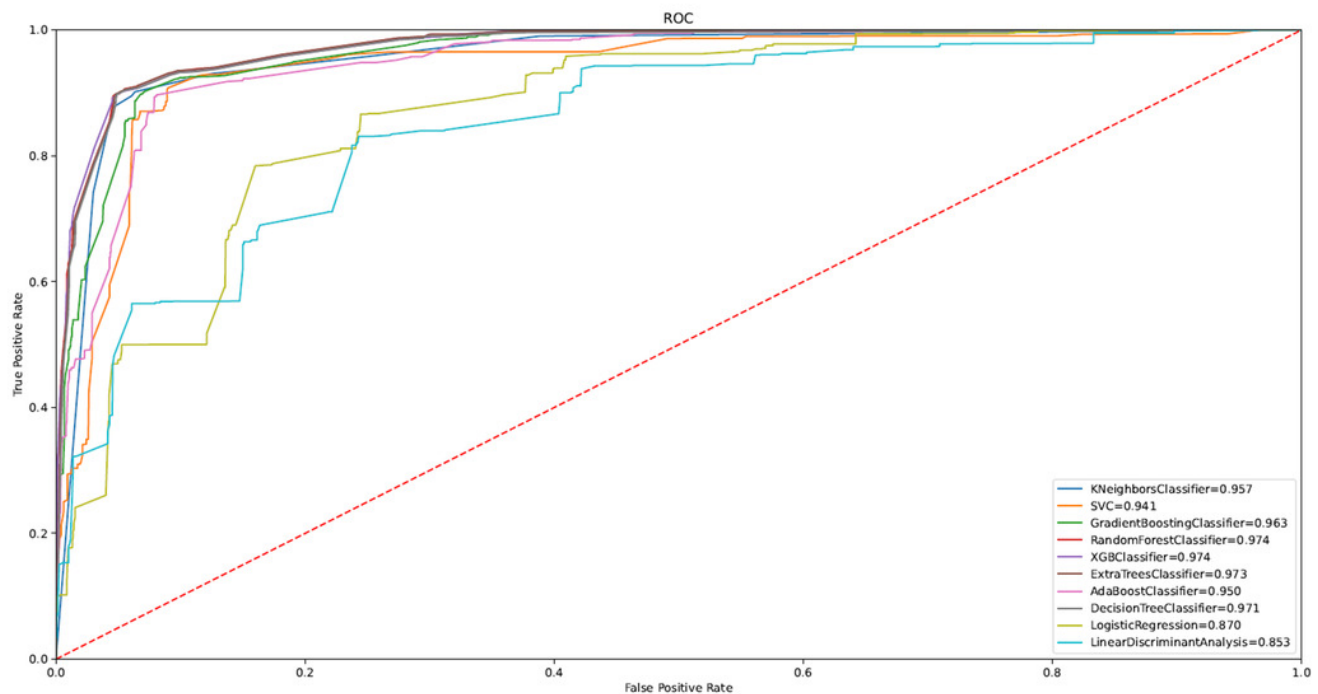



Table 1 Summary of current works on static analysis based in Android malware
detection

1   **Table 1 Summary of current works on static analysis based in Android malware detection**

Ref	Dataset	Feature Extraction	Classification	Classification Rate	Sec. for Identification each app
Arp [55]	Drebin	Used permissions, sys. Api calls, network address	Machine learning	94%	10
Anastasia [25]	Own dataset	Api calls, network address	ML(NB, RF, KNN)	96%	0.29
MamaDroid [28]	Drebin	Api calls, call graphs	SVM, RF, 1-NN, 3-NN	87%	0.7 ±1.5
Taheri [42]	Drebin, Genome	Api calls, intents, permissions(21492 features)	FNN, ANN, WANN, KMNN	90%-99%	Very high
Apkauditor[54]	Own dataset	Permissions, services, receivers	Signature based	92.5%	-
Syrris et al. [24]	Drebin	Static features	ML(6 six classifiers)	99%	-
Droidmat[27]	Own dataset	Intents, Api calls	Signature based	91.83%	-
Alazab [23]	Own dataset	Api calls	ML(RF, J48, KNN, NB)	94.30%	0.2 – 0.92
Pektaş [21]	Drebin, AMD, Androzoo	Api calls	SDNE(DNN model)	98.5%	-
Saleh [18]	Own dataset	Activities, services, receivers, providers, permissions	RF	97.1%	-
Janani [19]	Androzoo	Permissions(113) –> PCA (10)	DT with PCA feature selection	94.3%	-
Proposed Model	**Drebin, Genome, Arslan**	**Permission groups**	**ML, DNN**	**97.7%**	**0.063**

Table 2 Computer System Architecture

Table 2 Computer System Architecture

1   **Table 2 Computer System Architecture**

Parameter	Value
CPU	Intel Core i5-10760
RAM	8 Gigabyte
Operating System	Windows 10 Pro
Python version	3.7.6
Libraries	Scikit learn, matplotlib, pandas, seaborn, numpy, imblearn

Table 3 Details of the hyper-parameters of all classifiers

Table 3 Details of the hyper-parameters of all classifiers

1   **Table 3 Details of the hyper-parameters of all classifiers**

Classifier	Hyper-parameter Tuning Range	Selected Values
KNeighbors	n_neighbors: (1,20,1) p: (1,5,1) weights: ('uniform', 'distance'),	'n_neighbors': 5 'p': 3 'weights': 'uniform'
SVC	C: [0.1, 1, 10, 100, 1000] kernel: ['rbf','linear']	C: 1, kernel: 'rbf'
GradientBoosting	loss: ['log_loss', 'deviance', 'exponential'] "learning_rate": [0.01, 0.025, 0.2]	loss:log_loss "learning_rate": 1.0
Random Forest	n_estimators: [100, 200, 800, 2000] criterion: ["gini", "entropy", "log_loss"]	n_estimators=100 criterion='gini'
XG Boost	'max_depth': [3, 5] 'learning_rate': [0.01, 0.1, 1, 10],	max_depth=5 'learning_rate'=1.0
Extra Tree	'n_estimators': [200-2000] max_features': ['auto','sqrt','log2'] max_depth: [10-110] Min samples split: [2,5,10]	'max_features': sqrt 'n_estimators': 1800 max_depth:30 Min samples split:10
Ada Boost	'learning_rate': [0.01, 0.1, 1, 10], 'n_estimators': [50, 500, 2000]	'learning_rate': 1 'n_estimators': 50
Decision Tree	'criterion': ['gini', 'entropy'],	criterion='gini'
Logistic Regression	'C': [1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03], 'penalty': ['l1', 'l2']}	C': 1.0, 'penalty': 'l2'
Linear Discriminant Analysis	solver : ['svd', 'lsqr', 'eigen']	'solver': 'svd'

Table 4 ML Classification Results with Proposed Feature Grouping Algorithm

1   **Table 4 ML Classification Results with Proposed Feature Grouping Algorithm**

ML Classifier Algorithm	AUC_score	Precision	Recall	F-score	Accuracy
KNeighbors	0.965	0.905	0.913	0.909	0.910
SVC	0.950	0.844	0.931	0.885	0.881
GradientBoosting	0.965	0.928	0.899	0.913	0.916
Random Forest	0.976	0.942	0.904	0.923	0.925
XG Boost	0.976	0.941	0.905	0.923	0.925
Extra Tree	0.979	0.944	0.909	0.926	0.929
Ada Boost	0.955	0.917	0.872	0.894	0.898
Decision Tree	0.970	0.939	0.901	0.920	0.922
Logistic Regression	0.881	0.790	0.816	0.803	0.803
Linear Discriminant	0.856	0.769	0.814	0.791	0.788

Table 5 Deep Learning Models Classification Results with Proposed Feature Grouping Algorithm

1   **Table 5 Deep Learning Models Classification Results with Proposed Feature Grouping Algorithm**

Deep Learning Models	Auc_score	Precision	Recall	F-score	Accuracy	Total number of processed parameters	Epoch
DNN(30,30)	0.918	0.936	0.906	0.921	0.925	1352	50
DNN(30,30,30)	0.920	0.927	0.916	0.921	0.921	2282	50
DNN(30,30,30,30)	0.935	0.942	0.908	0.925	0.930	3212	50
DNN(100,100,100)	0.925	0.944	0.891	0.917	0.924	31702	50
DNN(300,300,300)	0.928	0.940	0.904	0.921	0.926	275102	50
RNN(10,10)	0.900	0.910	0.910	0.920	0.902	242	50
RNN(30,30)	0.916	0.928	0.891	0.939	0.918	1322	50
RNN(100,100)	0.867	0.899	0.830	0.863	0.875	11402	50
RNN(300,300)	0.887	0.905	0.865	0.885	0.895	94202	50
GRU(10,10)	0.910	0.918	0.897	0.907	0.908	712	50
GRU(30,30,30)	0.902	0.915	0.887	0.901	0.905	3932	50
GRU(100,100)	0.915	0.836	0.891	0.913	0.913	34102	50
GRU(300,300)	0.914	0.934	0.893	0.913	0.915	282302	50
LSTM(10,10)	0.920	0.939	0.891	0.914	0.918	902	50
LSTM(30,30)	0.909	0.890	0.908	0.908	0.910	5102	50
LSTM(100,100)	0.916	0.940	0.890	0.914	0.920	45002	50
LSTM(300,300)	0.916	0.937	0.892	0.914	0.920	375002	50

Table 6 Training and testing time with different feature selection algorithms and proposed model

1   **Table 6 Training and testing time with different feature selection algorithms and proposed model**

Feature Selection Method	Selected/Grouped Feature Size	Training time(s.)	Prediction time(s.)
FG-Droid (Proposed Model)	11	0.344	0.063
Without feature selection	349	2.234	0.109
Feature importance with extra tree algorithm	46	0.609	0.078
Feature importance with random forest algorithm	45	0.594	0.078
chi2	35	0.500	0.078
f_classiif	35	0.519	0.078
f_regression	35	0.516	0.078
PCA	35	0.688	0.078

Table 7 Similar works with proposed model

1   **Table 7 Similar works with proposed model**

Paper	Dataset	Feature Extraction	Feature selection or grouping algorithms and classification methods	Classification Performance	Sec. for identification each app
Mat et al. [6]	Androzoo, Drebin	Permissions	Naïve Bayesian	91.10%	-
Arif et al. [47]	Androzoo, Drebin	Permissions	MCDM	90.54% (4 classification levels)	-
Millar et al. [48]	Drebin, Genome	Opcode and permissions	No feature selection, classification with multi-view deep learning	91.00%	0.008
Zhang et al. [49]	Drebin	Text sequences of apps	No feature selection. Classification with Text CNN	95.20%	0.28
Arp [55]	Drebin	Used permissions, sys. Api calls, network address	Machine learning	94%	10
Anastasia [25]	Own dataset	Api calls, network address	ML(NB, RF, KNN)	96%	0.29
MamaDroid [28]	Drebin	Api calls, call graphs	SVM, RF, 1-NN, 3-NN	87%	0.7 ±1.5
Taheri [42]	Drebin, Genome	Api calls, intents, permissions(21492 features)	FNN, ANN, WANN, KMNN	90%-99%	Very high
Apkauditor[54]	Own dataset	Permissions, services, receivers	Signature based	92.5%	-
Syrris et al. [24]	Drebin	Static features	ML(6 six classifiers)	99%	-
Droidmat[27]	Own dataset	Intents, Api calls	Signature based	91.83%	-
Alazab [23]	Own dataset	Api calls	ML(RF, J48, KNN, NB)	94.30%	0.2 – 0.92
Pektaş [21]	Drebin, AMD, Androzoo	Api calls	SDNE(DNN model)	98.5%	-
Saleh [18]	Own dataset	Activities, services, receivers, providers, permissions	RF	97.1%	-
Janani [19]	Androzoo	Permissions(113) –> PCA (10)	DT with PCA feature selection	94.3%	-
FG-Droid	**Drebin, Genome, Arslan**	**Permission groups**	**ML, DNN**	**97.7%**	**0.063**