information

Article

A Context-Aware Android Malware Detection Approach Using Machine Learning

Mohammed N. AlJarrah [1], Qussai M. Yaseen [1,2,*] and Ahmad M. Mustafa [1]

[1] CIS Department, Jordan University of Science and Technology, Irbid 22110, Jordan
[2] Artificial Intelligence Research Center (AIRC), Ajman University, Ajman 346, United Arab Emirates
* Correspondence: q.yaseen@ajman.ac.ae

Abstract: The Android platform has become the most popular smartphone operating system, which makes it a target for malicious mobile apps. This paper proposes a machine learning-based approach for Android malware detection based on application features. Unlike many prior research that focused exclusively on API Calls and permissions features to improve detection efficiency and accuracy, this paper incorporates applications' contextual features with API Calls and permissions features. Moreover, the proposed approach extracted a new dataset of static API Calls and permission features using a large dataset of malicious and benign Android APK samples. Furthermore, the proposed approach used the Information Gain algorithm to reduce the API and permission feature space from 527 to the most relevant 50 features only. Several combinations of API Calls, permissions, and contextual features were used. These combinations were fed into different machine-learning algorithms to show the significance of using the selected contextual features in detecting Android malware. The experiments show that the proposed model achieved a very high accuracy of about 99.4% when using contextual features in comparison to 97.2% without using contextual features. Moreover, the paper shows that the proposed approach outperformed the state-of-the-art models considered in this work.

Keywords: Android; API Calls; contextual information; machine learning; malware; permissions

Citation: AlJarrah, M.N.; Yaseen, Q.M.; Mustafa, A.M. A Context-Aware Android Malware Detection Approach Using Machine Learning. *Information* **2022**, *13*, 563. https://doi.org/10.3390/info13120563

Academic Editor: Krzysztof Szczypiorski

Received: 27 September 2022
Accepted: 26 November 2022
Published: 30 November 2022

1. Introduction

The Internet's expansion and the technical revolution in smartphones have led to a tremendous increase in the number of smartphone users. This encouraged competition among various software industries to serve customers by releasing powerful platforms for their smartphones. Android is the most popular mobile operating system, with millions of users around the world taking advantage of its services. Google created and developed the Android operating system in 2005, and the first Android smartphone was introduced in 2008 [1]. By 2021, there were approximately 2 Billion Android-based devices (smartphones and tablets), indicating that Android applications are rapidly growing and exceeding other mobile operating systems such as IOS, Windows, and others [2]. As the most popular and powerful platform, Android provides a vast number of mobile applications in various categories to be available for all android-based mobile users of various ages worldwide.

Android malware is growing immensely due to the vast growth of Android users, which poses threat to the security and privacy of Android users. Android malware is known for sending fraudulent SMS, misusing users' private information, devouring traffic, downloading malicious applications, remote control, data exploitation, and other dangerous behaviors [3]. According to some statistics [4], the number of Android-based malware cases rises every year. About 3.5 million Android malware samples were observed in the first quarter of 2021, up from 1 million in 2019 and 2020.

Android malware differs in various ways; it acts differently, hacks differently, and performs different damage. Some Android malware infiltrates the device by exploiting

the user and then launching an assault via malicious applications, while others replicate and clone themselves in various locations before installing malicious applications in these locations to inflict and spread damage as broadly as possible. Table 1 shows a list of mobile malware types and their behaviors, as well as an example of each one.

Table 1. Types of mobile malware.

Mobile Malware	Behavior	Example
Trojan [5]	Looks to be a harmless application that convinces users to download it and then installs malware on their mobile devices.	Android.Counterlank
Worms [6]	Worms can infect additional devices while they are operating on infected systems, and they can carry a payload that degrades mobile network capacity.	Ikee.B
Adware [7]	Deceives the user through malicious advertising.	UAPush
Spyware [8]	Collects user's data and behavior, such as email and passwords, and sends it to another location across the network.	Zitmo
Botnet [9]	Comprises many internet-connected cellphones controlled by a malicious user; it gains full access to the device and its contents and sends data to the malicious controller.	Not compatible

The diversity and complexity of Android malware, as well as the employment of various strategies to elude detection, make traditional malware detection techniques ineffective, necessitating the development of more efficient and powerful ways to overcome this constraint [10]. Existing malware detection methods are limited and only reveal malware after it has been infected. Automated detection techniques, such as the use of supervised and unsupervised machine learning algorithms, operate effectively by extracting app features using both static and dynamic analysis to execute an optimal and clear classification of Android apps into two groups: malware or benign [11]. Current detection software is unable to detect zero-day attacks. As a result, most researchers use various machine learning classifiers in the detection of Android-based malware, such as Support Vector Machines (SVM), Naïve Bayes (NB), Random Forest (RF), and Decision Trees (DT) [12]. Machine learning algorithms take advantage of features and characteristics learned from malware and benign samples during training.

Machine learning detection techniques rely on static, dynamic, and hybrid analysis to extract and gather application features that are used to classify and detect malicious behaviors. System API Calls, permissions, privileges used, and contextual information are some of the extracted features that machine learning-based algorithms use [13]. The main machine-learning approaches used to detect Android malware are static and dynamic approaches.

Static analysis involves extracting features from Java bytecode or the AndroidManifest.xml file, which contains contextual information and a collection of features, such as permissions required by the app [14]. The static analysis of Android applications to extract static features is shown in Figure 1.

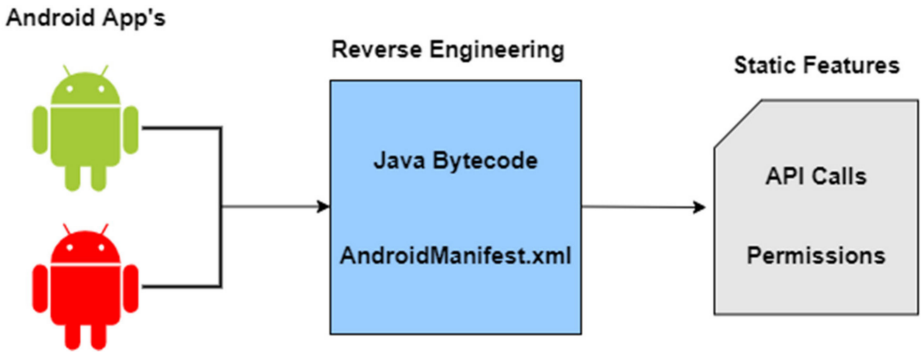

Figure 1. Features extraction-based static analysis.

Dynamic analysis is used to discover harmful behavior in applications while they are running. It collects the system calls that the application makes while it is executing. Furthermore, the dynamic analysis also works well with unidentified application signatures [15]. Figure 2 illustrates the dynamic analysis of Android applications.

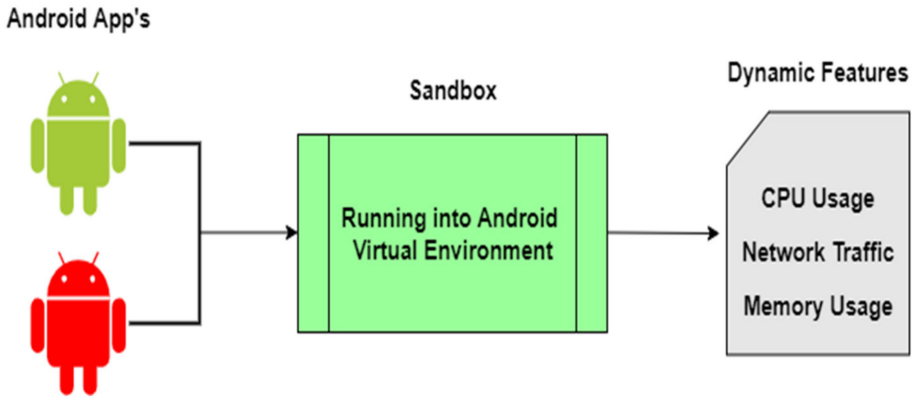

Figure 2. Features extraction-based dynamic analysis.

The hybrid technique combines static and dynamic features to improve malware detection results and prevent flaws that can occur when using either a static or dynamic approach alone. The hybrid analysis of Android applications is illustrated in Figure 3.

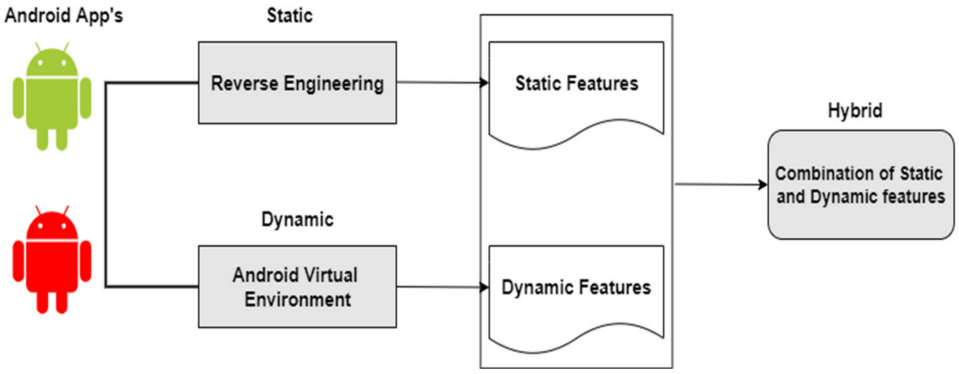

Figure 3. Features extraction-bdased hybrid analysis.

The benefits of a static analysis include the ability to detect malware before it executes, as well as the ability to detect unknown malware and code vulnerabilities. A dynamic analysis has the advantage of being able to detect undiscovered malware as well as zero-day attacks [16]. However, a dynamic analysis can be time-consuming and resource-intensive usage.

The contribution of the paper is summarized as follows:

1. The paper created a new dataset of static API Calls and permissions features from a large number of Android APKs.
2. The paper selected and used the most relevant contextual features along with the API Calls and permissions to test the efficacy of using contextual information in detecting Android malware.
3. The proposed model used the Information Gain algorithm [17] to reduce the feature space from 527 API Calls and permissions to 50 features only and achieve a very close accuracy to what was achieved using 527 features.
4. The paper tested several machine learning algorithms, which are Random Forest, Logistic Regression, SVM, K-NN, and Decision Trees using different combinations of API Calls, permissions, and contextual features to evaluate their accuracy in detecting Android malware.
5. The experiments show that using the selected contextual features, the proposed model achieved a high accuracy of about 99.4% in detecting Android malware.
6. The paper considers different state-of-the art models that used contextual features or the same dataset used in this work, and it shows that the proposed models outperformed the state-of-the art models.

The rest of the paper is organized as follows. Section 2 discusses some related work. Section 3 describes the methodology of the proposed approach, discusses the dataset, data pre-processing, features extraction, features selection, and machine learning algorithms. Section 4 shows and discusses the experiments and results. Finally, Section 5 concludes the work.

2. Related Work

Many research approaches have been conducted on detecting Android malware using machine learning. This section discusses some related work in this direction.

Le et al. [18] proposed an approach for Android malware detection that employs different machine learning methods as detectors to identify and detect malicious Android applications. They extracted the features using the static analysis technique by decompiling source files into snail code and using some C++ libraries to read the information from AndroidManifest.xml. In their work, Decision Tree, Naïve Bayes, and an ensemble of Random Forest, Stochastic Gradient Boosting, and AdaBoost were trained based on some application features such as behavior, permission, the size of the application, the class number in the application, and the user interface number of the application. The used dataset contained about 16,589 malicious Android applications collected from different sources, such as Virusshare [19] and Koodous [20], in addition to 12,290 benign Android applications installed from Google Play. The results of their approach showed that the Random Forest classifier achieved the best accuracy at about 98.66%.

Kaushal et al. [21] used permissions from AndroidManifest.xml files as features of Android applications to build an automated Android malware detection system. They trained two machine learning algorithms (Support Vector Machine (SVM) and Naïve Bayes) with a deep learning algorithm (Recurrent Neural Network with LSTM architecture). Then, they used the extracted permissions to perform malware classification into malicious or benign. Their results showed that the Recurrent Neural Network achieved the best results with an accuracy of 95%, outperforming other machine learning classifiers.

Hyoil et al. [22] used Support Vector Machines (SVM) for Android malware detection. They have used a dataset of two samples of Android applications (malicious and benign), where the number of malicious samples is 30,113 applications from the AMD [23] and Drebin [24] dataset, and the benign samples contain 28,489 applications downloaded from Google Play, the Amazon AppStore, and APKPure [25]. Then, they employed a static analysis technique to extract 133,227 API Calls to be used as features for classification. They claimed that the experiments showed that their approach outperformed existing approaches by obtaining an accuracy of 99.97% in detecting malicious Android applications.

Bilal et al. [26] used static and dynamic analysis techniques to propose an approach for Android malware detection using machine learning algorithms based on a hybrid of static, dynamic, and some intrinsic features. They extracted 20 different features from a sample of about 600 Android applications (malicious and benign) that were collected from the Androtracker project [27] and Google Play store [28]. After extracting these features, two machine learning classifiers, which are the K-Nearest Neighbor and Logistic Regression, were created as detection models to perform malware classification. The experiments showed that the proposed approach performs well, and both machine learning classifiers achieved the same accuracy of 97.5% in malware detection on the same training dataset, while the Logistic Regression classifier outperformed other classifiers over the testing dataset.

Fang et al. [29] proposed a method based on the Dalvik Executable file (Dex file) for Android malware family classification. The method extracted the Dex files of 24,553 Android malicious samples from the Android Malware Dataset (AMD) [23] and obtained RGB images and plaintext from the DEX file. Next, it extracted the text features of plaintext, as well as the color and texture features of images. To perform the classification, the study used the feature fusion algorithm based on multi-kernel machine learning. The experiments showed that the proposed method achieved an accuracy of about 96%.

Danish et al. [30] proposed an image-based malware families multiclassification detection method using fine-tuning Convolutional Neural Network (CNN). The method transformed the raw malware binary files into color images to be used as inputs to the CNN for classification. The experiments were performed on two different datasets. The first one is the Malimg malware dataset [31], which consists of 9435 malicious samples, while the other dataset is the IoT-Android mobile dataset, which includes 14,733 malicious samples and 2486 benign samples. The data augmentation technique was adopted by the proposed method during the fine-tuning process. The experiments showed that the proposed method achieved an accuracy of about 98.82% on the Malimg malware dataset, while on the IoT-Android mobile dataset achieved an accuracy of about 97.35%.

Halil et al. [32] proposed an approach for Android malware classification and detection based on a visualization technique and various machine-learning algorithms. They converted some Android application files (Manifest.xml, DEXcode, and Resources (ARSC)) into grayscale images to extract different types of global and local image-based features to be used for training. Before training the algorithms, they normalized the extracted global features in one feature vector and applied the Bag of Visual Words (BOVW) algorithm to build one feature vector from the extracted local features descriptors. To test the model, they performed the experiments on three grayscale image datasets that consisted of 4850 benign samples and 4580 malicious samples for each one. Six machine learning algorithms, Random Forest, K-Nearest Neighbor, Decision Tree, Bagging, AdaBoost, and Gradient Boost, were tested. The experiments showed that the model achieved an accuracy of about 98.75%.

Nuren et al. [33] proposed a machine learning-based approach for Android malware detection. They extracted the application permissions from the AndroidManifest.xml file using static analysis and loaded them in WEKA, where the top 15 permissions were used as malware features. Five machine learning classifiers, Random Forest, J48, Multi-Layer perceptron, Decision Table, and Naïve Bayes, were trained for classification. Using a dataset of 10,000 malicious Android applications and 10,000 benign applications, the Random Forest classifier achieved the best accuracy of about 89.36%.

Talal et al. [34] conducted an empirical study for Android malware detection using various supervised machine-learning algorithms. They employed static analysis to extract some features from the AndroidManifest.xml and Dex files. The extracted features were permissions, intents, and API Calls. Then, they evaluated and compared the performance of six different supervised machine learning classifiers, K-Nearest Neighbour, Support Vector Machine, Decision Tree, Naïve Bayes, Random Forest, and Logistic Regression, on a dataset of 1260 malicious Android applications and 2539 benign Android applications. The results

showed that the Random Forest classifier achieved the best accuracy of about 99.21%, while the Naïve Bayes classifier achieved the lowest detection accuracy of about 95.45%.

Other methods in this field were performed by Du et al. [35], Narayanan et al. [36], Mahdavifar et al. [37], and Hadiprakoso et al. [38]. Du et al. [35] proposed a context-based approach that used the semantics and contextual information of the network flow of Android applications. Similarly, Narayanan et al. [36] proposed a contextual-based approach that used a multiple kernel learning method to detect malicious code patterns. Mahdavifar et al. [37] and Hadiprakoso et al. [38] used the same dataset that is used in this work, which is CIC_Maldroid2020 [39]. However, both approaches did not use contextual information. The aforementioned approaches in this paragraph are selected as the state-of-the-art models and are explained in detail in Section 5.

3. Methodology

This section introduces and discusses the methodology of the proposed approach. Figure 4 shows an illustration of the methodology, and each step is discussed in the following subsections.

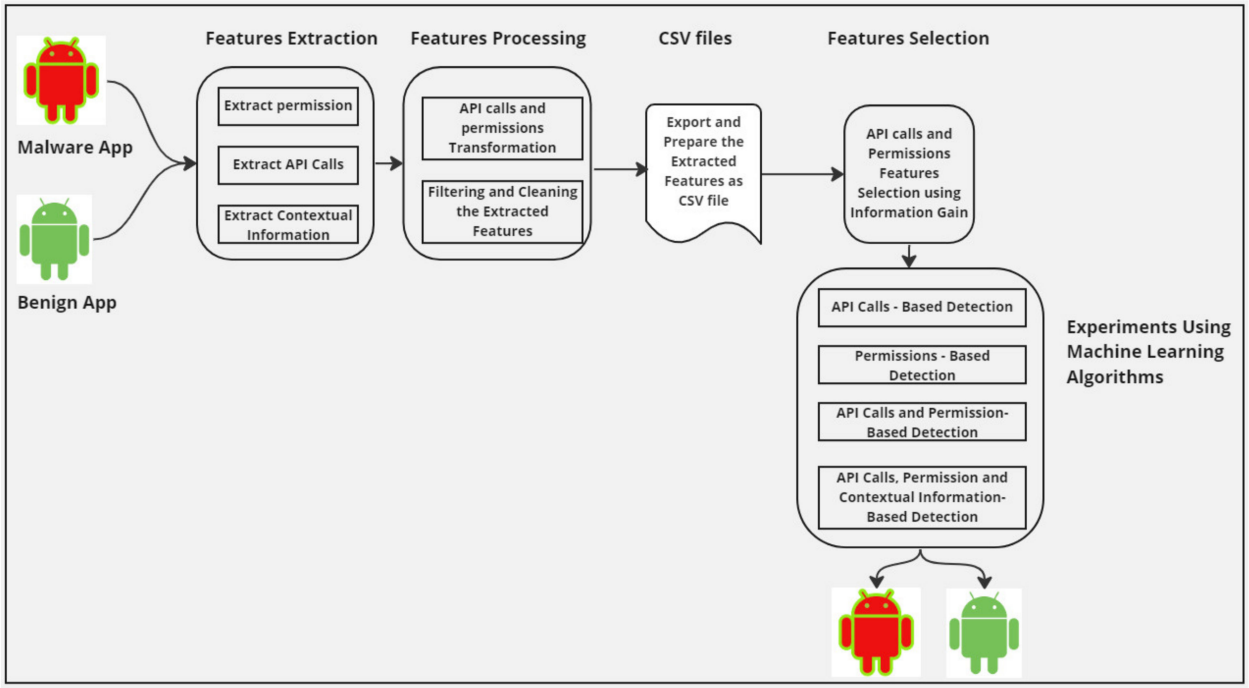

Figure 4. A framework for context-aware Android malware detection approach using machine learning techniques.

3.1. Datasets

Unlike many studies conducted in this field, which used small datasets, this paper used a large dataset of APK samples of malicious and benign Android applications (APKs) called CICMalDroid2020 [39] and extracted a new dataset of API Calls and permissions features. The APKs were collected and published by Mahdavifar et al. [37] and provided by the Canadian Institute for Cybersecurity [40]. It consists of about 16,900 Android samples in different categories; 12,800 samples of the dataset are malware applications, while the rest of the 4100 samples are benign applications. The dataset was collected from 2017 to 2018 from different sources such as MalDozer [41], AMD [23], the VirusTotal service [42], and the Contagio security blog [43]. The collected Android application samples include various application categories, such as advertising, social, educational, etc. The dataset categories samples and their numbers are shown in Figure 5. Table 2 provides a brief explanation of each category of malware inspected in this paper.

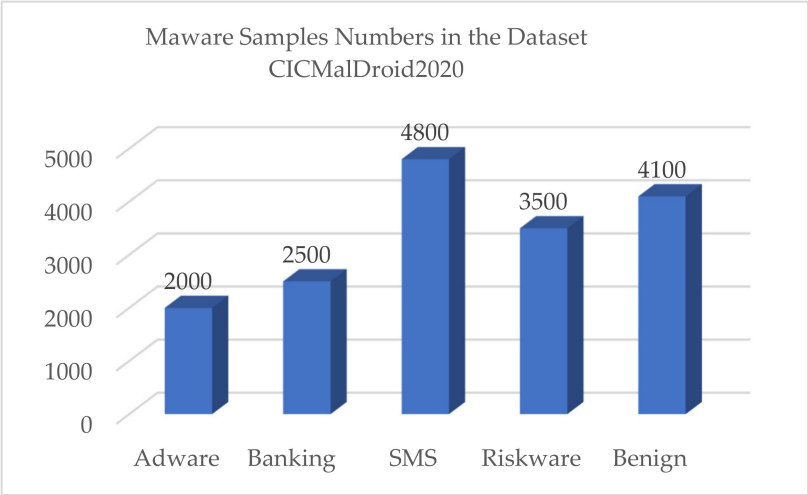

Figure 5. Dataset description.

Table 2. Malware categories.

Android Malware Category	Concept
Adware	Malware uses advertising to exploit the user.
Banking	Malware exploits the banking accounts of the user.
SMS	Malware exploits the user by sending a malicious SMS.
Riskware	A program that behaves as good but is malware.

3.2. Features Extraction

The samples of Android applications are in the form of an Android Application Package (APK). An APK is a file that holds all files and components that are responsible for running the application. Figure 6 illustrates the structure of the Android application (APK) [44]. APKs need to be converted and analyzed to get the features that will be used for detection.

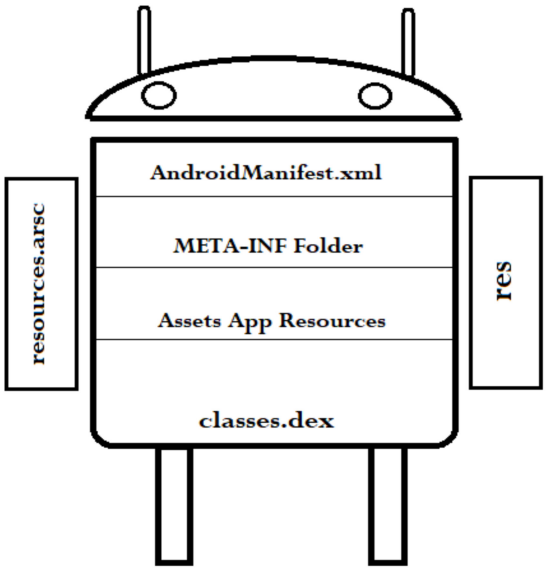

Figure 6. Android Application Package (APK) structure.

Static analysis is a technique to extract static application features from the APK files without running the application. This paper adopts a static analysis approach to extract

static features from the applications such as API Calls, permissions, and some contextual information, using Python programming language. The feature extraction process produced a total of 531 distinct features, as shown in Table 3. More information about the extracted features is provided next.

Table 3. Number of extracted features.

Application Features Set	Number of Extracted Features
API Calls	15
Permissions	512
Contextual Information	4
Total	531 Features

3.2.1. API Calls Features

The Application Programming Interface (API) is a set of rules that the application uses for communication. API Calls are considered a significant indicator to distinguish between malware and benign applications and to reveal any suspicious behavior [45]. Therefore, we extracted a set of API Calls from Android APK samples to be used for malware detection. Table 4 shows the extracted API Calls. The API Calls for each application were extracted using a script code written in Python using "Androguard", which is a full-featured Python utility for manipulating and handling Android files [46]. It uses reverse engineering by analyzing the DEX file of each APK file [47]. Then, API Calls were converted into binary values (0 or 1) that indicate the presence of API Calls in an APK. A total of 15 API Call features were extracted from 15,836 Android samples, resulting in 11,800 malicious applications and 4036 benign applications. Table 5 displays the number of Android application samples for each category from which API Call features were extracted.

Table 4. Extracted API Call categories.

API Calls	Desc.
startService	Requests the launch of a specific app service.
getDeviceId	Gets the device ID from which an event originated.
createFromPdu	Sending an SMS message using the Protocol Data Unit (PDU), which is a cellular data transmission technology.
getClassLoader	Returns a class loader that can be used to get classes from a package.
getClass	Returns the object's runtime class.
getMethod	Returns a method object that represents the class or interface represented by this class object's public member method.
getDisplayOriginatingAddress	Gives the message's originating address, or the email address if it was sent through an email gateway.
getInputStream	Returns a read-only input stream from any of the open connections.
getOutputStream	Returns a write-only output stream to the specified connection.
killProcess	The process with the supplied ID will be terminated.
getLine1Number	For line 1, this function returns the phone number string.
getSimSerialNumber	Gives the SIM serial number.
getSubscriberId	Provides the subscriber's unique ID.
getLastKnownLocation	Returns the data from the last known location retrieved from the supplied source.
isProviderEnabled	Returns if the given provider is enabled or disabled.

Table 5. Number of Android application samples from which API Call features were extracted.

Android Application Category	Number of Samples from Which API Call Features Were Extracted
Adware	1499
Banking	2277
SMS	4761
Riskware	3263
Benign	4036
Total	**15,836**

3.2.2. Permissions Features

Android application permissions grant apps access to the phone's hardware and data, as well as the ability to control the phone. Permissions can be legitimate or malicious. For example, when an application asks for permission to access sensitive data, such as the phone book or the camera, the application could be suspicious and potentially malicious. Therefore, permissions are powerful indicators for detecting malicious apps and separating them from benign ones. Figure 7 shows examples of some requested permissions by an Android application.

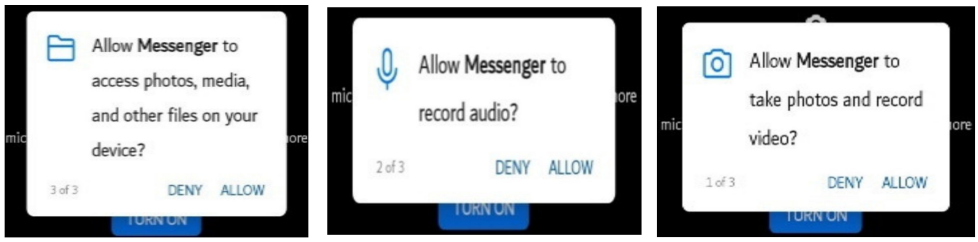

Figure 7. Android application permissions.

A large set of permissions were extracted from Android application samples for use in malware detection. Tables 6 and 7 provide a brief description of some normal and dangerous permissions extracted from various Android applications [44]. Each application's used permissions were extracted using a Python script code using "Androguard" and then converted to binary representation "0 or 1". About 700 permissions features were extracted from 16,703 Android samples, including 12,692 malicious apps and 4011 benign apps. The number of Android application samples from which permissions features were extracted is shown in Table 8.

Table 6. Sample of normal extracted permissions.

Normal Permission	Desc.
'android.permission.INTERNET'	This permission opens network ports for applications.
'android.permission.ACCESS_WIFI_STATE'	Permits Wi-Fi network information to be accessed by apps.
'android.permission.ACCESS_NETWORK_STATE'	Allows apps to gain access to network information.
'android.permission.SET_WALLPAPER'	Allows apps to change the background image.
'android.permission.SET_TIME_ZONE'	Allows apps to change the time zone of the phone.

Table 7. Sample of extracted dangerous permissions.

Dangerous Permission	Desc.
'android.permission.READ_CONTACTS'	Allows apps to access the contact information of the user.
'android.permission.CAMERA'	Allows apps to gain access to the phone camera.
'android.permission.READ_CALL_LOG'	Allows apps to see the call log of a user.
'android.permission.SEND_SMS'	This permission enables apps to send text messages.
'android.permission.READ_PHONE_STATE'	Gives apps access to the current state of the phone, such as the device's phone number, cellular network, and active calls.

Table 8. Number of Android application samples from which permissions features were extracted.

Android Application Category	Number of Samples from Which Permissions Features Were Extracted
Adware	1499
Banking	2494
SMS	4803
Riskware	3896
Benign	4011
Total	**16,703**

3.2.3. Contextual Features Extraction

Contextual Information refers to information that characterizes the state of an Android application, such as the activities that the app launches, the system services that the app uses, the resources that the app loads, and so on. Because many studies relied solely on API Calls and permissions to distinguish between malware and benign applications, this paper combines application contextual information with API Calls and permissions to enhance the detection performance and detect malicious behavior in Android applications with high accuracy. The authors of the dataset used in this paper employed a dynamic analysis technique to run all Android application samples in a VMI-based dynamic analysis system using CopperDroid, and then recorded the results in JSON format [40]. In this paper, we parsed and analyzed the massive chunk of JSON data for each Android application (APK) using Python scripting codes to extract related contextual information that helps in improving Android malware detection. As indicated in Table 9 below, four categories of application contextual information were selected from various Android application samples. Table 10 shows the number of Android application samples from which contextual information was extracted.

Table 9. Extracted contextual features.

Android Application Contextual Information
num_services
num_receivers
num_activities
num_providers

Table 10. Number of APK samples from which the contextual features were extracted.

Android Application Category	Number of Samples from Which Contextual Features Were Extracted
Adware	1243
Banking	1878
SMS	3908
Riskware	2498
Benign	494
Total	**10,021**

3.3. Features Processing

The term "features preprocessing" refers to the act of preparing and transforming features so that they may be trained by machine learning algorithms. Features transformation, or converting features from one format to another, is one of the feature preprocessing mechanisms used in this paper. All extracted API Calls and permissions were converted to binary representation (0 or 1). In other words, if an application calls a specific API (for example, startService), the value will be 1; otherwise, it will be 0. Similarly, if an application uses a specific permission (for example, SEND SMS), the value will be 1; otherwise, it will be 0. An example of the binary representation process of the extracted API Calls and permissions features is shown in Tables 11 and 12. In this example, as shown in Table 11, the first Android application uses API Call 1 and API Call 2, the second Android application only uses API Call 3, and the third Android application uses all API Calls (1, 2 and 3). According to Table 12, the first Android application only uses permission 3, the second Android application uses all permissions (1, 2 and 3), and no permission have been used by the third Android application.

Table 11. Binary representation of the extracted API Calls.

Android Application (APK)	API Call 1	API Call 2	API Call 3
Application 1	1	0	1
Application 2	0	0	1
Application 3	1	1	1

Table 12. Binary representation of the extracted permissions.

Android Application (APK)	Permission 1	Permission 2	Permission 3
Application 1	1	0	1
Application 2	0	0	1
Application 3	1	1	1

3.4. Feature Extraction and Selection

Many application API Calls and permissions have been extracted. Therefore, this paper uses feature selection on the API Calls with permissions to eliminate duplicate and inconsistent API Calls and permissions features that reduce classification efficiency. To achieve this task, Information Gain (IG) was used. IG is a feature evaluation method that evaluates the quantity of information about the class prediction and the projected reduction in entropy if the only information provided is the presence of a feature and the accompanying class distribution [48]. IG is based on entropy, which is calculated by determining how much of a term may be used for the classification of data [49]. It is a method for selecting the optimal API Calls and permissions features that have been adopted in this paper. The results of using IG in the API Calls and permissions selection, where the top 50 ranked were selected, are shown in Figure 8. Each API Call and permissions feature has an IG value with a high value indicating a significant impact on classification.

The proposed method employed mutual information to measure the correlation between variables, where a higher value means higher dependency.

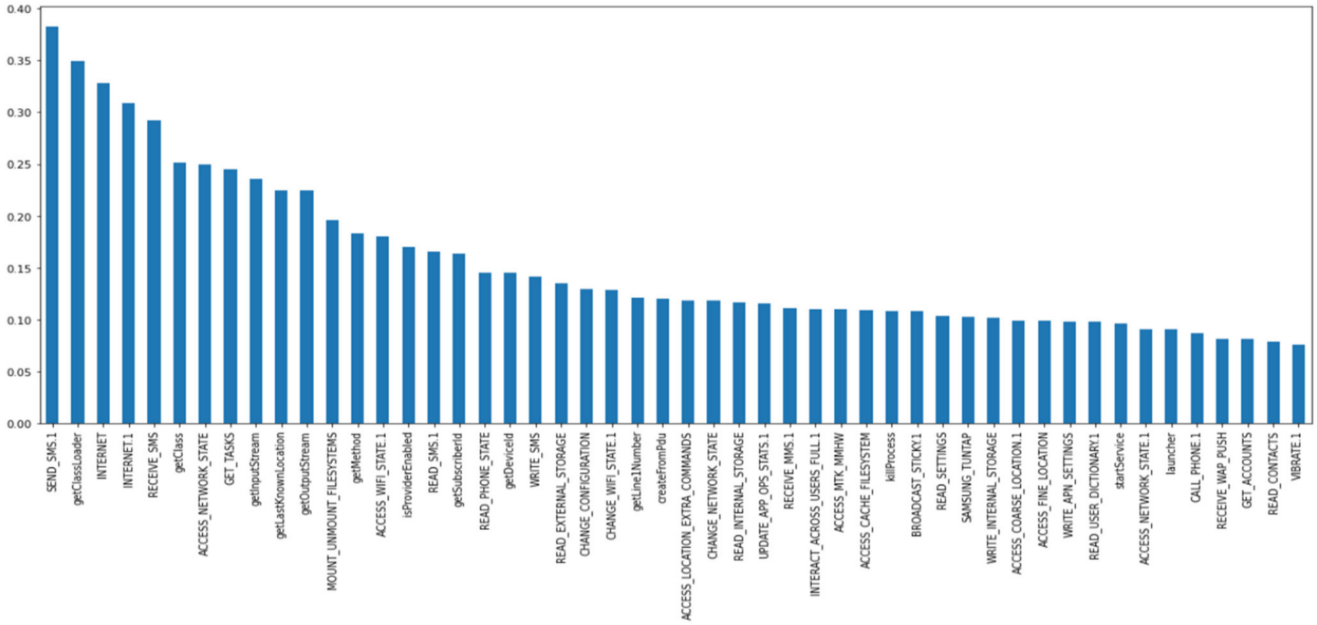

Figure 8. Top 50 ranked-selected API and permissions based on Information Gain (IG).

3.5. Machine Learning Algorithms

Supervised and unsupervised learning are two types of machine learning algorithms. This paper relies entirely on the supervised technique, which predicts the class of problem-based on related input examples of similar objects. Machine learning classifiers use many features extracted from static analysis to accomplish training for malware classification. This section discusses the various machine learning algorithms used in this paper.

3.5.1. Random Forest RF

RF is one of the most powerful and versatile supervised machine-learning algorithms for classification and regression. Random Forest fits the forest of numerous decision trees, in which the number of trees increases the robustness of the prediction, resulting in improved accuracy and avoiding overfitting [50]. The Random Forest classifier is the best machine learning discriminator between malware and benign applications, according to the results of a literature review performed on Android malware detection, as described by [51–54]. This paper sets the value of the *n_estimators* to 100, after testing 10, 50, 100, and 200.

3.5.2. Support Vector Machines SVM

SVM is a supervised learning model used for classification analysis by creating the hyperplane to divide the data into classes [55]. SVM are solid classifiers that give accurate results, but their computations are complex and they work slowly with huge datasets [56]. This paper applied the linear support vector classification SVC, with kernel = "linear"; this has more flexibility in the choice of the loss functions and is better to scale with large numbers of samples.

3.5.3. Logistic Regression LR

LR is a statistical machine learning classification technique used for predicting binary dependent variables. This classifier excels at linear problems, delivering accurate results while consuming minimal computer resources [57].

3.5.4. Naïve Bayesian NB

This is a Bayes Theorem-based probabilistic supervised machine learning algorithm that gives the conditional probability of an event A given event B, and is used for classification tasks [57]. The NB classifier is quick to calculate and can deal with noisy data, but it performs poorly when the data includes many features [56].

3.5.5. K-Nearest Neighbor KNN

The K-Nearest Neighbors (KNN) technique is a simple, easy-to-implement, and commonly used supervised machine learning algorithm that calculates the similarity between training and testing samples to handle classification and regression tasks [58]. This paper set the value of *n_neighbors* to 10.

3.5.6. Decision Trees DT

DT is a supervised machine learning algorithm and a type of tree structure classifier, which is used to accomplish classification and regression tasks. DT splits the data into subsets and presents the results as a tree with two entities: decision nodes for data splitting and leaf nodes for final decisions [59,60].

4. Experiments and Results

This section discusses the experiments conducted to evaluate the model and analyzes the results. The first subsection goes over each experiment that was performed in order to detect malicious Android apps based on their features. The second subsection summarizes all the experiments and determines which one had the highest detection accuracy.

4.1. Results and Analysis

Different metrics were computed to measure and evaluate the performance and effectiveness of each machine learning classifier in order to select the best and most accurate one. The mathematical calculations of the various evaluation metrics are shown in the following equations:

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

$$\text{Recal} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

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

where

- *TP* (True Positive) is the number of malware detections that are correctly labeled as malware,
- *TN* (True Negative) is the number at which benign is accurately identified as benign,
- *FP* (False Positive) is the number of benign that are mistakenly identified as malware, and
- *FN* (False Negative) is the number at which malware is incorrectly identified as benign.

The most intuitive evaluation metric is accuracy, which reflects the correctly predicted ratio. In some circumstances, accuracy is not always a reliable indicator; instead, alternative metrics should be evaluated, such as Precision, which is the rate of correctly predicted positive outcomes to all positive outcomes. The F1-Score (F-Measure) is the average of Precision and Recall, with Recall referring to the percentage of properly recognized outcomes across all samples [61].

4.1.1. API Calls-Based Android Malware Detection

In this part, Android API Calls features were used to classify the applications. A dataset of 15 features from 15,836 Android application APKs (11,800 malware and 4036 benign) were been used to train various machine learning classifiers. The outcomes of the classification using the Random Forest classifier, along with the confusion matrix, are shown in Figure 9.

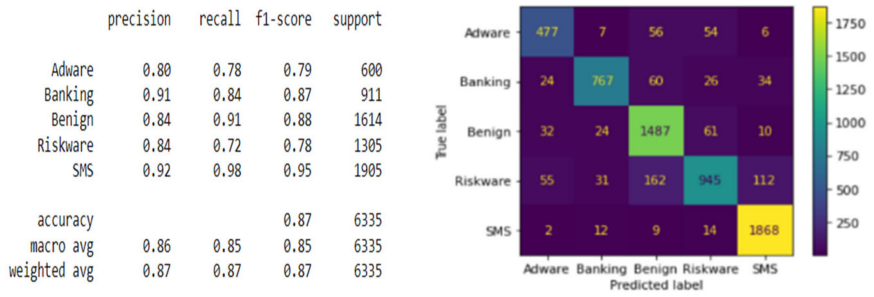

Figure 9. Android application permissions classification report and confusion matrix for the Random Forest classifier—API Calls only.

The classification report in Figure 9 shows the Random Forest classifier detection performance for each class of Android applications. For example, on adware, the Random Forest obtained 80% Precision, which means it can identify 80% of the adware dataset. Similarly, it shows a 78% Recall, which means it correctly predicts 78% of the adware. Random Forest on adware also earned a 79% F1-Score, which implies it properly predicts 79% of the data.

The number of correct and incorrect predictions for each type of Android application is shown in the confusion matrix in Figure 9. For example, the number of true adware predictions achieved by Random Forest is 477 out of 600, while the number of incorrect predictions is 123. Here are some examples of faulty predictions of adware: seven adware are falsely labeled as banking, fifty-six adware are labeled as benign, fifty-four adware are labeled as riskware, and four adware are labeled as SMS.

Table 13 and Figure 10 show how API Calls-based Android malware detection compares to different machine learning classifiers. They show that employing API Calls only to detect malicious applications is insufficient to produce accurate detection results. The Random Forest classifier had the best Accuracy, Precision, Recall, and F1-Score, but overall, this experiment performed poorly over different machine learning algorithms. The results show that the algorithm's detection accuracy is between 75% and 87%, implying that the percentage of inaccurate predictions is about 25%, which is not excellent. Furthermore, the algorithms attained a precision of 75–87%, implying that they were able to identify 75–87% of the data during testing. The different algorithms have achieved a Recall of 75–87%, which implies they properly detect 75–87% of malicious applications.

Table 13. Machine learning classifiers results—API Calls only.

	Accuracy	Precision	Recall	F1-Score
RF	0.873244	0.873244	0.873244	0.855546
LR	0.776796	0.776796	0.776796	0.738052
SVM	0.753118	0.753118	0.753118	0.708540
DT	0.871665	0.871665	0.871665	0.853826
NB	0.754854	0.754854	0.754854	0.705594
K-NN	0.865509	0.865509	0.865509	0.846865

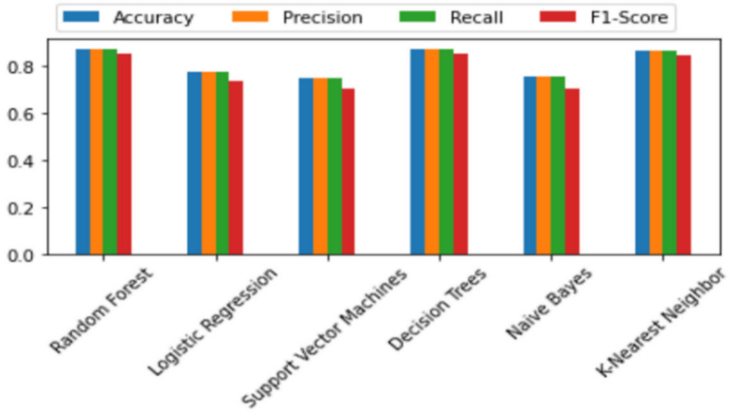

Figure 10. Other machine learning classifier results in detecting malicious Android applications—API Calls only.

4.1.2. Permissions-Based Android Malware Detection

This experiment involves extracting permissions from Android apps to train various machine-learning algorithms to classify whether the app is malicious or benign. For training, a dataset of 512 features from 16,703 Android application APKs (12,692 malicious and 4011 benign) was employed. Figure 11 shows the results of the classification using the Random Forest classifier, as well as the confusion matrix.

	precision	recall	f1-score	support
Adware	0.97	0.98	0.97	600
Banking	0.96	0.93	0.95	998
Benign	0.98	0.97	0.98	1605
Riskware	0.95	0.98	0.97	1558
SMS	0.99	0.99	0.99	1921
accuracy			0.97	6682
macro avg	0.97	0.97	0.97	6682
weighted avg	0.97	0.97	0.97	6682

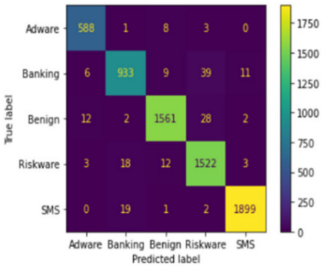

Figure 11. Android application permissions classification report and confusion matrix for Random Forest classifier—permissions only.

The detection performance of the random forest classifier for each class of Android applications is shown in Figure 11. For riskware, for example, the Random Forest achieved 95% precision, which means it can properly predict 95% of the riskware dataset, and 98% Recall, which means it can identify 98% of the riskware dataset. Random Forest on riskware received an F1-Score of 97%, indicating that it correctly predicts 97% of the data.

The confusion matrix in Figure 11 shows the number of correct and wrong predictions for each type of Android application. The number of genuine riskware predictions made by Random Forest, for example, is 1522 out of 1558, with 36 incorrect predictions. Here are some examples of riskware predictions that were incorrect: three riskware are incorrectly categorized as adware, eighteen riskware are incorrectly labeled as banking, twelve riskware are benign, and three riskware are incorrectly labeled as SMS.

The performance of different machine learning classifiers in detecting Android malware applications is shown in Table 14 and Figure 12. Table 14 and Figure 12 show that using application permissions to discriminate between malicious and benign programs is effective and yields reliable detection results. The Accuracy, Precision, Recall, and F1-Score of the Random Forest classifier were the best. The detection accuracy of the algorithms is between 95% and 97%, meaning a low percentage of incorrect predictions, which is desirable. Furthermore, the algorithms achieved a precision of 95–97% during testing, meaning that they were able to recognize 95–97% of the data. The various methods have a recall of 95–97%, indicating that they correctly detect 95–97% of the malicious programs.

Table 14. Other machine learning classifier results in Android malware detection—permissions only.

	Accuracy	Precision	Recall	F1-Score
RF	0.973810	0.973810	0.973810	0.971126
LR	0.953607	0.953607	0.953607	0.950187
SVM	0.954205	0.954205	0.954205	0.951107
DT	0.961838	0.961838	0.961838	0.955036
NB	0.801556	0.801556	0.801556	0.792031
K-NN	0.955253	0.955253	0.955253	0.949705

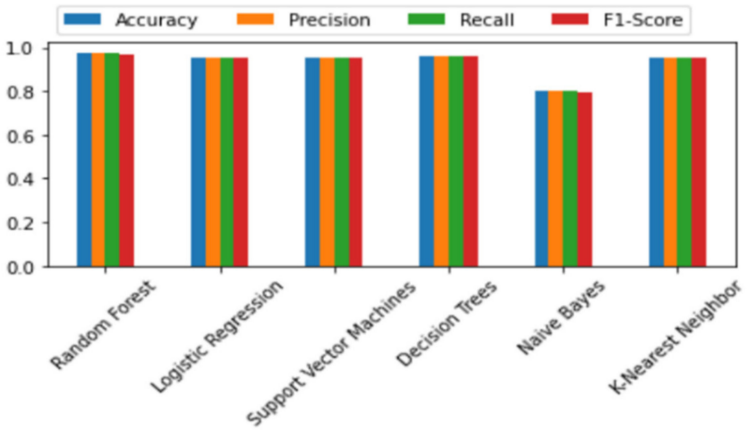

Figure 12. Other machine learning classifier results in detecting malicious Android applications—permissions only.

4.1.3. API Calls and Permissions-Based Android Malware Detection

In this experiment, we leveraged the existing API Calls and permissions in the apps and combined them so that the machine learning algorithms learn to achieve better efficiency in detecting harmful Android applications. Various machine learning algorithms have been trained with 527 features from 11,781 malware and 4008 benign real-world Android applications. The classification report, as well as the confusion matrix using the Random Forest classifier, is shown in Figure 13.

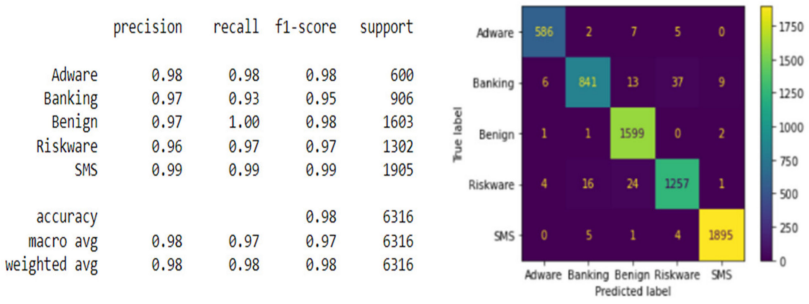

Figure 13. Android application permissions classification report and confusion matrix for the Random Forest classifier—API Calls with permissions.

Figure 13 displays the random forest classifier detection performance for each class of Android applications. For SMS, for example, the Random Forest obtained 99% precision, meaning it can correctly predict 99% of the dataset and 99% Recall, meaning it can correctly identify 99% of the dataset. The F1-Score for Random Forest on SMS was 99%, showing that it correctly predicts 99% of the data. Moreover, the number of correct and incorrect predictions for each type of Android application is shown in the confusion matrix in Figure 13. Random Forest, for example, produced 1895 genuine SMS predictions out of 1905, with only 10 wrong guesses. Here are a few examples of wrong SMS predictions: four

SMSs were wrongly classified as riskware, one SMS was incorrectly classified as benign, five SMSs were classified as banking, and no SMSs were classified as adware.

Table 15 and Figure 14 demonstrate that combining API Calls and permissions improves the detection of Android malicious apps. The Random Forest classifier has the best Accuracy, Precision, Recall, and F1-Score. The algorithm's detection accuracy ranges from 96% to 98%, indicating a high percentage of correct predictions. Furthermore, during testing, the algorithms were able to recognize 96% to 98% of the data with a Precision from 96% to 98%. The various approaches have a Recall from 96% to 98%, meaning that 96% to 98% of harmful applications are correctly detected.

Table 15. The results of using machine leaning algorithms in detecting Android malware—API Calls with permissions.

	Accuracy	Precision	Recall	F1-Score
RF	0.980526	0.980526	0.980526	0.977692
LR	0.966276	0.966276	0.966276	0.963571
SVM	0.967226	0.967226	0.967226	0.964235
DT	0.969601	0.969601	0.969601	0.965232
NB	0.852913	0.852913	0.852913	0.841969
K-NN	0.963743	0.963743	0.963743	0.958180

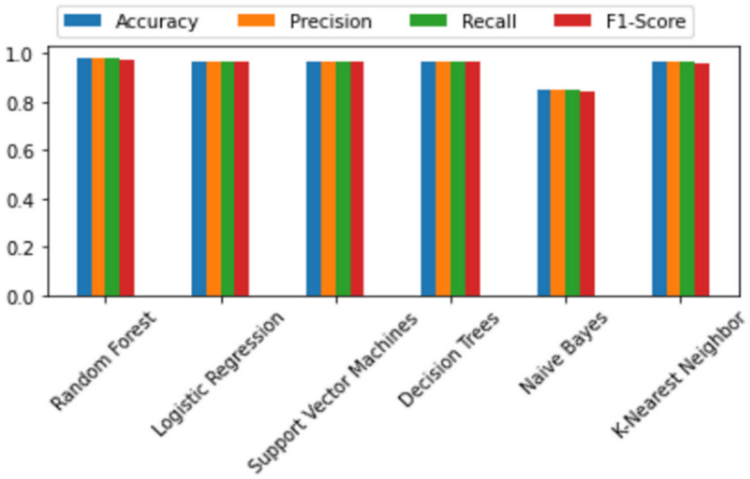

Figure 14. Other machine learning classifier performance in Android malware detection—API Calls with permissions.

4.1.4. API Calls and Permissions-Based Android Malware Detection with Feature Selection

To reduce the dimension of the feature and improve detection performance, we used the feature selection method (mutual information gain) on the combined API Calls and permissions in this experiment. The top-ranked 50 API Calls and permissions features were picked from 11,781 malware and 4008 benign real-world Android applications; refer to Figure 8 for more information. They were picked after experimenting with different feature dimensions, as shown in Figure 15. This experiment shows an increased efficiency with no discernible effect on classification accuracy.

Figure 15. Accuracy scores when training with different feature dimensions.

Figure 16 shows the confusion matrix as well as the classification report using the Random Forest classifier. The results of various machine learning classifiers in detecting fraudulent Android applications are shown in Table 16 and Figure 17.

	precision	recall	f1-score	support
Adware	0.96	0.96	0.96	600
Banking	0.97	0.91	0.94	906
Benign	0.97	1.00	0.98	1603
Riskware	0.96	0.96	0.96	1302
SMS	0.99	0.99	0.99	1905
accuracy			0.97	6316
macro avg	0.97	0.96	0.97	6316
weighted avg	0.97	0.97	0.97	6316

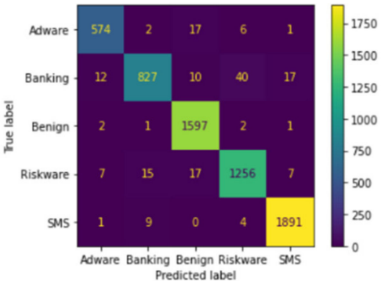

Figure 16. Classification report and confusion matrix for the Random Forest Classifier—API Calls with permissions (with feature selection).

Table 16. The results of the other machine learning algorithms in Android malware detection—API Calls with permissions (with feature selection).

	Accuracy	Precision	Recall	F1-Score
RF	0.972451	0.972451	0.972451	0.967090
LR	0.932394	0.932394	0.932394	0.923305
SVM	0.929544	0.929544	0.929544	0.918958
DT	0.960735	0.960735	0.960735	0.955430
NB	0.826314	0.826314	0.826314	0.802296
K-NN	0.955668	0.955668	0.955668	0.946952

The adoption of the feature selection technique did not increase detection accuracy, as shown in Table 16 and Figure 17. The findings of this experiment are nearly identical to those of the prior experiment (without feature selection), in which the Random Forest classifier had the best Accuracy, Precision, Recall, and F1-Score. The only advantage we can see in this experiment is that the algorithm's performance, i.e., the time it takes to predict, has improved. That is, while this experiment did not enhance the accuracy results, it improved performance.

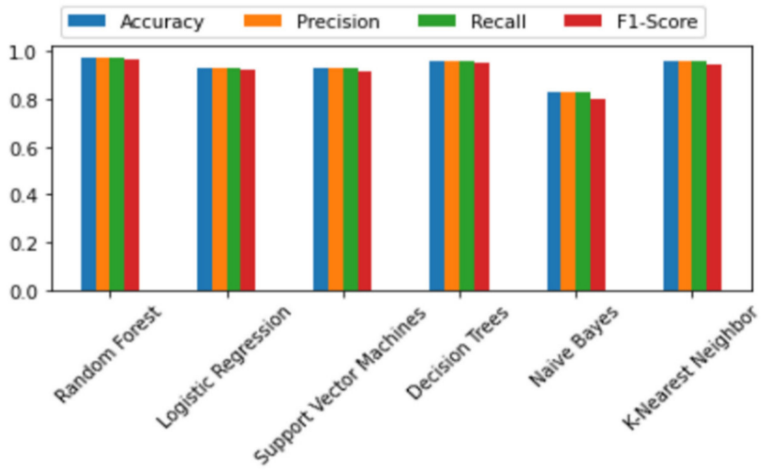

Figure 17. Machine learning classifiers results in detecting Android malware—API Calls with permissions (with feature selection).

4.1.5. API Calls and Permissions with Feature Selection and Contextual Information-Based Android Malware Detection

In this experiment, four contextual information features, num_services, num_receivers, num_activities, and num_providers, were selected and used along with the selected 50 API Calls and permissions features (used in the previous experiment) to show the effectiveness of using contextual information on the detection accuracy. These four features represent the number of times the application performs a certain activity, making them crucial and high-indication features for detecting malicious behavior and distinguishing between malware and benign applications. The number of applications used in this experiment is 842 benign samples and 5866 malware samples.

Table 17 and Figure 18 show that using contextual features along with API Calls and permissions increases detection accuracy and delivers better results. The Random Forest classifier produced the greatest results, with an accuracy of 99.4%. Figure 19 compares the accuracy results of all tested algorithms according to the use of API and permissions features only, without feature selection (527 features), with feature selection (50 features), with feature selection (50 features), and contextual features. The figure shows that using the contextual features with 50 API and permissions features only enhances the accuracy of all algorithms and outperforms the accuracy of the other models (with using feature selection and without using feature selection). In addition, an important finding is obvious in the results, which is the rise of the accuracy of NB when using contextual information. The results show that when we used the contextual features, the accuracy of NB increased sharply from 82.6% to 92.5%, which is an outstanding enhancement.

Table 17. Android malware detection results with other machine learning algorithms—API Calls, permissions, and contextual information.

	Accuracy	Precision	Recall	F1-Score
RF	0.994220	0.994220	0.994220	0.991228
LR	0.972598	0.972598	0.972598	0.971393
SVM	0.971247	0.971247	0.971247	0.970111
DT	0.978740	0.978740	0.978740	0.969944
NB	0.925197	0.925197	0.925197	0.914010
K-NN	0.975449	0.975449	0.975449	0.958354

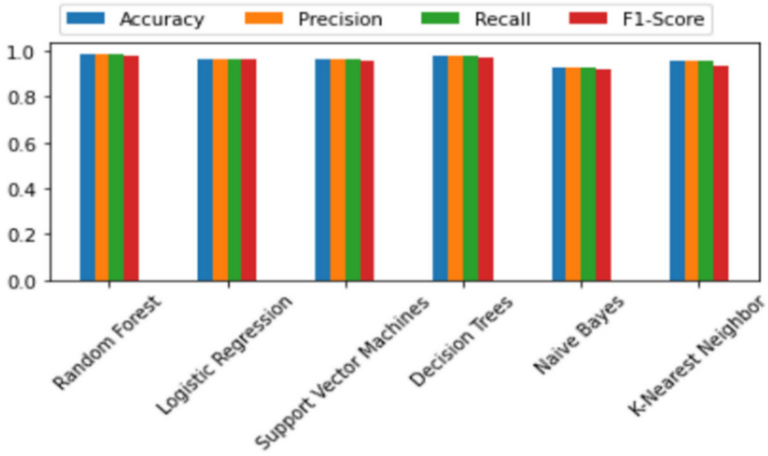

Figure 18. Detection of Android malware results with other machine learning algorithms—API Calls, permissions, and contextual information.

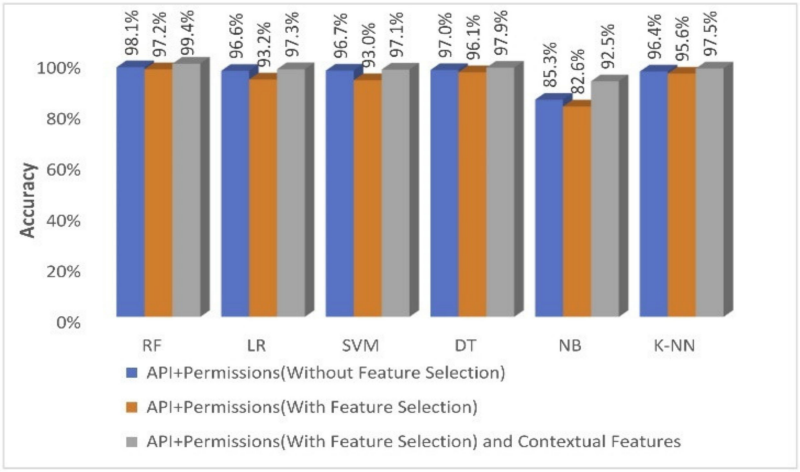

Figure 19. Enhancement of the accuracy of the proposed model using contextual features.

Figure 20 shows the confusion matrix as well as the Random Forest classifier classification results. As shown in the Figure, the detection results for each Android application category are outstanding, with a very small proportion of wrong predictions. The Random Forest algorithm was capable of successfully identifying 99.4% of malicious apps. Moreover, the number of incorrect predictions for all Android malware categories is modest, as illustrated in the confusion matrix in Figure 20. For example, just 6 predictions out of 1331 are incorrect predictions for SMS, and only 8 predictions out of 426 are wrong predictions for adware.

	precision	recall	f1-score	support
Adware	0.98	0.98	0.98	426
Banking	0.98	0.97	0.97	584
Benign	0.98	0.96	0.97	193
SMS	0.99	1.00	0.99	1337
accuracy			0.99	2540
macro avg	0.98	0.98	0.98	2540
weighted avg	0.99	0.99	0.99	2540

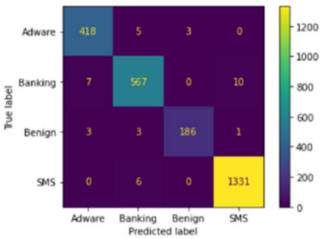

Figure 20. The classification results and confusion matrix for the Random Forest Classifier—API Calls, permissions, and contextual features.

4.2. Results Summary

The results of many experiments conducted in this paper show that using API Calls to identify suspicious applications is insufficient; the results were not accurate enough, and there were numerous incorrect predictions. Meanwhile, the results of detection based on application permissions only were better than those of using API Calls only. However, the results of Android malware detection based on API Calls and permissions together were clearly better, which was higher by 1% and reached around 98% accuracy using the Random Forest algorithm. The features selection approach did not enhance the security. However, it achieved a close accuracy of about 97.2% using 50 features only, instead of 98% using 527 features.

The interesting results were achieved when the contextual features were used along with the selected 50 API Calls and permissions. Using this combination, the highest results reached about 99.4% using the Random Forest algorithm. This proves that using the selected contextual features enhances the classification and detects Android malware with very high accuracy when it is used with API Calls and permissions features. Moreover, using contextual features enhanced Naïve Bayesian accuracy sharply from 82.6% to 92.5%.

4.3. State of Art

The proposed model in this work has achieved very high accuracy, as discussed in the previous section. To show the significance of this work, four stat of art models are considered, which are Du et al. [35], Narayanan et al. [36], Mahdavifar et al. [37], and Hadiprakoso et al. [38].

Du et al. [35] proposed a context-based approach, called FlowCog, that used natural language processing and deep learning methods to analyze the semantics and contextual information of the network flow of Android applications. Their approach used a large dataset of more than 8000 samples collected from different sources, such as the ICC-bench dataset [62], Google Play, and Drebin. The results of their approach showed that their proposed model achieved an accuracy of about 95.4%. Similarly, Narayanan et al. [36] proposed a contextual-based approach, called MKLDroid, which used a multiple kernel learning method that extracted the contextual subgraph features from the applications' dependency graphs to detect malicious code patterns. MKLDroid was applied using two datasets, Drebin and Virusshare. The authors claimed that MKLDroid achieved an accuracy of about 97%.

Mahdavifar et al. [37] and Hadiprakoso et al. [38] used the same dataset used in this work, CIC_Maldroid2020. However, both methods did not use contextual information in their approaches. The authors in [37] proposed a deep neural networks method that used about 470 features, such as system calls, binders, and composite behaviors. Their results showed that their proposed method achieved an accuracy of about 97.84%. Meanwhile, Hadiprakoso et al. [38] proposed a machine-learning model and tested several machine-learning algorithms such as SVM, KNN, RF, and XGBoost. Their model used many static and dynamic features such as API Calls, permissions, and system calls. The authors claimed that their model achieved an accuracy of about 96%.

Table 18 shows a comparison between the accuracy of the proposed model and the state-of-theart models. The table shows that the proposed work outperformed the approaches that used contextual features, which are [35] and [38], which achieved an accuracy of about 95.4% and 97%, respectively. However, these approaches did not use the MalDroid2020 dataset that was used in this work, and they did not use conventional machine learning algorithms. This proves the significance of the machine learning model that has been proposed in this work and the significance of the chosen contextual features in detecting Android malware with very high accuracy. Moreover, the table shows that the proposed work outperformed the approaches that used the same dataset in this work. These approaches used deep learning algorithms and conventional machine learning methods, but they did not use contextual features. This proves the significance of using contextual features in achieving very high accuracy in Android malware detection.

Table 18. Comparison with the state-of-the-art methods.

Work	Year	Dataset	Features	Number of Features	Methods	Accuracy
[35]	2022	Modified ICC-Bench dataset [61] Drebin [24]	Network flow semantics, such as flow contexts and inter-component communication	NA	Natural language processing and deep learning approaches	95.4%
[37]	2020	CICMalDroid2020 [39]	System calls, binders, and composite behaviors	470	Deep neural networks	97.84%
[38]	2020	Drebin [24] Malgenome [63] CICMALDROID2020 [39]	Static (permissions, API Calls, intent, command signatures, and binaries) Dynamic (system calls, binder calls, and composite behaviors)	261	SVM, KNN, MLP, RF DT, and Naïve Bayes XGBOOST	96%
[36]	2018	DREBIN [24] Virusshare [19]	contextual information (contextual subgraph features)	NA	Multiple lernel learning	97%
The proposed work	2022	CICMalDroid2020 [39]	API Calls, permissions, and contextual features	54	Random Forest	99.4%

5. Conclusions

The accuracy of Android malware detection methods using machine learning depends on the features used. API Calls and permissions are two of the most important features that are used in Android malware detection. However, most machine learning methods use these features without considering the context. This paper has shed light on the importance of using contextual features with API Calls and permissions on the detection accuracy of machine learning models. The paper has proposed a machine learning model based on the use of four important contextual features and fifty API Calls and permission features, which were extracted from a large dataset of 12,800 malicious and 4100 benign Android apps. To test the model, the paper has used several machine learning algorithms, Random Forest, SVM, Linear Regression, Naïve Bayesian, K-NN, and Decision Tree. The results have shown that when using the proposed model with API Calls and permissions only, the best results achieved were 98.1% using the Random Forest algorithm. Moreover, the results have shown that after applying the Information Gain selection algorithm to select the best relevant features, only 50 features out of 527 can be used to achieve a close accuracy of about 97.2%. Furthermore, the results have shown that using contextual features along with the 50 API Calls and permissions achieved a very high accuracy of about 99.4% when using the Random Forest algorithm. In addition, the results have shown that the most affected algorithm by using contextual features was the Naïve Bayesian algorithm, where its accuracy raised sharply from 82.6% to 92.5%, which is an interesting change for the Naïve Bayesian. Moreover, this paper considered four important methods as state-of-theart models. The comparison has shown that the proposed model outperformed the state-of-the-art models.

Author Contributions: All authors provided equal contributions. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is already available by previous authors, not new dataset.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Singh, R. An Overview of Android Operating System and Its Security Features. *J. Eng. Res. Appl.* **2014**, *4*, 519–521.
2. Mobile Security Review 2021—AV-Comparatives. Available online: https://www.av-comparatives.org/tests/mobile-security-review-2021/#google-android (accessed on 9 September 2022).

3. Singh, P.; Tiwari, P.; Singh, S. Analysis of Malicious Behavior of Android Apps. *Procedia Comput. Sci.* **2016**, *79*, 215–220. [CrossRef]
4. 2021 Mobile Malware Evolution: Fewer Attacks, Escalating Dangers. Available online: https://www.techrepublic.com/article/2021-mobile-malware-evolution-fewer-attacks-escalating-dangers/ (accessed on 9 September 2022).
5. Sk, H.K. A Literature Review on Android Mobile Malware Detection using Machine Learning Techniques. In Proceedings of the 6th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 29–31 March 2022; pp. 986–991. [CrossRef]
6. Salah, Y.; Hamed, I.; Nabil, S.; Abdulkader, A.; Mostafa, M. Mobile Malware Detection: A Survey. *Int. J. Comput. Sci. Inf. Secur.* **2019**, *17*, 56–65.
7. Moses, A.; Morris, S. Analysis of Mobile Malware: A Systematic Review of Evolution and Infection Strategies. *J. Inf. Secur. Cybercrimes Res.* **2021**, *4*, 103–131. [CrossRef]
8. Kambar, M.E.; Esmaeilzadeh, A.; Kim, Y.; Taghva, K. A Survey on Mobile Malware Detection Methods using Machine Learning. In Proceedings of the IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 26–29 January 2022; pp. 0215–0221. [CrossRef]
9. Yerima, S.Y.; Alzaylaee, M.K. Mobile Botnet Detection: A Deep Learning Approach Using Convolutional Neural Networks. *arXiv* **2020**, arXiv:2007.00263.
10. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. Dynalog: An automated dynamic analysis framework for characterizing android applications. In Proceedings of the International Conference On Cyber Security and Protection of Digital Services (Cyber Security), London, UK, 13–14 June 2016; pp. 1–8.
11. Kosmidis, K.; Kalloniatis, C. Machine learning and images for malware detection and classification. ACM International Conference Proceeding Series. In Proceedings of the 21st Pan-Hellenic Conference on Informatics, Larissa, Greece, 28–30 September 2017.
12. Chumachenko, K. Machine Learning Methods for Malware Detection and Classification. Bachelor's Thesis, South-Eastern Finland University of Applied Sciences, Kouvola, Finland, 2017.
13. Narayanan, A.; Chandramohan, M.; Chen, L.; Liu, Y. Context-Aware, Adaptive, and Scalable Android Malware Detection Through Online Learning. *IEEE Trans. Emerg. Top. Comput. Intell.* **2017**, *1*, 157–175. [CrossRef]
14. Kapratwar, A.; Di Troia, F.; Stamp, M. Static and dynamic analysis of android malware. In Proceedings of the 3rd International Conference on Information Systems Security and Privacy, Porto, Portugal, 19–21 February 2017.
15. Bhatia, T.; Kaushal, R. Malware detection in android based on dynamic analysis. In Proceedings of the 2017 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), London, UK, 19–20 June 2017.
16. Amamra, A.; Talhi, C.; Robert, J.M. Smartphone malware detection: From a survey towards taxonomy. In Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software, Fajardo, PR, USA, 16–18 October 2012.
17. Larose Daniel, T. *Discovering Knowledge in Data: An Introduction to Data Mining*; Wiley: Hoboken, NJ, USA, 2014; pp. 174–179. ISBN 9780470908747.
18. Le, N.C.; Nguyen, T.M.; Truong, T.; Nguyen, N.D.; Ngo, T. A Machine Learning Approach for Real Time Android Malware Detection. In Proceedings of the 2020 RIVF International Conference on Computing and Communication Technologies (RIVF), Ho Chi Minh City, Vietnam, 14–15 October 2020.
19. Virusshare. Available online: https://virusshare.com/ (accessed on 30 August 2022).
20. Koodous: Collective Intelligence against Android Malware. Available online: https://koodous.com/ (accessed on 30 August 2022).
21. Kavediya, V.; Sadashiv, M.; Mhaskar Kulkarni, K.; Prabhu, S.; Balbudhe, K. Android Malware Detection using Machine learning technique. *Int. J. Res. Anal. Rev.* **2020**, *7*, 777–780.
22. Han, H.; Lim, S.; Suh, K.; Park, S.; Cho, S.J.; Park, M. Enhanced android malware detection: An SVM-based machine learning approach. In Proceedings of the 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), Busan, Korea, 19–22 February 2020.
23. Li, Y.; Jang, J.; Hu, X.; Ou, X. Android Malware Clustering through Malicious Payload Mining. In *International Symposium on Research in Attacks, Intrusions, and Defenses*; Springer: Cham, Switzerland, 2017.
24. Arp, D.; Spreitzenbarth, M.; Hübner, M.; Gascon, H.; Rieck, K. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the Symposium on Network and Distributed System Security (NDSS), San Diego, CA, USA, 23–26 February 2014.
25. APKPure. Available online: https://m.apkpure.com/ (accessed on 30 August 2022).
26. Mantoo, B.A.; Khurana, S.S. Static, Dynamic and Intrinsic Features Based Android Malware Detection Using Machine Learning. *Lect. Notes Electr. Eng.* **2020**, *597*, 31–45.
27. Kang, H.J.; Jang, J.W.; Mohaisen, A.; Kim, H.K. AndroTracker: Creator Information based Android Malware Classification System. In Proceedings of the 15th International Workshop in Information Security Applications, Jeju Island, Korea, 25–27 August 2014.
28. Google Paly Store. Available online: https://play.google.com/store/apps (accessed on 30 August 2022).
29. Fang, Y.; Gao, Y.; Jing, F.; Zhang, L. Android Malware Familial Classification Based on DEX File Section Features. *IEEE Access* **2020**, *8*, 10614–10627. [CrossRef]
30. Vasan, D.; Alazab, M.; Wassan, S.; Naeem, H.; Safaei, B.; Zheng, Q. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Comput. Netw.* **2020**, *171*, 107–138. [CrossRef]

31. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B. Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, Pittsburgh, PA, USA, 20 July 2011.
32. Ünver, H.M.; Bakour, K. Android malware detection based on image-based features and machine learning techniques. *SN Appl. Sci.* **2020**, *2*, pp. 1–15. [CrossRef]
33. Nasri, N.; Razak, M.A. Android Malware Detection System using Machine Learning. *Int. J. Adv. Trends Comput. Sci. Eng.* **2020**, *9*, 327–333. [CrossRef]
34. Ali, W.; Abdulghafor, R.; Abdullah, T. Empirical Study on Intelligent Android Malware Detection based on Supervised Machine Learning. *Int. J. Adv. Comput. Sci. Appl.* **2020**, *11*, 215–224.
35. Du, X.; Pan, X.; Cao, Y.; He, B.; Fang, G.; Chen, Y.; Xu, D. FlowCog: Context-aware Semantic Extraction and Analysis of Information Flow Leaks in Android Apps. In *IEEE Transactions on Mobile Computing*; IEEE: Piscataway, NJ, USA, 2022.
36. Narayanan, A.; Chandramohan, M.; Chen, L.; Liu, Y. A multi-view context-aware approach to Android malware detection and malicious code localization. *Empir. Softw. Engg.* **2018**, *23*, 3. [CrossRef]
37. Mahdavifar, S.; Kadir, A.; Fatemi, R.; Alhadidi, D.; Ghorbani, A. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In Proceedings of the IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Calgary, AB, Canada, 17–22 August 2020.
38. Hadiprakoso, R.B.; Kabetta, H.; Buana, I. Hybrid-Based Malware Analysis for Effective and Efficiency Android Malware Detection. In Proceedings of the 2nd International Conference on Informatics, Multimedia, Cyber, and Information System, ICIMCIS 2020, Jakarta, Indonesia, 19–20 November 2020.
39. Mahdavifar, A.; Abdul Kadir, R.; Fatemi, D.; Alhadidi, A. Ghorbani, Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In Proceedings of the 18th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC), Calgary, AB, Canada, 17–24 August 2020.
40. MalDroid 2020, Canadian Institute for Cybersecurity. Available online: https://www.unb.ca/cic/datasets/maldroid-2020.html (accessed on 9 September 2022).
41. Karbab, E.; Debbabi, M.; Derhab, A.; Mouheb, D. MalDozer: Automatic framework for android malware detection using deep learning. *Digit. Investig.* **2018**, *24*, S48–S59. [CrossRef]
42. VirusTotal. Available online: https://www.virustotal.com/ (accessed on 30 August 2022).
43. Parkour, M. Contagio Mini-Dump. Available online: http://contagiominidump.blogspot.it/ (accessed on 30 August 2022).
44. Saint Yen, Y.; Sun, H.M. An Android mutation malware detection based on deep learning using visualization of importance from codes. *Microelectron. Reliab.* **2019**, *93*, 109–114.
45. Thomas, T.; Vijayaraghavan, A.P.; Emmanuel, S. *Machine Learning Approaches in Cyber Security Analytics*; Springer: Singapore, 2019; pp. 1–209.
46. Welcome to Androguard's Documentation! Androguard 3.4.0 Documentation. Available online: https://androguard.readthedocs.io/en/latest/ (accessed on 9 September 2022).
47. GitHub-Androguard/Androguard: Reverse Engineering, Malware and Goodware Analysis of Android Applications ... and More. Available online: https://github.com/androguard/androguard (accessed on 9 August 2022).
48. Sharma, A.; Dash, S.K. Mining API Calls and Permissions for Android Malware Detection. *Lect. Notes Comput. Sci.* **2014**, *8813*, 191–205.
49. Lei, S. A feature selection method based on information gain and genetic algorithm. In Proceedings of the 2012 International Conference on Computer Science and Electronics Engineering, Hangzhou, China, 23–25 March 2012.
50. Ho, T.K. Random decision forests. In Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, 14–16 August 1995.
51. Pandey, S.; Lal, R. Opcode-Based Android Malware Detection Using Machine Learning Techniques. *J. Innov. Eng. Techn.* **2021**, *5*, 56–61.
52. Mohamad Arif, J.; Razak, M.F.; Awang, S.; Tuan Mat, S.R.; Ismail, N.S.N.; Firdaus, A. A static analysis approach for Android permission-based malware detection systems. *PLoS ONE* **2021**, *16*, e0257968. [CrossRef]
53. Singh, D.; Karpa, S.; Chawla, I. Emerging Trends in Computational Intelligence to Solve Real-World Problems' Android Malware Detection Using Machine Learning. In *International Conference on Innovative Computing and Communications*; Springer: Singapore, 2022.
54. Muzaffar, A.; Ragab Hassen, H.; Lones, M.A.; Zantout, H. Android Malware Detection Using API Calls: A Comparison of Feature Selection and Machine Learning Models. *Lect. Notes Networks Syst.* **2022**, *378*, 3–12.
55. Cortes, C.; Vapnik, V.; Saitta, L. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
56. Dutt, S.; Chandramouli, S.; Kumar Das, A. *Machine Learning*, 1st ed.; Pearson Education: Bengaluru, India, 2018.
57. Agrawal, P.; Trivedi, B. Machine Learning Classifiers for Android Malware Detection. *Adv. Intell. Syst. Comput.* **2021**, *1174*, 311–322.
58. Abu Alfeilat, H.A.; Hassanat, A.B.; Lasassmeh, O.; Tarawneh, A.S.; Alhasanat, M.B.; Eyal Salman, H.S.; Prasath, V.S. Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review. *J. Big Data* **2019**, *7*, 221–248. [CrossRef] [PubMed]

59. Zulkifli, A.; Hamid, I.R.A.; Shah, W.M.; Abdullah, Z. Android Malware Detection Based on Network Traffic Using Decision Tree Algorithm. *Adv. Intell. Syst. Comput.* **2018**, *700*, 485–494.
60. Kouliaridis, V.; Kambourakis, G. A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection. *Information* **2021**, *12*, 185. [CrossRef]
61. Powers, D. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. *Mach. Learn. Technol.* **2008**, *2*, 37–63.
62. Icc-Bench. Available online: https://github.com/fgwei/ICC-Bench (accessed on 15 September 2022).
63. Malgenome Project. Available online: http://www.Malgenomeproject.org (accessed on 15 September 2022).