Received 17 June 2022, accepted 3 July 2022, date of publication 11 July 2022, date of current version 18 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3189645

RESEARCH ARTICLE

PAIRED: An Explainable Lightweight Android Malware Detection System

MOHAMMED M. ALANI[1,2], (Senior Member, IEEE), AND ALI ISMAIL AWAD[3,4,5,6], (Senior Member, IEEE)

[1]Department of Computer Science, Toronto Metropolitan University, Toronto, ON M5B 2K3, Canada
[2]School of IT Administration and Security, Seneca College of Applied Arts and Technology, Toronto, ON M2J 2X5, Canada
[3]College of Information Technology, United Arab Emirates University, Al Ain 17551, United Arab Emirates
[4]Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, 971 87 Luleå, Sweden
[5]Electrical Engineering Department, Faculty of Engineering, Al-Azhar University, Qena 83513, Egypt
[6]Centre for Security, Communications and Network Research, University of Plymouth, Plymouth PL4 8AA, U.K.

Corresponding author: Ali Ismail Awad (ali.awad@uaeu.ac.ae; ali.awad@ltu.se)

ABSTRACT With approximately 2 billion active devices, the Android operating system tops all other operating systems in terms of the number of devices using it. Android has gained wide popularity not only as a smartphone operating system, but also as an operating system for vehicles, tablets, smart appliances, and Internet of Things devices. Consequently, security challenges have arisen with the rapid adoption of the Android operating system. Thousands of malicious applications have been created and are being downloaded by unsuspecting users. This paper presents a lightweight Android malware detection system based on explainable machine learning. The proposed system uses the features extracted from applications to identify malicious and benign malware. The proposed system is tested, showing an accuracy exceeding 98% while maintaining its small footprint on the device. In addition, the classifier model is explained using Shapley Additive Explanation (SHAP) values.

INDEX TERMS Android, malware, malware detection, XAI, machine learning.

I. INTRODUCTION

In 2008, the Android operating system was first introduced as an open-source project in its current form. Since then, it has been gaining wide popularity among users because of its customizability and low hardware requirements. With the number of active Android devices approaching 2 billion in 2021, Android has remained at the lead of operating systems used worldwide [1]. Figure 1 illustrates the rapid growth of the Android OS adoption. This noticeable growth is a result of the wide adoption of the operating system in mobile phones, Internet of Things (IoT), industrial IoT (IIoT), connected vehicles, and smart home appliances.

This wide growth of the Android operating system comes with vast security challenges. One challenge that is particularly important is the emergence of malicious applications or those containing or implanting malware. Google Play Store, which is the main platform from which Android users can download applications, rapidly grew from 16,000 applications in December 2009 to 2,797,581 applications in August 2021 [2]. This rapid growth witnessed periods of low-security checks on the applications included in the store, resulting in many large incidents where malware-infested applications have been downloaded by millions of users [3]. In these incidents, the play store protection may have failed to detect malware instances and potentially unwanted programs or these applications did not go through the checking process at all.

According to [3], Google Play Store provides only 87% of downloaded Android applications. Other sources include alternative markets, older application backups, and direct downloads using browsers, among other sources. This 13% mostly comes from non-reliable sources. In many cases, attackers market these non-play store downloads as a free version of a normally paid application. They also try to masquerade their malicious applications as useful ones. In a more recent incident, attackers created applications that can pass the security check of Google Play Store. They do not

The associate editor coordinating the review of this manuscript and approving it for publication was Mehul S. Raval.

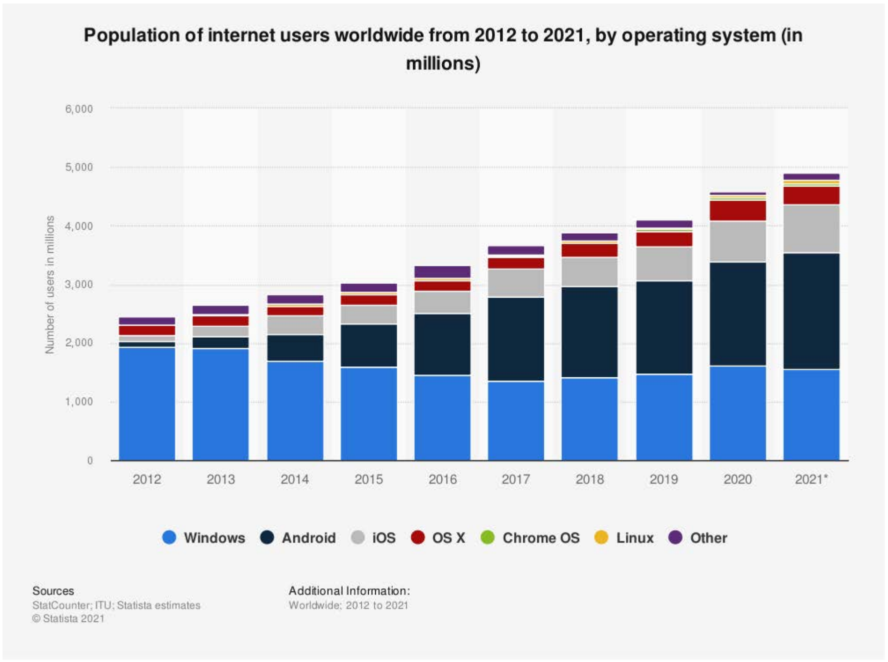

FIGURE 1. **Population of internet users worldwide from 2012 to 2021 by operating system [1].**

contain malware, but they find ways to download and inject malware into Android devices after installation [4]. In that particular incident, a banking trojan attack was delivered by an application that was downloaded for over 300,000 times.

A study in [5] reported that only 35% of Android users read all the permissions an application asks for before installation. Only 77% have at least once refused to install an application because of the permissions it asks for. These numbers indicate a serious gap in user awareness of the risks accompanying the installation of applications without reading and understanding what these applications will have access to.

The above mentioned security challenges set the case for the urgent need for a robust and reliable malware detection system. The malware detection solution must be lightweight to fit into the wide range of Android hardware specifications ranging from octa-core smartphones to simple network-connected photo displays.

A. RESEARCH CONTRIBUTIONS

This study presents ''PAIRED'' as an explainable lightweight malware detection system based on machine learning. Our research focuses on the production of a machine learning solution that does not demand high resources and relies only on a minimal number of features to be extracted to maintain a low impact on resources while maintaining high malware detection accuracy.

The contributions of this research can be summarized in the following points:

1) Feature Selection: Selecting the minimum possible number of critical features that can provide high accuracy, while reducing the weight of the malware detection solution to the minimum. The use of recursive feature elimination (RFE) ensures that the number of features used in the detection process is kept at minimum. In addition, it also reduces the number of features that need to be acquired and extracted at the data acquisition phase.
2) Model Explainability: Explaining the selected features using Shapley additive explanation (SHAP) values to ensure that the high accuracy of the classifier originates from explainable conditions.
3) Lightweight: Building a high-accuracy machine-learning based malware detection system for Android devices, that relies on static features extracted from installed applications without the need to run these applications.
4) New Dataset: Producing a reduced version of the dataset with 35 effective features that can be used in future research on malware detection in the Android ecosystem.

B. PAPER LAYOUT

The rest of the paper is organized as follows: Section II discusses the most significant malware detection solutions in previous works. This section will be divided into two parts; classical malware detection, and machine-learning-based malware detection. Section III explains the dataset used in training and testing of the machine-learning model. The design of the proposed system, ''PAIRED'', includes its practical aspects is described in Section IV. Section V shows the steps of implementation along with the testing results. The proposed model's explainability using SHAP is introduced in Section VI. Section VII will present discussions and comparisons with previous works, Finally, conclusions and suggestions for future research directions are written in Section VIII.

II. RELATED WORKS

According to [6], a new malicious Android application is created every 10 s. This section reviews previous works that performed machine learning- and non-machine learning-based research on Android malware detection.

A. NON-MACHINE LEARNING-BASED MALWARE DETECTION

In 2019, Arora *et al.* [7] introduced PermPair that aimed to identify pairs of permissions, which can be associated with malware. The proposed detection model constructs and compares graphs for malware and normal samples by extracting permission pairs from the manifest file of an application. The testing results of the proposed model showed 95.44% accuracy when compared to other similar approaches and favorite mobile anti-malware applications. The study also proposed an edge elimination algorithm that removes 7% and 41% of unnecessary edges from malware and normal graphs, respectively. This led to a minimum space utility and a 28% decrease in the detection time.

In 2020, Taheri *et al.* [8] presented four malware detection methods using the Hamming distance to find similarities between the samples of first nearest neighbors, all nearest neighbors, weighted all nearest neighbors, and k-medoid-based nearest neighbors. Testing was performed on three datasets, including benign and malware Android applications like Drebin, Contagio, and Genome. The test results showed more than 90% accuracy. In some cases (i.e., considering API features), the accuracy was more than 99% and comparable with that of existing state-of-the-art solutions.

In the same year, Han *et al.* [9] presented the feature transformation-based Android malware detector (FARM). The FARM takes the well-known features for Android malware detection and introduces three new types of feature transformations that irreversibly transform these features into a new feature domain. The detector was initially tested on six Android classification problems separating goodware and other malware from three malware classes, that is, rooting malware, spyware, and banking trojans. Han *et al.* also proposed three realistic attacks on the FARM and showed that it is highly robust to attacks in all classification problems. The detector automatically identified two malware samples that were not previously classified as rooting malware by any of the 61 anti-viruses on VirusTotal. These samples were reported to Google's Android Security Team, who confirmed the findings.

B. MACHINE LEARNING-BASED MALWARE DETECTION

Many machine learning-based solutions have generally been proposed for malware detection. In this subsection, we will review a few papers within the area of the proposed system.

In 2018, Li *et al.* [6] presented a machine learning Android malware detection system based on application permissions. The proposed solution, called the Significant Permission Identification, was based on the permission usage analysis for detecting malicious applications and malware. As an alternative to extracting and analyzing all Android permissions, the proposed system mined the permission data to identify the most significant permissions that can be effective in distinguishing benign and malicious applications. The study proposed the selection of 22 significant permissions. System testing achieved 90% precision, recall, accuracy, and F-measure.

Also in 2018, Yerima and Sezer [10] presented a multi-level machine learning classifier combination framework to detect Android malware. They introduced the dataset that we will be using in building our proposed system (i.e., PAIRED). The proposed framework, called DroidFusion, generates a model by training base classifiers at a lower level. It then applies a set of ranking-based algorithms on their predictive accuracies at a higher level to derive a final classifier. The DroidFusion method enables the fusion of ensemble learning algorithms for improved accuracy. The testing results of DroidFusion showed that it can outperform stacked generalization, a well-known classifier fusion method that employs a meta-classifier approach at its higher level.

In the same year, Firdaus *et al.* [11] presented another machine learning-based Android malware detection system that uses genetic search (i.e., a search based on a genetic algorithm) to select features among the 106 strings generated from the application static analysis. These features would feed into five different classifiers, namely Naive Bayes, functional trees, J48, random forest, and multilayer perceptron. The testing results showed that the functional trees classifier had the highest accuracy (i.e., 95%) and true positive rate (i.e., 96.7%) using six features.

Cai *et al.* in 2018 produced an app-profiling-based Android malware detection system, called DroidCat [12]. DroidCat was introduced as a dynamic app-classification technique that inspects method calls and inter-component communication without involving permissions, app resources, or system calls while fully handling reflection. DroidCat consistently achieved 97% F1 measure accuracy for classifying apps during the testing phase.

In 2019, Xiao *et al.* presented a deep learning-based Android malware detection method [13]. The proposed model examined semantic information in system call sequences as the natural language, treated one system call sequence as a sentence in the language, and constructed a classifier based on the long short-term memory language model. The experiments showed that this approach can achieve high efficiency and high recall (96.6%) with a false positive rate (9.3%).

In the same year, Lei *et al.* [14] also presented an event-aware Android malware detection system that exploited behavioral patterns in different events to effectively detect new malware based on the insight that events can reflect the applications' possible running activities. This system was called EveDroid. The proposed system used an event group to describe the applications' behaviors at the event level, capturing a higher level of semantics than at the API level. EveDroid was based on a neural network specifically designed to aggregate multiple events and automatically mine the semantic relationship among them. The testing results showed a high F_1 score of EveDroid (i.e., exceeding 99%).

Li *et al.* in 2019 also introduced a machine learning-based malware classifier based on the factorization machine architecture and the extraction of Android app features from manifest files and source code [15]. The proposed classifier was tested with the DREBIN and AMD [16] datasets and achieved accuracies of 100% and 99.22%, respectively. The proposed system had fast training by a factor of 50.

In 2020, Millar *et al.* presented an Android malware detection model using a deep learning discriminative adversarial network (DAN) that classifies both obfuscated and unobfuscated apps as either malicious or benign [17]. The proposed method was robust against a selection of four real-world obfuscation techniques. In addition, it also used three feature sets (i.e., raw opcodes, permissions, and API calls) combined in a multi-view deep learning architecture to increase this obfuscation resilience. The testing results showed that the proposed model achieves an average F_1 score of 0.973.

In the same year, Kouliaridis *et al.* [18] presented Androtomist, a tool capable of symmetrically applying the static and dynamic analyses of applications on the Android platform to detect malware. Androtomist utilized the features stemming from the static analysis along with dynamic instrumentation to dissect applications and decide if they are benign or not. It focused on anomaly detection using machine learning and autonomously conducted a signature-based detection. The proposed method assessed the detection accuracy of Androtomist against three different popular malware datasets and a handful of machine learning classifiers. The paper introduced an ensemble approach by separately averaging the output of all base classification models per malware instance and provided an insight on the most influencing features regarding the classification process.

In 2021, Hei *et al.* presented a new malware detection framework for evolutionary Android applications, called Hawk [19]. Hawk modeled Android entities and behavioral relationships as a heterogeneous information network, exploiting its semantic meta-structures for specifying implicit higher-order relationships. The experiments examined more than 80,860 malicious and 100,375 benign applications developed over a period of 7 years. Hawk yielded a high detection accuracy against baselines and spent 3.5 ms in average to detect an out-of-sample application.

In 2021 as well, Frenklach *et al.* presented a static Android application analysis method that relied on an app similarity graph [20]. The proposed method was demonstrated on the Drebin benchmark in both balanced and unbalanced settings, on a brand new VTAz dataset from 2020, and on a dataset of approximately 190,000 applications provided by VirusTotal, achieving 0.975 accuracy in balanced settings and 0.987 area under the curve (AUC) score. The analysis and classification times of the proposed methods were from 0.08 to 0.153 s/app.

Şahin *et al.* presented, in 2021, a machine-learning based malware detection system that utilizes deep neural networks [21]. The proposed system relies on features extracted from the application permissions. A linear-regression based method was used for feature selection that ended up selecting 27 effective features to be used in malware detection. The proposed system produced a F_1 score of 0.961. However, with it utilizing multi-layer perceptron as the classifier type, it does not qualify as a lightweight solution due to the intensive processing requirements needed by neural networks in comparison to other machine learning classifiers. The same authors presented in another research publication a filter-based feature selection method for Android malware detection [22]. Within this study, the authors focus on building a classifier with a smaller number of features. However, the features were all extracted from application permissions only. This neglects important features that can be extracted from malware behavior such as API call signatures, intents, and command signatures.

Mahindru and Sangal presented, in 2021, another permissions-focused Android malware detection system [23]. The proposed system utilizes least square support vector machine along with ten feature selection approaches. The proposed system was tested and presented an accuracy of 98.8% with a detection time of 12 seconds. The detection time is indicative of very slow processing when compared to our proposed system.

Although many published works are related to malware detection in Android using machine learning-based techniques, an in-depth analysis of the state-of-the-art revealed the need for a feature- and resource-optimized malware detection tool that can achieve high malware detection accuracy. The present study addresses this problem by presenting PAIRED as a robust, reliable, and lightweight malware detector for Android.

III. DATASET

Android applications are written in Java and executed within a custom Java virtual machine. Each application package is contained in a Java Archive (JAR) file with the extension of the Android Application Package, known as the APK. Android applications have four fundamental building blocks, namely activities, services, broadcast receivers, and content providers. These components are declared in the application manifest file for use. The communication between these components is achieved by using intents and intent filters. Intents are messaging objects that can be used to request actions from other application components. Intent filters are expressions declared in the application manifest file that specify the intent type a component will receive. Application components interact via the intent method; hence, both components and their communication intents must be analyzed for security concerns [15].

TABLE 1. Feature types in the Drebin-215 dataset.

Feature type	Number of features
API call signature	73
Manifest permission	113
Command signature	06
Intent	23
Total	215

In this work, we used the Drebin-215 dataset first introduced in 2014 by Arp *et al.* [24]. As discussed in Section II, Drebin-215 is a well-known dataset used in many machine learning-based malware detection systems. The dataset was generated by extracting 215 features using the static analysis of malicious applications. The dataset included 15,036 instances, each of which represents an application. A total of 9476 instances in this dataset were benign. The remaining 5560 were malicious. The 215 features came from different extraction sources (Table 1).

For testing purposes, we will also use two other datasets to ensure that the trained model generalizes well beyond its training dataset. The first testing dataset is Malgenome-215 [25], a well-known dataset published in 2015 that is used in creating machine learning classifiers. The features included in Malgenome-215 are similar to those in Drebin-215. However, it comprised 3799 instances, out of which 1260 were malicious, and 2593 were benign.

The second testing dataset is CICMalDroid2020 [26]. This dataset is a newer dataset that was published in 2020. It included many features extracted by different means including the features that were selected in this research. The dataset includes features extracted from 11,598 instances (1,795 benign, and 9,803 malicious). The dataset included different types of malware such as: adware, banking malware, SMS malware, and riskware. The extracted features were preprocessed to prepare them for testing using our trained classifier.

IV. PROPOSED SYSTEM OVERVIEW

This section presents the proposed system; PAIRED, as a lightweight malware detection system for Android devices. The proposed system was based on the extraction of static features from the installed applications to evaluate them. The choice of static features was based on the fact that we wanted the proposed system to deliver faster decisions and not wait for the application to act maliciously because, by then, the damage might have already been done. The dynamic analysis can be of high risk in malicious applications that achieve the rooting of Android devices because the device might have already been rooted by the time the application is detected.

The design principles behind PAIRED are as follows:

1) High accuracy: This is achieved by properly training the machine learning classifier with a smaller number of features having the highest classification effectiveness.
2) Being lightweight: This is achieved by reducing the number of features by a factor of 84%. The processing power and the memory requirements for extracting the features and running the classifier will be significantly reduced while maintaining the accuracy. Another contribution toward making the system lightweight is using a classifier that is not resource-intensive.
3) High efficiency: Reducing the number of features means that the time needed to make a decision would be dramatically reduced. Most importantly, the time needed for feature extraction is also significantly reduced.
4) Creating a generalizable classifier: The trained classifier model will be tested using a dataset subset that was not used in training. The model will also be tested using a different dataset that has never been used in training to assure generalization.

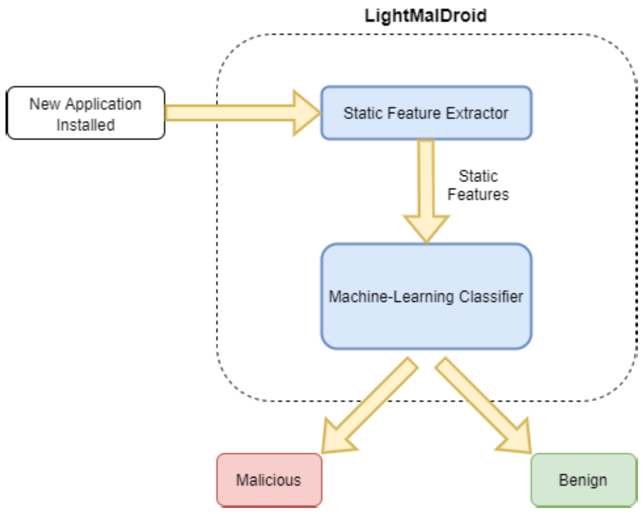

FIGURE 2. Overview of the PAIRED architecture.

Figure 2 presents an overview of the proposed system architecture. In Figure 2, the proposed system operation is triggered whenever a new application is installed on the Android device. Each application consists of an APK file, which is a zipped file consisting of the application source code, resources, assets, and manifest file. The source code is encoded as a Dalvik Executable (DEX) file that can be interpreted using the Dalvik Virtual Machine. The manifest file comprises a number of declarations and specifications. The other resources may contain images, and HTML files.

DEX files are a compiled binary executable code, and features cannot be readily extracted from them directly. Therefore, they must be decompiled into other formats that can be read and interpreted (e.g., Smali code or Java code). The Smali code is an intermediate form decompiled from DEX files and essentially the assembly code format of an application. Only after the APK files have been decompiled can features be extracted from them. The static feature extractor starts extracting the needed features and passes them to the classifier. The classifier then takes the input and generates a prediction of whether the installed application is malicious or benign. A malicious application can either be sandboxed or removed immediately.

In newer versions of Android, Dalvik was replaced with Android Runtime (ART). ART is the managed runtime used by applications and system services to execute the DEX bytecode specifications mentioned earlier. This means that ART and Dalvik are compatible runtimes that run DEX bytecode, and applications developed for Dalvik can easily work on ART, given some specific conditions are met. In both systems, the features used in our proposed model can be extracted without running the application on the Android device.

V. IMPLEMENTATION AND RESULTS

A. IMPLEMENTATION ENVIRONMENT

Tables 2 and 3 present the hardware and software specifications of the implementation computer, respectively. This computer was used for preprocessing, training, and testing machine learning classifiers. It was also used to host the virtual testing environment.

TABLE 2. Hardware for the implementation environment.

Item	Configuration
Processor	AMD Ryzen 5 3600
Clock speed	4.2 GHz
RAM	128 GB
GPU	NVidia GeForce RTX 3060Ti

TABLE 3. Software for the implementation environment.

Item	Version
Operating system	Windows 10 Pro
Python	3.8.5
Sci-Kit Learn	0.24.1
VMWare Workstation Pro	16.2.0

B. PERFORMANCE METRICS

The four basic performance metrics of a binary ML-based classifier are as follows:

1) True positive (TP): the number of test instances with true and predicted values of 1 divided by the number of test instances with a true value of 1.
2) True negative (TN): the number of test instances with true and predicted values of 0 divided by the number of test instances with a true value of 0.
3) False positive (FP): the number of test instances with a true value of 0 and a predicted value of 1 divided by the number of test instances with a true value of 0.
4) False negative (FN): the number of test instances with a true value of 1 and a predicted value of 0 divided by the number of test instances with a true value of 1.

When combined together, these four measures generate the confusion matrix.

We used the following six performance parameters in this work:

1) Accuracy: measures the ratio of correct predictions using the following equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

2) Precision: measures the ratio of the accuracy of positive predictions using the following equation:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

3) Recall: measures the ratio of positive instances correctly detected by the classifier using the following equation:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

4) F_1 score: measures the harmonic mean of precision and recall using the following equation:

$$F_1Score = 2 * \frac{\frac{TP}{TP+FN} * \frac{TP}{TP+FP}}{\frac{TP}{TP+FN} + \frac{TP}{TP+FP}} \tag{4}$$

5) Training time
The time required to train the classifier using the training subset.
6) Testing time
The time required for the trained classifier to process one input instance and produce a prediction.

C. TRAINING AND TESTING STRATEGY

To select the best-performing classifier, we chose five different types of classifiers to train and test. These classifiers were the random forest (RF), logistic regression (LR), decision tree (DT), Gaussian Naive Bayes (GNB), and support vector machine (SVM). Our goal was to create a lightweight solution. Accordingly, we chose not to use deep neural networks because of their high computational requirements compared to the five selected models [27].

The experiments were split into multiple phases as follows:

1) Initial training and testing: The dataset was randomly split into 75% training subset and 25% testing subset to obtain the initial results using all five classifiers.
2) Feature selection: The next step was to select a lower number of features from the original 215. The method used in the feature selection was recursive feature elimination (RFE) based on the feature importance. This method will be explained in Section V-E. After the feature selection process, the resulting reduced dataset was used to retrain the classifiers and produce a new set of results.
3) 10-fold cross-validation: We used 10-fold cross-validation on the reduced-feature data as an additional validation step. In the 10-fold cross-validation process, the data were randomly split into 10 subsets. The data then went through 10 cycles of training and testing.

TABLE 4. Initial results with 215 features.

Model	Accuracy	Precision	Recall	F_1 score	Training time (s)	Testing time (μs)
RF	0.9865	0.9866	0.9865	0.9865	0.7631	1.0109
LR	0.9790	0.9790	0.9790	0.9790	0.0710	0.3149
DT	0.9740	0.9741	0.9740	0.9740	0.1300	0.0409
GNB	0.7199	0.8245	0.7199	0.7195	0.0200	0.3641
SVM	0.9702	0.9705	0.9702	0.9700	2.2602	26.1088

In each cycle, one subset of the 10 was excluded from the training process and used for the testing process. This was repeated for 10 times until all 10 subsets had been used for testing one time. Each cycle produced a classifier with specific performance parameters. If these parameters have a high variance, then the classifier suffers from overfitting and is not properly generalizing within the dataset. If the variance is low, then the mean values of the performance parameters are reliable.

4) Additional datasets testing: For further validation, the trained classifiers are tested using two more datasets to ensure that the classifiers generalize well beyond their training dataset.

D. INITIAL TRAINING RESULTS

As mentioned in Section V-C, five classifiers were created for training and testing. These five classifiers were trained using 75% of the dataset, which was randomly selected, and tested using the remaining 25%. The dataset used in the initial training stage was the full dataset with 215 features. Table 4 presents the initial performance measurements obtained from the initial testing process.

In Table 4, the RF classifier yielded the highest accuracy of 0.9865, whereas the LR classifier exhibited the lowest testing time of 0.3149 $\mu s/instance$. This result implies the high accuracy needed as we progress in the creation of a lighter-weight model. Figure 3 illustrates the AUC of the five classifiers using the 215-feature dataset. Examining Figure 3 and Table 4, we found the GNB classifier producing the lowest accuracy of 0.7199.

E. FEATURE SELECTION AND ITS RESULTS

One of the major contributions of our research is to select a lower number of features without compromising accuracy of the malware detection system. As we explored statistical dimensionality reduction algorithms such as principal component analysis, singular value decomposition, and linear discriminant analysis, we found that these algorithms raise two concerns. The first is that the reduction in the number of features takes place in the features fed into the classifier only. It does not impact the data acquisition and hence does not have a noticeable impact on efficiency. The second concern is that all of these algorithms require resource intensive pre-processing to the acquired data to prepare it before being fed into the classifier in real-life deployments. Hence, we decided

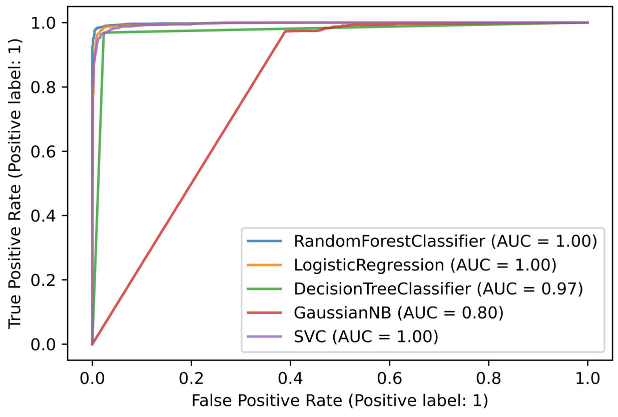

FIGURE 3. Area under the curve (AUC) for the classifiers with 215 features.

to utilize recursive feature elimination. Algorithm 1 presents the steps followed in RFE.

In each cycle of RFE, the dataset is randomly split into 75% training subset, and 25% testing subset. A random forest classifier is then created and trained using the training subset. Following this training, the classifier is tested using the testing subset. Afterwards, the feature importance is calculated for all features used in training the classifier. Feature importance can be measured as the averaged impurity decrease computed from all decision trees in the forest, without assuming that the data is linearly separable or not [28]. The feature with the lowest importance value is then eliminated. Then,

Algorithm 1 Recursive Feature Elimination Based on Feature Importance

```
1  Input: Dataset with 215 features
2  Output: Dataset with 35 features
3  $Array \leftarrow Dataset$
4  $model \leftarrow RandomForestClassifier$
5  $TargetFeatures \leftarrow 35$
6  while $Features(Dataset) > TargetFeatures$ do
7      $RandomSplit(Array) \rightarrow Train\_Array, Test\_Array$
8      train $model$ with $Train\_Array$
9      $importance \leftarrow FeatureImportance(model)$
10     $i \leftarrow$ index of feature with lowest importance
11     $Array.DeleteFeature(i)$
12 end
13 $Dataset \leftarrow Array$
```

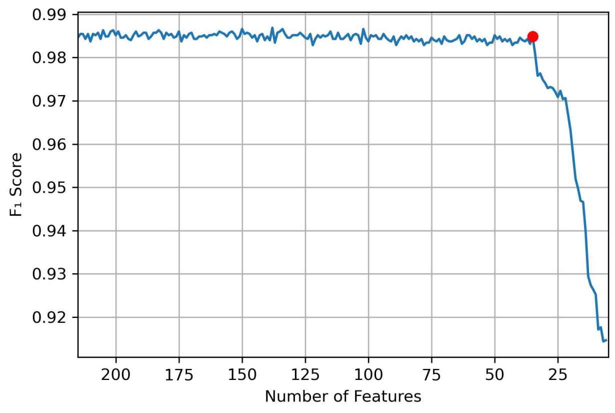

FIGURE 4. Impact of the feature reduction on the generated F_1 score.

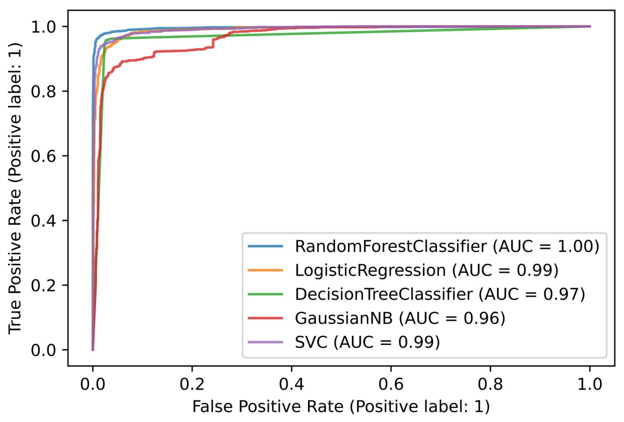

FIGURE 5. Area under the curve (AUC) for the classifiers with 35 features.

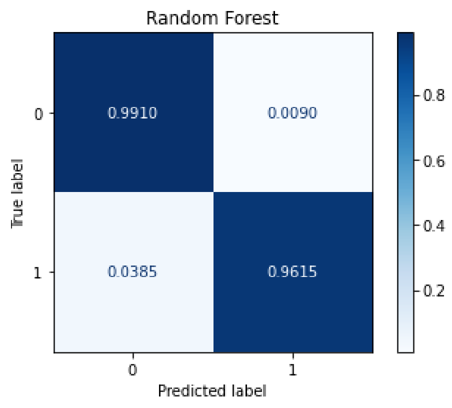

FIGURE 6. Confusion matrix of RF classifier with 35 features.

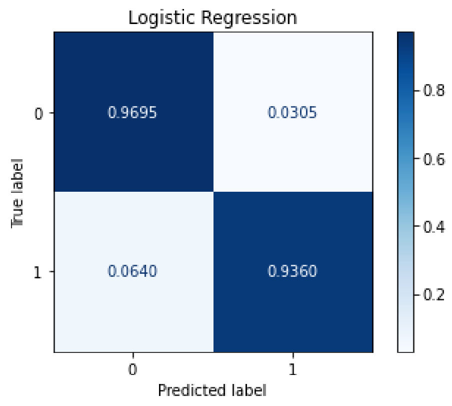

FIGURE 7. Confusion matrix of LR classifier with 35 features.

the cycle is repeated while closely monitoring the system's performance by recording the F_1 score of the tested classifier. The process is repeated until a certain threshold is reached. This threshold is chosen upon the system witnessing rapid drop in performance. As shown in Figure 4, the threshold chosen in our experiment was 35, as we noticed a rapid drop in the classifier's F_1 score beyond the 35-feature threshold.

Based on this reduction method, the number of features that need to be used in the prediction in live deployments was reduced, not only the dimensionality of the data input to the system. This enabled more efficient data acquisition, training, testing, and lightweight real-life deployment.

As mentioned in Section II, several other papers relied on feature importance to select features with the highest importance. Our research did not follow the same method of selecting features with the highest importance. It instead relied on the repetitive elimination of the feature with the lowest importance and model re-training. This successive elimination assured that any correlation between the features will not affect the independent feature importance. In other words, the importance of one feature might be affected by the existence (and the elimination of) another feature. Hence, we retrained and re-calculated the importance after each feature was eliminated.

We used the new dataset with 35 features to train and test the classifier models. Table 5 shows the accuracy, precision, recall, F_1 score, and training and testing times for the five classifiers. Figure 5 illustrates the AUC for the five classifiers with the reduced dataset.

Examining Table 5, the accuracy dropped less than 0.006 in the RF classifier, 0.02 in the LR classifier, and 0.01 in the DT and SVM classifiers. In contrast, the GNB classifier accuracy improved by 0.17. The timing parameters improved in all models with the reduction of the features from 215 to 35. Appendix A presents comparisons of the performance measures for all classifiers using the 215- and 35-feature datasets. Figures 6, 7, 8, 9, and 10 illustrate the confusion matrix for the 35-feature RF, LR, DT, GNB, and SVM classifiers, respectively.

The results show that the RF classifier outperformed the other classifiers in terms of accuracy by approximately 2%. In addition, RF yielded the lowest FP and FN rates with values of 0.0385 and 0.0090 respectively. In terms of performance, LR achieved the lowest training time of 0.0028 μs per instance. The difference in the accuracy in high-sensitivity applications like malicious application detection is not something to be overlooked. Hence, we chose the RF classifier as

TABLE 5. Results after the feature reduction process.

Model	Accuracy	Precision	Recall	F_1 score	Training time (s)	Testing time (μs)
RF	0.9807	0.9807	0.9807	0.9806	0.4170	0.7631
LR	0.9571	0.9570	0.9571	0.9570	0.0306	0.0028
DT	0.9667	0.9667	0.9667	0.9667	0.0207	0.0030
GNB	0.8982	0.9011	0.8982	0.8989	0.0003	2.0276
SVM	0.9661	0.9664	0.9661	0.9660	0.7380	15.8450

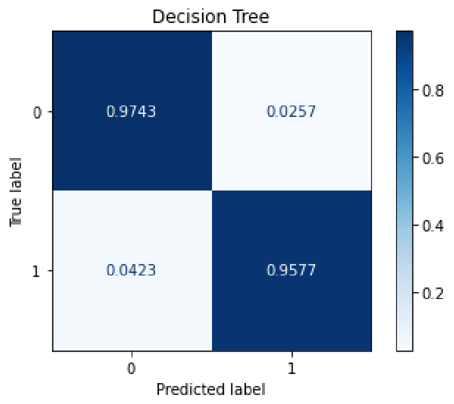

FIGURE 8. Confusion matrix of DT classifier with 35 features.

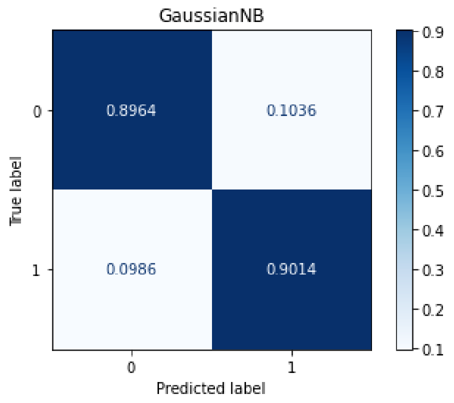

FIGURE 9. Confusion matrix of GNB classifier with 35 features.

the final proposed model to conduct further testing and verify the generalization capabilities of the model.

F. 10-FOLD CROSS-VALIDATION RESULTS

As explained in Section V-C, we conducted a 10-fold cross-validation to assure that the trained model can properly generalize. Table 6 presents the cross-validation results for the chosen RF classifier.

In Table 6, the classifier maintained a high accuracy during all 10 folds, yielding a mean accuracy of 0.9820. The standard deviation in the accuracy measure was minimal with 0.002511. This result proved that the model generalized well and did not suffer from overfitting. Furthermore, the other performance measures maintained high resilience all throughout the 10 folds and caused only a minimal standard deviation.

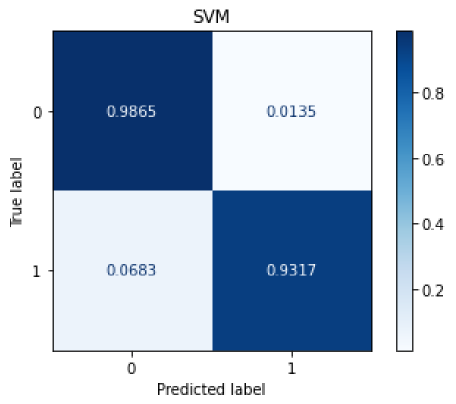

FIGURE 10. Confusion matrix of SVM classifier with 35 features.

TABLE 6. 10-Fold cross-validation results for the random forest classifier.

Fold	Accuracy	Precision	Recall	F_1 Score
1	0.986037	0.987410	0.975133	0.981233
2	0.981383	0.986989	0.961957	0.974312
3	0.984707	0.985481	0.973118	0.979261
4	0.983378	0.989286	0.966841	0.977935
5	0.982048	0.985321	0.965827	0.975477
6	0.984707	0.988014	0.973019	0.980459
7	0.980705	0.988212	0.956274	0.971981
8	0.978044	0.977273	0.960894	0.969014
9	0.980705	0.985375	0.962500	0.973803
10	0.978709	0.981132	0.959410	0.970149
Std dev	0.002511	0.003474	0.006118	0.004059
Mean	0.982042	0.985449	0.965497	0.975362

G. TESTING WITH MALGENOME-215 AND CICMalDroid2020

We used two additional testing dataset to further ensure the generalization capability of the model. The first testing dataset, which is the Malgenome-215 dataset [25], also has 215 features. It comprises 3799 instances, in which 2539 were benign, and 1260 were malicious. We extracted a subset of the Malgenome-215 dataset with only the 35 features chosen for our proposed model, and then ran these instances through our model for testing. Table 7 presents the testing results. Figure 11 depicts the confusion matrix plot for the test using the Malgenome-215 dataset.

The results obtained by testing with the 35-feature version of the Malgenome-215 dataset show that the classifier maintained a very high accuracy of 0.9873 while maintaining a

TABLE 7. Performance measure for the proposed system using 35 features from the Malgenome-215 and CICMalDroid2020 datasets.

Performance measure	Malgenome-215	CICMalDroid2020
Accuracy	0.9873	0.9798
Precision	0.9847	0.9810
Recall	0.9876	0.9759
F_1 score	0.9860	0.9783
Test time	0.7661 μs	0.8206 μs

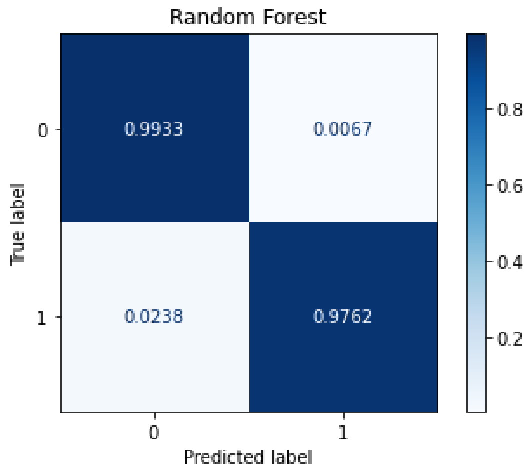

FIGURE 11. Confusion matrix for testing the RF classifier with the 35-feature version of Malgenome-215.

low processing time of 0.7661μs. Furthermore, the classifier achieved a very low FN rate of 0.6% and a FP rate of 2.38%. These values are a good indication that the classifier generalizes well to data not seen before.

The second testing dataset was CICMalDroid2020 dataset [26]. We extracted the 35 features that were selected earlier to feed them into the trained classifier to tests it. The 11,598 instances were sent through our model for testing. Table 7 presents the testing results. Figure 12 depicts the confusion matrix plot for the test using the CICMalDroid2020 dataset.

The results obtained by testing with the 35-feature version of the CICMalDroid2020 dataset show that the classifier maintained a very high accuracy of 0.9798 while maintaining a low processing time of 0.8206μs. Furthermore, the classifier achieved a very low FN rate of 0.87% and a FP rate of 3.96%. These values are a good indication that the classifier generalizes well to data not seen before.

VI. EXPLAINABILITY OF THE MODEL

One of the goals of this research is to produce an explainable model, which will increase trust in the proposed solution. Explainability not only builds trust in the model, but also ensures that the achieved accuracy originates from explainable conditions, and not from a black-box operation.

In explaining our proposed model, we will rely on Shapley Additive Explanation (SHAP). In 2017, [29] introduced SHAP as a model-agnostic method for explaining machine

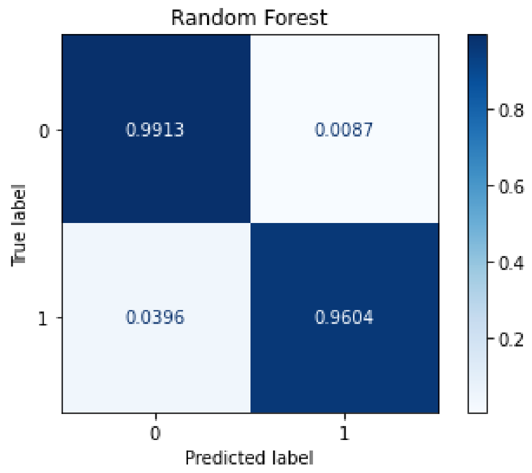

FIGURE 12. Confusion matrix for testing the RF classifier with the 35-feature version of CICMalDroid2020.

learning models. SHAP is based on the Shapley values taken from the game theory [30]. Shapley values are calculated by measuring the impact of each individual player in a team game by computing the team performance with and without that player. In machine learning, this method calculates the impact of each feature by calculating the difference between the model performance with and without the feature. This helps us understand how much each feature contributes to the prediction in a positive or negative way.

As discussed in [31], SHAP values are considered as a better explanation method compared to feature importance. Feature importance is the technique of calculating a score for all the input features for a given model. This score represents the importance of each feature. Feature importance is calculated through Gini importance using node impurity and can only be calculated in linear machine learning algorithms, such as linear and LR, RF, and DT. In contrast, SHAP values are model-agnostic and can be used to explain any type of classifier, including deep learning. The way the SHAP values are calculated can provide better insights into the impact of each individual feature on the classification decision.

Figure 13 shows a summary plot of the SHAP values of the top 10 features within the dataset. These 10 features were arranged in a descending order from the feature with the highest impact on the decision to that with the lowest impact.

To obtain a better understanding of the SHAP explanation, we must re-iterate that the prediction of the proposed model is 1 when the prediction is "malware" and 0 when "benign". In Figure 13, the values on the left side of the vertical axis drag the prediction value down, making the prediction closer to "benign". Meanwhile, the values on the right side of the vertical axis push the prediction value up, making the prediction closer to "malware". The red dots represent a high feature value, while the blue dots represent a low feature value. Most of the features used in our model were binary features; hence, the red dots represent 1, while the blue dots represent 0.

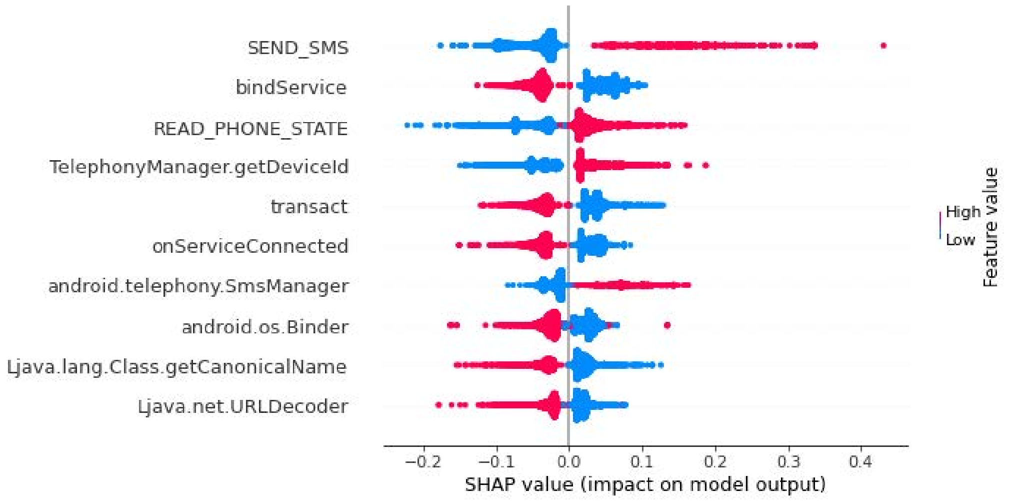

FIGURE 13. Summary plot of the SHAP values for the 10 highest impact features.

TABLE 8. Comparison of the performance measures of PAIRED with those presented in previously published studies.

Reference	Dataset	Features	Classifier	F_1 score	FP	FN	Testing time
[10]	Malgenome, Drebin, McAfee	100–350	–	0.9840	0.0250	0.0175	–
[15]	Drebin and AMD	93,324	MLP	0.9973	–	–	$\leq$ 5s
[21]	AMD	27	MLP	0.961	0.03	0.04	–
[32]	Drebin	93,324	RF	0.9310	0.077	–	–
Proposed PAIRED	Drebin-215	35	RF	0.9860	0.0238	0.0067	0.7631(μs)

Figure 13 depicts that the "SEND_SMS" feature holds the highest impact on the prediction. With a value of 1, it pushes the prediction closer to malicious. With a value of 0, it brings the prediction closer to benign. This comes from the fact that many malware instances would send text messages either to share the infection with other devices or to subscribe to premium service numbers that would cause significant financial losses to the user. In a similar manner, the value of 1 in the "READ_PHONE_STATE," "TelephonyManager.getDeviceId," and "android.Telephony.SmsManager" features would push the prediction closer to malware. READ_PHONE_STATE is an application permission that, when granted, allows reading access to the phone state, including the current cellular network information, status of any ongoing calls, and list of any calling accounts registered on the device. This information can be used to capture coarse location through cellular network information, which is a serious privacy violation. Applications can use the TelephonyManager.getDeviceId action to retrieve the International Mobile Equipment Identity, which is the unique identifier of a phone's SIM card. This number can be used in spoofing attacks to take over the phone number and in mobile banking fraud or other types of fraud. Meanwhile, android.Telephony.SmsManager is an object that manages SMS operations, such as sending data, text, and pdu SMS messages. This feature connects to the SEND_SMS feature, but gives a different access to the application in reading and sending SMS messages. An individual examination of these three features might not conclusively indicate the application to be "malware". However, when combined together, and with other features (e.g., SEND_SMS), their combination becomes very suspicious.

The second feature in terms of impact is "bindService". In an Android environment, bindService would allow a client to be bound to an existing service when called. This feature's value of 1 pushes the prediction closer to benign because this service does not allow a client to bind to a service without authorization. Hence, in most cases, it is a technique that is useless to malware creators. The "Ljava.lang.Class.getCanonicalName" and "Ljava.net.URLDecoder" features have the similar effect of pushing the prediction closer to benign when having a value of 1. This comes from the fact that these two services are rarely used by malware. Both features are relevant to resolving domain names to IP addresses. As with most malware, when communicating with a command and control (C&C) center, they directly communicate using IP addresses without needing to resolve a domain name. By contrast, most benign applications rely on domain names, such that they need not update the application when their server IP addresses change. Appendix C presents the SHAP summary plot for the 35 features.

VII. DISCUSSION

The results provided in Section V show very good performance measures with 0.9807 accuracy, 0.9806 F_1 score,

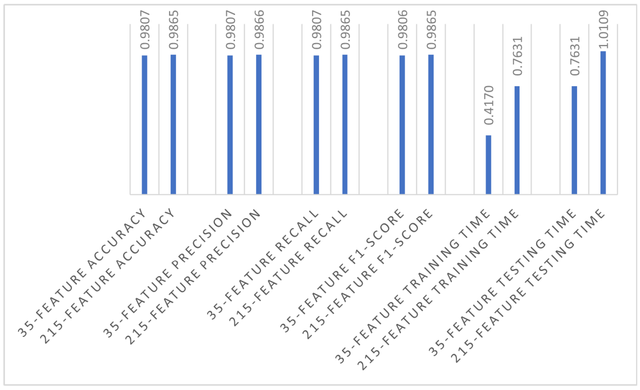

FIGURE 14. Impact of feature reduction on the RF classifier.

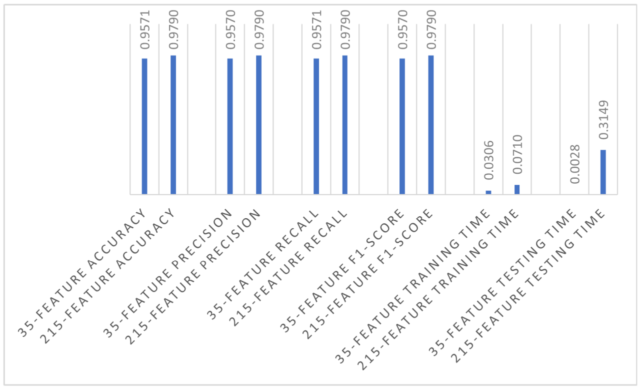

FIGURE 15. Impact of feature reduction on the LR classifier.

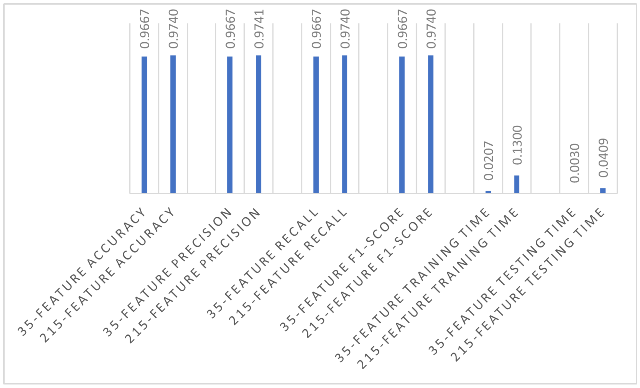

FIGURE 16. Impact of feature reduction on the DT classifier.

0.7631 μs test time, 1.00 AUC, 0.0238 FP, and 0.0067 FN. Having such a high accuracy with a very low FN value makes the classifier model a powerful tool for malware detection. Table 8 presents a comparison of the performance of the proposed system with other state-of-the-art approaches.

Table 8 shows that the proposed system outperformed most of the state-of-the-art approaches in terms of the performance measures despite the fact that it was designed to be lightweight. Although [15] presented a higher F_1 score, it was based on MLP with 93,324 features, making it heavily

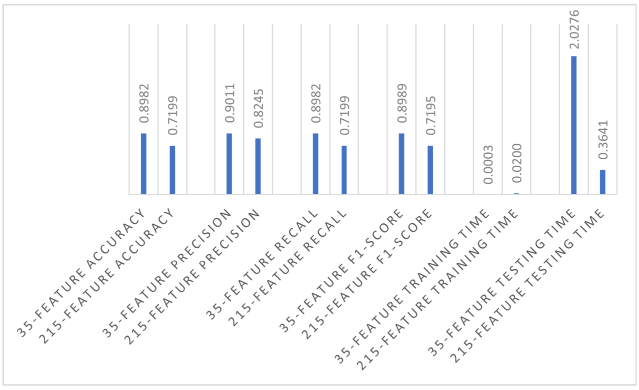

FIGURE 17. Impact of feature reduction on the GNB.

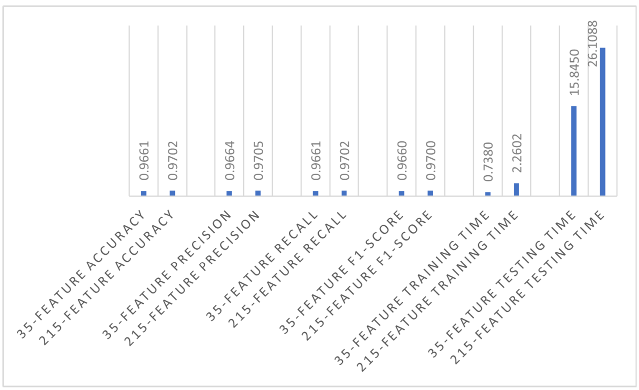

FIGURE 18. Impact of feature reduction on the SVM.

resource-intensive and requiring a testing time of approximately 5 s. By contrast, PAIRED requires only 0.7μs.

When compared to [21], our proposed system does not only provide a higher F_1 score, but also presents itself to be of lighter weight. Although [21] utilizes 27 permission-dependant features only, it uses MLP as its classifier type. It is well-known that neural-network based classifiers are considered much more resource intensive in comparison to RF classifiers [27]. This makes our proposed system more suitable to Android devices of lower processing power.

The proposed system was successful in producing a smaller version of the Drebin-215 dataset with only 35 features while maintaining a high accuracy. This sets the stage for further research using the reduced dataset to produce more agile and efficient malware detection systems. Appendix B provides the list of selected features.

The proposed system meets all of its design principles in delivering high accuracy and efficiency, being lightweight, and being a generalizable classifier. High accuracy was achieved with a 0.9807 accuracy value in testing. The lightweight state was achieved by using 35 features instead of the full 215 features. The 84% reduction in the classifier input translates to a lower memory requirement of approximately 169 MB and lower processing requirements. The memory and processing requirements in neural networks were much higher than those for the RF; hence, using the

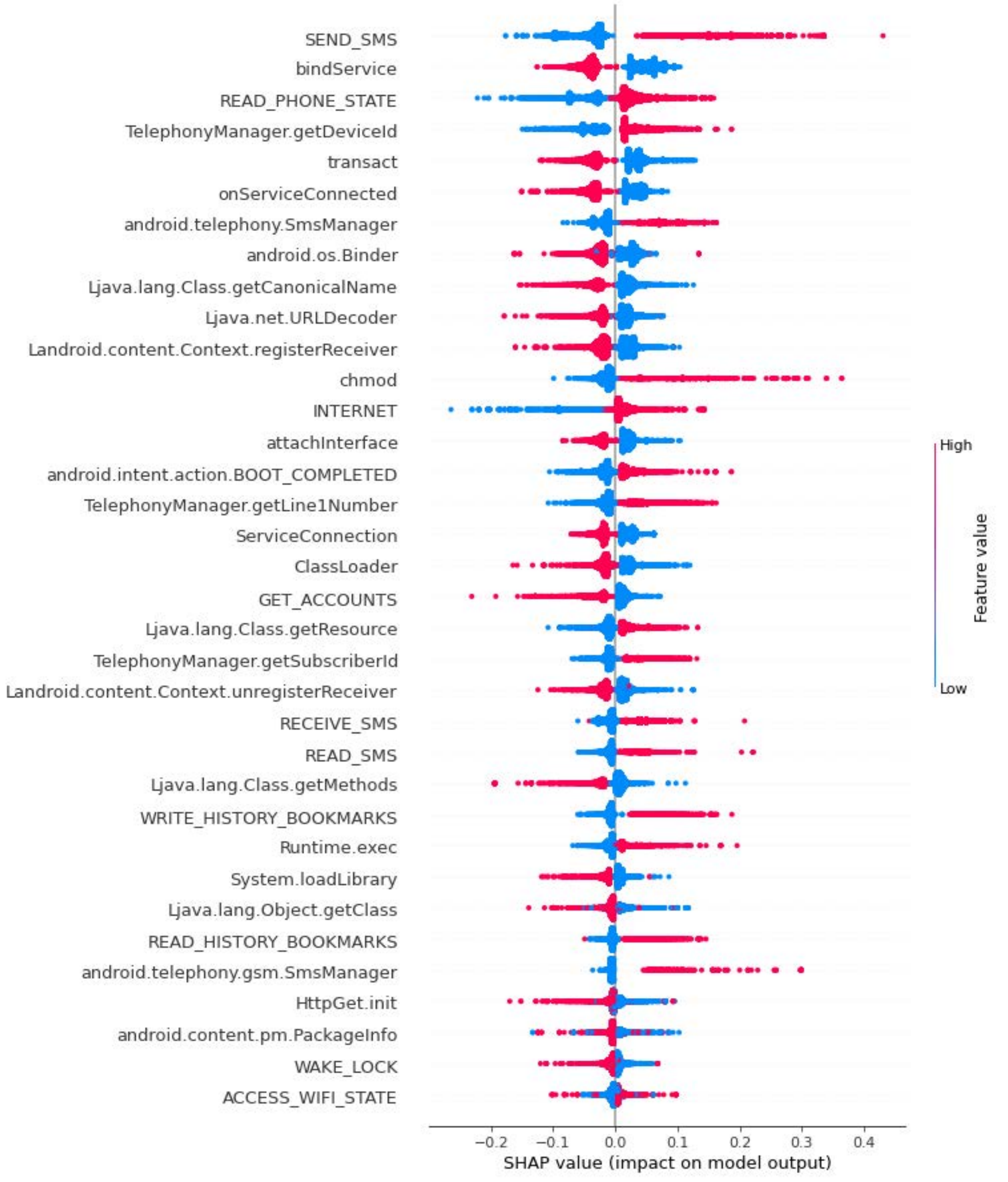

FIGURE 19. SHAP summary plot for the 35 features.

RF classifier instead of neural networks also contributed to the creation of a lightweight system. High efficiency was apparent in the classifier's testing time (i.e., $0.7631 \mu s$). The efficiency improvement occurred in the feature extraction phase, where only 35 instead of 215 features were extracted. Testing showed that the model generalizes well beyond the training dataset. We demonstrated this by splitting the dataset into 75% training subset and 25% testing subset to assure that testing is done with previously unseen instances. Another testing step was the 10-fold cross-validation. The proposed model performed very well in the cross-validation, yielding a mean accuracy of 0.9820 and a standard deviation of 0.002511. Further testing was done to assure generalization through testing with two different dataset, called the Malgenome-215, and CICMalDroid2020 datasets. The Malgenome-215, and CICMalDroid2020 datasets produced accuracy of 0.9873, and 0.9798 respectively. These high performance metrics prove that the produced classifier generalizes well beyond the training dataset.

The size of the trained classifier model was very small (approximately 8 MB); thus, the model could be retrained when new malware data become available. The updated trained model can easily be downloaded as a system update. This provides high flexibility in addressing the future types of malware.

VIII. CONCLUSION AND FUTURE WORK

In this study, we presented an explainable lightweight high accuracy Android malware detection system, called PAIRED. PAIRED extracted 35 static features from applications and

TABLE 9. Feature importance for the 35 selected features.

Feature Number	Feature Name
1	transact
2	onServiceConnected
3	bindService
4	attachInterface
5	ServiceConnection
6	android.os.Binder
7	SEND_SMS
8	Ljava.lang.Class.getCanonicalName
9	Ljava.lang.Class.getMethods
10	Ljava.net.URLDecoder
11	android.telephony.SmsManager
12	READ_PHONE_STATE
13	ClassLoader
14	Landroid.content.Context.registerReceiver
15	Landroid.content.Context.unregisterReceiver
16	GET_ACCOUNTS
17	RECEIVE_SMS
18	READ_SMS
19	android.intent.action.BOOT_COMPLETED
20	android.content.pm.PackageInfo
21	TelephonyManager.getLine1Number
22	HttpGet.init
23	System.loadLibrary
24	android.telephony.gsm.SmsManager
25	WRITE_HISTORY_BOOKMARKS
26	TelephonyManager.getSubscriberId
27	Ljava.lang.Object.getClass
28	READ_HISTORY_BOOKMARKS
28	INTERNET
30	TelephonyManager.getDeviceId
31	WAKE_LOCK
e 32	chmod
33	Runtime.exec
34	Ljava.lang.Class.getResource
35	ACCESS_WIFI_STATE

passed them to a trained machine learning classifier to produce a prediction of whether an application is malicious or benign. The testing results showed that the proposed PAIRED system performs very well compared to the state-of-the-art approaches. PAIRED achieved a high accuracy of 0.9802 with a very low FN rate of 0.0090. The novelty of our contribution is summarized by the creation of a lightweight system with high accuracy, the production of a reduced version of the Drebin-215 dataset, and the improvement of the efficiency and generalization of the proposed system. The model was also explained using SHAP values to increase trust and understand the internal operations of the proposed classification model.

Our future research directions lean toward the creation of a robust cloud-based machine learning model update function. Another direction would be working further on the reduction of the memory and processing requirements of the classifier. Finally, further work can be done to improve accuracy by using additional datasets and classifiers.

APPENDIX A
COMPARISONS OF THE PERFORMANCE MEASURES FOR THE CLASSIFIERS USING 215 AND 35 FEATURES

Figures 14, 15, 16, 17, and 18 show the impact of feature reduction on the performances of the RF, LR, DT, GNB, and SVM classifiers, respectively.

APPENDIX B
SELECTED 35 FEATURES

See Table 9.

APPENDIX C
SHAP SUMMARY PLOT FOR THE 35 FEATURES

See Figure 19.

REFERENCES

[1] (Dec. 2021). *Internet Users Worldwide by OS 2012-2021 | Statista*. Accessed: Dec. 10, 2021. [Online]. Available: https://www.statista.com/statistics/543185/worldwide-internet-connected-operating-system-population

[2] (Dec. 2021). *Google Play Store: Number of Apps 2021 | Statista*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store

[3] P. Kotzias, J. Caballero, and L. Bilge, "How did that get in my phone? Unwanted app distribution on Android devices," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 53–69.

[4] C. Pernet. (Dec. 2021). *Android Malware Infected More Than 300,000 Devices With Banking Trojans*. TechRepublic. [Online]. Available: https://www.techrepublic.com/article/android-malware-infected-more-than-300000-devices-with-banking-trojans

[5] M. M. Alani, "Android users privacy awareness survey," *Int. J. Interact. Mobile Technol.*, vol. 11, no. 3, p. 130, Apr. 2017.

[6] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.

[7] A. Arora, S. K. Peddoju, and M. Conti, "PermPair: Android malware detection using permission pairs," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1968–1982, 2020.

[8] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features," *Future Gener. Comput. Syst.*, vol. 105, pp. 230–247, Apr. 2020.

[9] Q. Han, V. S. Subrahmanian, and Y. Xiong, "Android malware detection via (somewhat) robust irreversible feature transformations," *IEEE Trans. Inf. Forensics Securiity*, vol. 15, pp. 3511–3525, 2020.

[10] S. Y. Yerima and S. Sezer, "DroidFusion: A novel multilevel classifier fusion approach for Android malware detection," *IEEE Trans. Cybern.*, vol. 49, no. 2, pp. 453–466, Feb. 2018.

[11] A. Firdaus, N. B. Anuar, A. Karim, and M. F. A. Razak, "Discovering optimal features using static analysis and a genetic search based method for Android malware detection," *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 6, pp. 712–736, Jun. 2018.

[12] H. Cai, N. Meng, B. G. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1455–1470, 2018.

[13] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimedia Tools Appl.*, vol. 78, no. 4, pp. 3979–3999, 2019.

[14] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, "EveDroid: Event-aware Android malware detection against model degrading for IoT devices," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6668–6680, Aug. 2019.

[15] C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang, and H. Kinawi, "Android malware detection based on factorization machine," *IEEE Access*, vol. 7, pp. 184008–184019, 2019.

[16] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. London, U.K.: Springer, 2017, pp. 252–276.

[17] S. Millar, N. McLaughlin, J. M. del Rincon, P. Miller, and Z. Zhao, "DANdroid: A multi-view discriminative adversarial network for obfuscated Android malware detection," in *Proc. 10th ACM Conf. Data Appl. Secur. Privacy*, New York, NY, USA, 2020, pp. 353–364, doi: 10.1145/3374664.3375746.

[18] V. Kouliaridis, G. Kambourakis, D. Geneiatakis, and N. Potha, "Two anatomists are better than one—Dual-level Android malware detection," *Symmetry*, vol. 12, no. 7, p. 1128, Jul. 2020.

[19] Y. Hei, R. Yang, H. Peng, L. Wang, X. Xu, J. Liu, H. Liu, J. Xu, and L. Sun, "Hawk: Rapid Android malware detection through heterogeneous graph attention networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 27, 2021, doi: 10.1109/TNNLS.2021.3105617.

[20] T. Frenklach, D. Cohen, A. Shabtai, and R. Puzis, "Android malware detection via an app similarity graph," *Comput. Secur.*, vol. 109, Oct. 2021, Art. no. 102386.

[21] D. Ö. Şahin, O. E. Kural, S. Akleylek, and E. Kılıç, "A novel permission-based Android malware detection system using feature selection based on linear regression," *Neural Comput. Appl.*, vol. 2021, pp. 1–16, Mar. 2021.

[22] D. Ö. Şahin, O. E. Kural, S. Akleylek, and E. Kılıç, "A novel Android malware detection system: Adaption of filter-based feature selection methods," *J. Ambient Intell. Hum. Comput.*, vol. 2021, pp. 1–15, Jul. 2021.

[23] A. Mahindru and A. L. Sangal, "FSDroid: A feature selection technique to detect malware from Android using machine learning techniques," *Multimedia Tools Appl.*, vol. 80, no. 9, pp. 13271–13323, Apr. 2021.

[24] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.

[25] S. Yerima. (Feb. 2018). *Android Malware Dataset for Machine Learning 1*. [Online]. Available: https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_1/5854590/1

[26] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android malware category classification using semi-supervised deep learning," in *Proc. IEEE Int. Conf. Dependable, Autonomic Secure Comput., Int. Conf. Pervasive Intell. Comput., Int. Conf. Cloud Big Data Comput., Int. Conf. Cyber Sci. Technol. Congr. (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2020, pp. 515–522.

[27] A. Géron, *Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[28] S. Raschka, Y. Liu, and V. Mirjalili, *Machine Learning With PyTorch and Scikit-Learn*. Packt, 2022.

[29] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–10.

[30] E. Winter, "The Shapley value," in *Handbook of Game Theory With Economic Applications*, vol. 3. Amsterdam, The Netherlands: Elsevier, 2002, pp. 2025–2054.

[31] W. E. Marcilio and D. M. Eler, "From explanations to feature selection: Assessing SHAP values as feature selection mechanism," in *Proc. 33rd SIBGRAPI Conf. Graph., Patterns Images (SIBGRAPI)*, Nov. 2020, pp. 340–347.

[32] T. Chen, Q. Mao, Y. Yang, M. Lv, and J. Zhu, "TinyDroid: A lightweight and efficient model for Android malware detection and classification," *Mobile Inf. Syst.*, vol. 2018, pp. 1–9, Oct. 2018.

MOHAMMED M. ALANI (Senior Member, IEEE) received the Ph.D. degree in computer engineering with a specialization in networks security. He has worked as a Professor and a cybersecurity expert in many countries around the world. His experiences include serving as the VP of Academic Affairs in the United Arab Emirates; networks and security consultancies in the Middle East; and the Cybersecurity Program Manager in Toronto, Canada. He currently works as a Cybersecurity Professor with the Seneca College and a Research Fellow with Toronto Metropolitan University (previously Ryerson University), Toronto. He has authored four books in different areas of networking and cybersecurity along with many research papers published in highly ranked journals and conferences. His current interests include applications of ML in cybersecurity and ML security.

ALI ISMAIL AWAD (Senior Member, IEEE) is currently an Associate Professor with the College of Information Technology (CIT), United Arab Emirates University (UAEU), Al Ain, United Arab Emirates. He is also an Associate Professor (Docent) with the Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden, where he also served as a Coordinator of the Master's Program in Information Security from 2017 to 2020; an Associate Professor with the Electrical Engineering Department, Faculty of Engineering, Al-Azhar University, Qena, Egypt; and a Visiting Researcher with the University of Plymouth, U.K. His research interests include cybersecurity, networks security, the Internet of Things security, and image analysis with biometrics and medical imaging applications. He has edited or co-edited eight books and has authored or coauthored several journal articles and conference papers in these areas. He is an Editorial Board Member of the *Future Generation Computer Systems* journal, *Computers and Security* journal, the *Internet of Things: Engineering Cyber Physical Human Systems* journal, and *Health Information Science and Systems* journal.