Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.053163

ARTICLE

# An Attention-Based Approach to Enhance the Detection and Classification of Android Malware

**Abdallah Ghourabi***

Department of Computer Science, College of Computer and Information Sciences, Jouf University, Sakaka, 72388, Saudi Arabia

*Corresponding Author: Abdallah Ghourabi. Email: aghourabi@ju.edu.sa

Received: 26 April 2024 Accepted: 07 June 2024 Published: 15 August 2024

**ABSTRACT**

The dominance of Android in the global mobile market and the open development characteristics of this platform have resulted in a significant increase in malware. These malicious applications have become a serious concern to the security of Android systems. To address this problem, researchers have proposed several machine-learning models to detect and classify Android malware based on analyzing features extracted from Android samples. However, most existing studies have focused on the classification task and overlooked the feature selection process, which is crucial to reduce the training time and maintain or improve the classification results. The current paper proposes a new Android malware detection and classification approach that identifies the most important features to improve classification performance and reduce training time. The proposed approach consists of two main steps. First, a feature selection method based on the Attention mechanism is used to select the most important features. Then, an optimized Light Gradient Boosting Machine (LightGBM) classifier is applied to classify the Android samples and identify the malware. The feature selection method proposed in this paper is to integrate an Attention layer into a multilayer perceptron neural network. The role of the Attention layer is to compute the weighted values of each feature based on its importance for the classification process. Experimental evaluation of the approach has shown that combining the Attention-based technique with an optimized classification algorithm for Android malware detection has improved the accuracy from 98.64% to 98.71% while reducing the training time from 80 to 28 s.

**KEYWORDS**

Android malware; malware detection; feature selection; attention mechanism; LightGBM; mobile security

## 1 Introduction

The mobile application market has grown rapidly in recent years. According to the newly released State of Mobile 2022 report by App Annie [1], 170 billion dollars were spent in application stores in 2021, 230 billion new applications were downloaded and an average of 4.8 h per day were spent on a mobile device. The major part of this application market is reserved for the Android environment. As of August 2022, an estimated 71.52% of mobile phone users are using Android (according to StatCounter [2]). This great popularity of Android applications has made them targets of several types of malwares. In fact, the personal data stored on Android devices including account credentials and banking transactions attract the attention of hackers on these devices and encourage them to perform malicious behavior through Android applications.

A recent report on the evolution of mobile malware [3] indicated that, in 2021, a large number of malware applications was detected by Kaspersky Labs, including 97,661 new mobile banking Trojans, 3,464,756 mobile malicious installer packages, 17,372 new mobile ransomwares. The same survey also illustrated how sophisticated mobile attacks are growing. Even with Google's efforts to keep threats away from its application platform, the experts still manage to find malware on Google Play. With regard to mobile malware detection systems, it is still challenging to create effective detection mechanisms comparable to those of personal computers due to several constraints associated with the characteristics of mobile devices.

The research community has proposed several solutions for Android malware detection based on machine learning. These solutions usually involve two main steps. The first step is the extraction and selection of features from the applications. The second step is the creation of a classifier to distinguish malicious from benign applications. Selecting appropriate techniques for both steps is crucial to achieving efficient classification results, especially if one wants to deploy these systems on mobile devices with limited hardware performance. For instance, the feature collection phase requires extracting diverse features, which may comprise requested permissions, API calls, opcodes, activities, and system features. This large number of features significantly increases the size of the dataset and can affect the classifier's execution time and accuracy. The choice of the classifier is also vital in this context. For example, classifiers based on deep learning are very efficient in the classification results; however, they are known for their high memory occupation and slow execution. In the current paper, we propose an approach for Android malware detection through feature selection and classification techniques that allow both to keep a high level of classification precision and to reduce the execution time of the model. Several recent studies [4–6] have shown that data optimization using feature selection techniques can improve the performance of attack and anomaly detection systems, especially in environments with limited hardware characteristics, such as mobile and IoT devices.

The set of features extracted from Android applications for malware analysis is large and diverse. This can result in a large amount of data that also contains irrelevant and redundant elements, making the task of classification extremely complex. For example, in the dataset CCCS-CIC-AndMal-2020, which we used for the experimental evaluation of our approach, each instance contains 5911 features. Analyzing such high-dimensionality dataset is challenging as it requires longer processing times and more storage and can potentially lead to misclassification of Android applications. This work proposes an innovative feature selection method for dimensionality reduction. In the literature, several research studies have proposed methods to select features and reduce their dimensionality in Android malware classification systems. The majority of works are based on statistical methods such as chi-square [7,8], PCA [8,9], information gain [10,11] and Fast Correlation-Based Filter [12]. Other studies have considered the use of techniques based on neural networks such as Restricted Boltzmann Machines [13] and RNN [14]. In other works, researchers have tried techniques based on graphs and trees such as Dominance Tree [15] and Random Forest [16]. The present paper complements and extends previous research efforts by exploring the effectiveness of the Attention mechanism as a feature selection technique for mobile malware detection and classification. The Attention mechanism has been very successful in the ML community in the last few years, especially in NLP tasks. Based on neural network architecture, this mechanism utilizes a weighted vector to help understand the relationship between the input features and the target and to estimate which part of the data is more pivotal than the others to accomplish the ML task. Although this mechanism is successful in natural language processing tasks, it is not yet well explored for mobile malware detection.

In this paper, we propose an Android malware detection system based on the Attention mechanism and the LightGBM classifier. The goal of our solution is to optimize the performance of the Android malware detection systems by reducing the number of features while maintaining optimal classification accuracy and ensuring a fast and efficient classification process.

The main contributions of our work are summarized below:

- The proposal of a neural network based on the Attention mechanism dedicated to reducing the number of features and selecting only those crucial to the classification results.
- The application of a classification model created using a LightGBM algorithm optimized through the use of the Bayesian method.
- The proposal of an optimized detection system that can be integrated into mobile devices while ensuring fast execution and high accuracy.
- The evaluation of the proposed solution demonstrated a reduction in processing time and a slight improvement in classification accuracy from 98.64% to 98.71%.

The rest of the paper is organized as follows: Section 2 reviews the related works. Section 3 describes our approach and its model design. Section 4 presents the results of the experimental tests. Finally, we conclude the paper in Section 5.

## 2 Related Work

In this section, we provide an overview of recent work on feature dimensionality reduction in the field of Android malware detection and classification. These works are classified into three categories depending on the type of methods used for feature selection: statistical based methods, neural network based methods, and tree based methods.

### *2.1 Statistical Based Methods*

Most papers in the literature favor the use of statistical techniques (such as chi-square test, Fisher's score, information gain, etc.) to select or reduce features due to their speed and low computational cost. For example, Cai et al. [10] proposed JOWMDroid, a feature-based approach to detect Android malware combining weight mapping optimization with classifier parameter optimization. Their idea is to utilize information gain to select the most relevant features from eight categories of features extracted from APK files. Next, three machine learning algorithms were used to calculate the initial weight for each feature and then map them to the final weights. The final step is to utilize the differential evolution algorithm to jointly optimize the parameters of the weighting function and the classifier.

Another work dealing with feature reduction for Android malware detection was proposed by Xie et al. [11]. In their approach, they employed a two-step feature selection method consisting of using InfoGain for an initial selection and applying the chi-square test for further reduction to remove redundant and irrelevant features. For the classification of Android malware, they used a stacking method of 5 base classifiers with the use of the Genetic Algorithm to optimize the hyperparameters of the model.

In [12], the authors introduced a malware detection framework called FAMD (Fast Android Malware Detector). It utilizes the FCBF (Fast Correlation-Based Filter) algorithm and the N-Gram technique to process the features and reduce their dimensionality. Then, the CatBoost classifier is utilized for malware classification. Experimental tests demonstrated a malware detection accuracy of 97.40% and a malware family classification accuracy of 97.38%.

In [7], the authors investigated the utility of the feature subset selection methods for Android malware detection by comparing and contrasting these methods along several factors. They utilized various learning algorithms to empirically evaluate the predictive accuracy of the feature subset selection methods and compare their predictive accuracy and execution times. The experiment findings demonstrated that feature selection is essential for increasing learning model accuracy and reducing runtime. The outcomes also illustrated that different learning algorithms perform differently when it comes to feature selection techniques, and no particular feature selection approach consistently outperforms the others.

Onwuzurike et al. [9] proposed a system that aims to detect Android malware from a behavioral perspective. It is based on abstracting the API calls executed by the application and building behavioral models through the use of Markov chains. The authors employed principal component analysis (PCA) to lower the dimensionality of the feature space and hence the computational and memory complexity of their system.

In [17], a novel method for selecting features called the selection of relevant attributes was developed by the authors in order to improve locally extracted features through the use of classical feature selectors (SAILS). This mechanism was constructed on top of conventional feature selection methods, including “mutual information”, “distinguishing feature selector”, and “Galavotti-Sebastiani-Simi”. It aimed to discover prominent system calls from Android applications.

Thiyagarajan et al. suggested in [8] a malware detection method based on the permissions requested by the application. Their idea was to minimize the data size by decreasing the number of permissions through a set of data reduction techniques (chi-square, permission ranking with a negative rate, support-based pruning, association-based pruning, and PCA). The reduced permissions were utilized for classifying the samples as malware or benign with a decision tree algorithm and categorizing the malware samples through the use of the K-means clustering algorithm.

### *2.2 Neural Network Based Methods*

In other cases, authors have preferred to benefit from the strength of neural networks for feature selection. For instance, Liu et al. [13] proposed an unsupervised feature learning method named “Subspace Based Restricted Boltzmann Machines” (SRBM) to reduce the data dimensionality in mobile malware detection. Their method includes searching for suitable subspaces over the entire feature set using a clustering method and learning the features in each feature subspace using Restricted Boltzmann Machines. Then, all learned features are concatenated to represent the original features in a lower dimension. The authors illustrated that their method outperforms other feature reduction methods including “RBM”, “Stacked Auto Encoder”, “Principal Components Analysis”, and “Agglomeration algorithms” in terms of clustering evaluation metrics.

In [14], Wu et al. proposed a feature reduction framework called DroidRL for Android malware detection. They used the Double Deep Q Network (DDQN) and the recurrent neural network (RNN) algorithms to select a valid subset of features over a larger range. They also attempted to determine the semantic relevance of features by using word embedding for the input features. The experiments conducted by the authors showed that their approach reduced the number of features from 1083 to 24 while maintaining high accuracy.

### *2.3 Tree Based Methods*

In [15], the authors proposed a method named DroidDomTree that searches the dominance tree of API calls included in the Android APK in order to find malicious modules. To efficiently select features, they developed a weighting scheme for assigning weights to each node in the dominance tree. This scheme aims to find the key modules that help in detecting malicious elements. In the experimental tests, the method presented detection rates between 98.1% and 99.3% when applied with eight machine learning classifiers. In another paper, Sharma et al. [16] have chosen a simple technique to reduce features based on calculating the importance of each feature using the feature_importances property of the Random Forest classifier, and then evaluating it using various machine learning algorithms.

Table 1 presents a comparative summary of the different works discussed in this section. These works denoted the importance of feature reduction methods in improving the performance of Android malware detection and classification systems. In the current paper, we tried to exploit a more innovative technique other than those described in the literature, that is, the Attention mechanism. Despite its importance, this mechanism is not yet well exploited in the field of Android malware detection. Among the few works done in this context, we can cite the paper by Wu et al. [18], in which the authors proposed a neural network approach to classify Android malware based on two layers: attention layer and multilayer perceptron (MLP). The attention layer is intended to learn feature weights, which can be thought of as scores of relevance between the features and classification outcomes. Then, the MLP maps the weighted features to classify the samples as benign or malicious. In this approach, the Attention-based feature importance is associated with an MLP classifier to classify Android samples. However, in our opinion, we believe that the classification process requires a more robust algorithm than a simple MLP. For this reason, we propose in this paper a new classification approach for android applications based on the Attention mechanism and the LightGBM algorithm. The Attention technique is utilized for determining the importance of features and reducing their dimensionality, while the LightGBM algorithm (with Bayesian optimization of hyperparameters) is utilized for performing an efficient classification of Android samples based on the set of selected features. To the best of our knowledge, the system that we propose in the current article is the first solution combining an Attention mechanism and a distributed gradient boosting framework like LightGBM to classify Android malware.

**Table 1:** Comparative summary of related work

| Paper reference | Approach objective | Used methods | Dataset type |
|---|---|---|---|
| Liu et al. [13] | Application of unsupervised feature learning to reduce data dimensionality in mobile malware dataset | Subspace based restricted Boltzmann machines | OmniDroid [19], CIC2019 [20] and CIC2020 [21] |
| Alam et al. [15] | Creation of dominance tree of API calls to improve the feature selection and the detection Android malware | Dominance tree, TF-IDF | Android applications collected from different sources |
| Cai et al. [10] | Optimization of feature weight-mapping to detect Android malware | Feature weighting, ML classifiers, differential evolution algorithm | Drebin [22], AMD [23], applications collected from Google Play and APKPure.com |

(Continued)

**Table 1 (continued)**

| Paper reference | Approach objective | Used methods | Dataset type |
|---|---|---|---|
| Wu et al. [14] | Feature reduction for Android malware detection and classification | DDQN, word embedding | AndroZoo [24] and Drebin [22] |
| Xie et al. [11] | Feature reduction for Android malware detection and classification | InfoGain, chi-square test, stacking and genetic algorithm | CIC-AndMal2017 [25] and CICMalDroid2020 [21] |
| Bai et al. [12] | Feature reduction for Android malware detection and classification | Fast correlation-based filter, catboost classifier | Drebin [22] and private dataset |
| Abawajy et al. [7] | Examine the effectiveness of the feature subset selection techniques for detecting Android malware | Pearson correlation coefficient, chi-square, analysis of variance (ANOVA), information gain, mutual information | Android applications collected from different sources |
| Onwuzurike et al. [9] | Detect Android malware by modeling application behavior | Markov chains, PCA | Android applications collected from different sources |
| Ananya et al. [17] | Feature selection for Android malware classification | SAILS, XGBoost, CART, logistic regression, random forest and deep neural networks | Drebin [22] |
| Thiyagarajan et al. [8] | Reduce the number of application permissions for real time malware detection and clustering | PCA, decision tree, K-means | AndroZoo [24] |
| Sharma et al. [16] | Android malware detection and family classification | Random forest, deep learning | AndroZoo [24] |
| Wu et al. [18] | Classify Android malware and interpret their malicious behaviors | Attention mechanism, multilayer perceptron | Drebin [22] |

## 3 Proposed Approach

In this section, we describe the approach we propose in this paper. The goal of this approach is to classify Android applications and detect those that are malware. It includes two major steps: (i) selecting the most important features based on an Attention mechanism and (ii) classifying Android applications as malware or normal through the use of an optimized LightGBM algorithm. The overall architecture of the proposed system is illustrated in Fig. 1. This system comprises 5 basic elements: Feature extraction, Attention-based feature importance, Feature selection, and LightGBM classification. The process starts with feature extraction from Android applications. Then, an Attention-based technique is applied to these features in order to identify the most important ones. This step assists in reducing the number of features and selecting only those that help enhance the performance of the classifier. Finally, an optimized LightGBM algorithm is applied to determine whether the application is malware or normal.

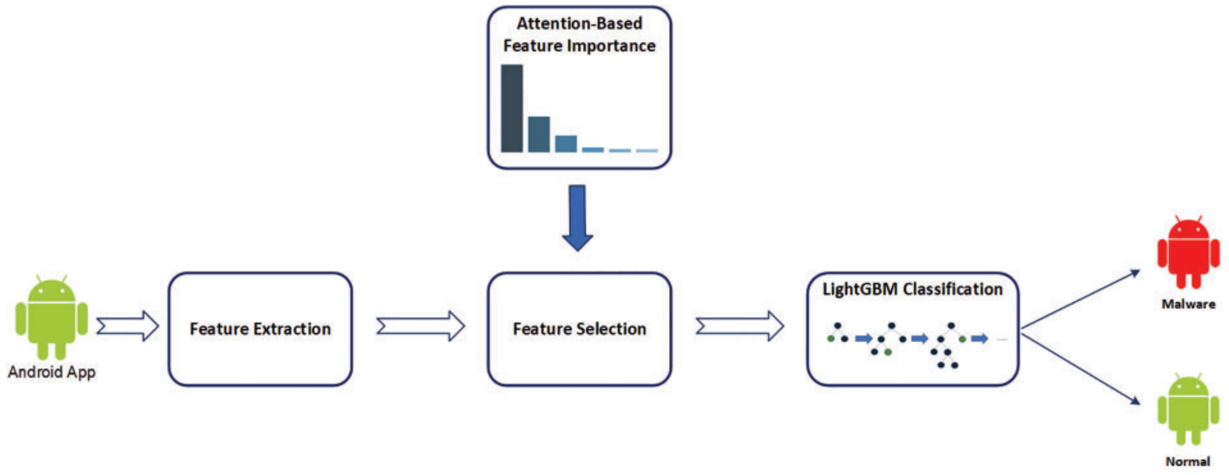

**Figure 1:** Approach architecture

### *3.1 Feature Extraction*

The feature extraction process is based on the technique utilized in the "CCCS-CIC-AndMal-2020" dataset [21] that we tested during the experimentation of the approach. This process consists in statically analyzing the Android application by reverse engineering its APK file. The extracted features contain a large set of information, including:

- Activities: the user interfaces of the Android app.
- Broadcast receivers and providers.
- Metadata: a method for storing information that can be accessed by application elements.
- Permissions indicating the restriction of access to data on the device.
- System features.

A feature vector is then created from the numerical values of the collected features. The dimension of the vector equals 9504.

### *3.2 Feature Selection*

Each instance of the dataset includes 9504 features. This number of features is very large and can impact the performance of the classifier in terms of speed and accuracy. In our approach, we decided to utilize a selection method to reduce the number of features and select only those that help in enhancing the performance of the classifier. We started by eliminating features with null or empty values, which reduced the number of features to only 5911. Then, we applied a feature selection algorithm based on the Attention mechanism. This idea aimed to calculate the weight of each feature through the use of a neural network that contains an Attention layer that allows paying attention to the features that are more crucial than the others in the classification process. The features with the highest scores are selected to take part in the classification task. The following paragraph describes in detail how this step works.

### *3.3 Attention-Based Feature Importance*

The Attention mechanism is a neural network concept that has gained much popularity in recent years and allows paying more attention to certain parts when processing data. It utilizes a weighted vector to help the neural architecture understand the relationship between the input elements and the target and estimate which part of the data is more important than others for the task at hand. In our approach, we used the Attention mechanism to select the most important features that help enhance the performance of the classifier.

The general architecture of the Attention-based feature importance is presented in Fig. 2. It is a neural network architecture. The input to this network is a feature vector containing the initial features of the model. This vector is connected to an attention layer to compute the weighted values of each feature based on its importance to the classification process. In this way, a weighted feature vector is created, which is then connected to a Dense layer. The purpose of the Dense layer is to compute a "*y*" score in order to predict whether the instance is classified as malware or normal. After the training process of the neural attention network is complete, the weighted feature vector is utilized for determining which features have higher weighted values than the others. The features with higher weighted values mean that they are more crucial for the classification task.

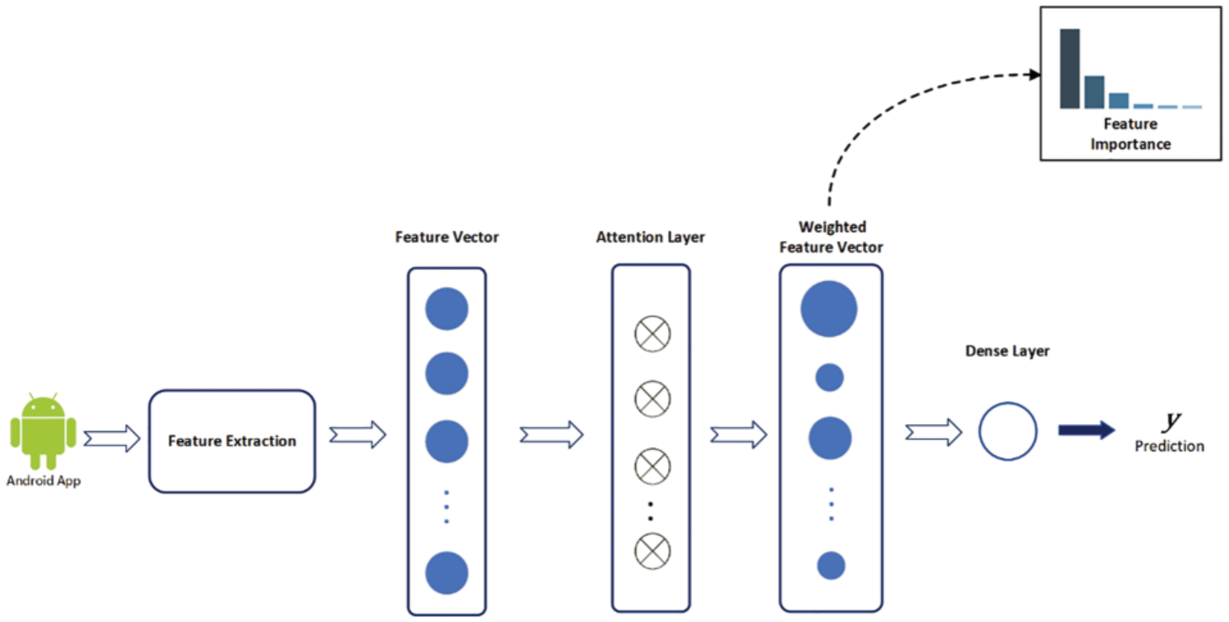

**Figure 2:** Attention-based feature importance

In order to explain the feature extraction process, consider $X$ as the set of feature vectors extracted from the Android samples and $x_i$ as the $i$-th sample of the set $X$ denoted as $\left(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(j)}, \ldots, x_i^{(N)}\right)$, where $x_i^{(j)}$ $(1 \leq j \leq N)$ refers to the $j$-th feature of the $i$-th sample. Each feature vector $x_i$ is assigned a label value $y_i \in \{0, 1\}$, where 0 means that the sample is classified as normal and 1 means that it is classified as malware.

The implementation of the Attention layer is inspired by the work of Wu et al. [18]. It consists of using an adapted fully connected network and a SoftMax function to calculate the weight of each feature. We started by using the following equation to evaluate how closely the output feature matches the input feature:

$$e_i^{(j)} = \sum_{k=1}^{N} x_i^{(k)} w_{kj} \tag{1}$$

where $w_{kj}$ represents a weighting parameter learned during the training of the fully connected network in the attention layer and $e_i^{(j)}$ denotes the output of the fully connected network at the $j$-th position, which can be viewed as an association of a set of features with varying relevance to the input feature at position $j$. Thus, training the neural model allows the parameter $w_{kj}$ to be assigned a relevant value to indicate the relationship between the $j$-th input feature and other input features.

Next, in order to determine the weights of the input features at various places, we apply a SoftMax function to the output of the fully connected network. The output of the attention layer is a vector denoted by $\alpha_i = \left(\alpha_i^{(1)}, \alpha_i^{(2)}, \alpha_i^{(3)}, \ldots, \alpha_i^{(N)}\right)$ calculated as follows:

$$\alpha_i^{(j)} = \frac{\exp\left(e_i^{(j)}\right)}{\sum_{k=1}^{N} \exp\left(e_i^{(k)}\right)} \tag{2}$$

where $\alpha_i^{(j)}$ denotes the weight of the $j$-th feature in the $i$-th sample and reflects its importance based on the classification results.

Then, in order to generate the weighted feature vector, we weight the input feature vector by the Attention vector $\alpha_i$ as follows:

$$c_i = \alpha_i x_i^T \tag{3}$$

where $c_i$ denotes the weighted feature vector of the $i$-th sample.

Finally, we calculate the classification result $y_i$ by mapping the input vector $c_i$ into a binary prediction value.

Once the training process is complete, the Attention-based model assigns different weights to all features based on their contribution to the classification results. The features with higher weights indicate that they are more important for the relevance of the classification results, while the features with lower weights are less important. The weighted feature vector assists in selecting the top $n$ features. These features are considered more crucial than the others, and our main classifier (LightGBM) is applied to them. The choice of the parameter $n$ affects the classification results, so it is essential to select a suitable value. In our case, we tried to choose the best value for $n$ based on the results of the experiments performed.

### *3.4 LightGBM Classification*

LightGBM [26] is a distributed gradient boosting algorithm released by Microsoft in 2017 for machine learning tasks. LightGBM is based on decision trees and can be used for several machine learning tasks such as classification, ranking and regression. It uses leaf-wise tree growth instead of the level-wise-tree growth which is widely used in several tree-based learning algorithms. LightGBM has outperformed several machine learning algorithms in multiple applications thanks to its characteristics, including efficiency, speed, and low memory consumption. In our model, LightGBM plays a role in classifying the Android samples as normal or malware after selecting the most important features from the previous step.

The implementation of LightGBM requires the use of a set of parameters called hyper-parameters, such as the number of leaves per tree, the maximum tree depth, and the learning rate. The mentioned parameters significantly affect how well the LightGBM algorithm performs and produces results. The choice of these hyper-parameters is crucial to achieving good results [27]. In our approach, a Bayesian optimization technique was utilized for identifying the best parameters for this model.

Bayesian optimization is a useful technique to optimize black-box functions that are expensive to evaluate [28,29]. The optimization problem can be formulated as follows:

$$x^* = \operatorname{argmax}_{x \in \chi} f(x) \quad (4)$$

where $x^*$ represents the LightGBM model's hyper-parameters that need to be optimized. The symbol $\chi$ denotes the search space for the hyper-parameters. The objective function is denoted by $f(x)$ and indicates how well the LightGBM model performs given the selected hyper-parameters. Accuracy is the measurement criterion we chose to evaluate how well the objective function performed. Therefore, the goal of the optimization is to determine the collection of hyper-parameters $x^*$ with which the function $f(x)$ performs best. The optimization procedure usually involves numerous iterations. The objective function yields an observed result $y_i = f(x_i)$ that will be added to the historical set $D = (x_1, y_1), \ldots, (x_i, y_i)$ and utilized for updating the surrogate probability model in order to generate the next proposal. The optimization procedure for the LightGBM model is presented in Algorithm 1.

**Algorithm 1:** Description of the Bayesian optimization with LightGBM

```
foreach iteration $i$ do
1) Select a new configuration of the hyper-parameters $x_{i+1}$ based on the function of acquisition $\alpha(x)$,
   $x_{i+1} = argmax\alpha(x, D_n)$
2) Determine the performance of $f$ based on the current configuration $x_{i+1}$: $y_{i+1} = f(x_{i+1})$
3) Update the set $D$ by adding the current result: $D_{i+1} = D_i, (x_{i+1}, y_{i+1})$
4) Update the surrogate probability model related to the objective function
```

After completing the optimization process and selecting the best hyperparameters, the classification process of the LightGBM model starts to identify whether the Android sample should be classified as normal or malware.

## 4 Experimental Evaluation

The objective of this section is to evaluate the performance of our approach based on several experimental tests. This evaluation consists in comparing the classification results of LightGBM and other machine learning models before and after the application of feature reduction based on the Attention mechanism.

### *4.1 Dataset Description*

In order to test and assess our approach, we utilized the CCCS-CIC-AndMal-2020 dataset [21]. This is a recent Android malware dataset created by the Canadian Institute for Cybersecurity (CIC). The dataset comprises 400 K Android applications (200 K are benign and 200 K are malware). The Android malware data is divided into the following 14 malware categories: Adware, Backdoor, FileInfector, No_Category, Potentially Unwanted Apps (PUA), Ransomware, Riskware, Scareware, Trojan, Banker Trojan, Dropper Trojan, SMS Trojan, Spy Trojan, and Zero-Day. Table 2 presents the number of families and samples for each of the 14 malware categories. In the Adware category, for example, there are 47,210 samples in the dataset that belong to this type of malware. Also in this category, we found 48 malware families, including Dowgin, Adflex, Airpush, Baiduprotect, etc. In the experimental tests, 12 malware categories were utilized, with the exception of No_Category and Zero

Day, because the data in these categories was incomplete. The features of the dataset contain a lot of information, including:

- Activities: the user interfaces of the Android app.
- Broadcast receivers and providers.
- Metadata: a method for storing information that can be accessed by application elements.
- Permissions indicating the restriction of access to data on the device.
- System features.

**Table 2:** CCCS-CIC-AndMal-2020 dataset details

| Class | Number of families | Amount of samples |
|---|---|---|
| Adware | 48 | 47,210 |
| No category | – | 2296 |
| PUA | 8 | 2051 |
| Backdoor | 11 | 1538 |
| Ransomware | 8 | 6202 |
| File infector | 5 | 669 |
| Riskware | 21 | 97,349 |
| Scareware | 3 | 1556 |
| Dropper trojan | 9 | 2302 |
| Banker trojan | 11 | 887 |
| Spy trojan | 11 | 3540 |
| SMS trojan | 11 | 3125 |
| Trojan | 45 | 13,559 |
| Zero day | – | 13,340 |

## *4.2 Parameters of the LightGBM Classifier*

The hyperparameters of the LightGBM classifier greatly affect the quality of the classification results. The integration of Bayesian optimization into this approach has remarkably helped us in carefully selecting the right parameters. To accomplish this task, we have selected a set of values for each parameter of the LightGBM classifier (Learning rates, Number of iterations, Number of leaves, Bagging fraction, Feature fraction, Min data in leaf and Max depth). We then enter these values as input to Algorithm 1 described above. The role of this optimization algorithm is to try different configurations of these parameters in several iterations with the aim of obtaining an optimal configuration for the best classification accuracy. Table 3 describes the selected hyperparameters for this classification model.

## *4.3 Experimental Results of Binary Classification*

The classification model used in this approach plays the role of a binary classifier whose goal is to classify Android samples into two classes: normal and malware. For this purpose, we grouped all the samples in the dataset belonging to the different malware families into a single class labeled "Malware", and the other class labeled "Normal" is intended for benign samples. In order to train and assess the model, we divided the dataset as follows: 80% for training and 20% for testing. The experiments conducted in this work include 7 classification models (LightGBM, Random Forest, AdaBoost, Naive Bayes, Decision Tree, XGBoost and K-Nearest-Neighbor). For each model we performed 5 tests with the features of the dataset: applying the model (i) to the totality of the features without reduction (5911 features), (ii) to the 200 best features of the dataset selected thanks to our Attention-based feature importance method, (iii) to the 300 best features, (iv) to the 500 best features, and finally (v) to the 1000 best features. The aim is to evaluate the effectiveness of the Attention-based feature importance method in association with the LightGBM model (as well as the other machine learning algorithms) and to identify whether it can contribute to the improvement of the classification model performance. In order to evaluate the performed experimentations, we calculated 5 measures: accuracy, precision, recall, F1-score and false alarm rate (FAR) as well as the speed of training.

**Table 3:** Hyperparameters of the LightGBM classifier

| Name of the hyperparameter | Value |
|---|---|
| Boosting method | gbdt |
| Learning rates | 0.1 |
| Number of iterations | 588 |
| Number of leaves | 453 |
| Bagging fraction | 0.8 |
| Feature fraction | 0.5 |
| Min data in leaf | 50 |
| Max depth | 15 |

Table 4 illustrates the results of the experiments described above. Concerning LightGBM, the model applied to the top 300 features (in terms of importance) obtained the best results in the 5 evaluation measures with an accuracy of 0.987110, a precision of 0.988937, a recall of 0.982342, an F1-score of 0.985628, and a FAR of 0.008990. These scores slightly exceeded the results obtained from the LightGBM model applied to the totality of features without reduction (5911 features). The latter exhibited an accuracy of 0.986462, a precision of 0.988772, a recall of 0.981053, an F1-score of 0.984897, and a FAR of 0.009114. Furthermore, the LightGBM model with 300 features took less time to complete the training with 28.099 s in comparison with 80.043 s for the LightGBM model without the feature reduction. Fig. 3 presents the ROC curve and the confusion matrix of the LightGBM model with the top 300 features.

Concerning the other classification models, the application of our feature reduction technique based on the Attention mechanism showed better classification performance with all algorithms except Naive Bayes. For example, for Random Forest, the use of the 200 best features showed an accuracy equal to 0.986086, exceeding that with all the features of the dataset. For AdaBoost, similarly, the use of the top 200 and 300 features showed better performance in terms of accuracy, recall, F1-score and training time. Regarding the Decision Tree algorithm, we obtained the best results with the top 500 features. The selection of the first 300 features with the XGBoost model showed the best results in terms of accuracy, precision, recall, F1-score, FAR and training time. For the K-Nearest-Neighbor model, accuracy reached its maximum with the top 1000 features. The only case where the feature reduction technique failed to improve the classification performance was with the Naive Bayes algorithm.

**Table 4:** Results obtained from the classification models

| Classification model | Number of features | Accuracy | Precision | Recall | F1-score | FAR | Training time (s) |
|---|---|---|---|---|---|---|---|
| LightGBM | Without feature reduction | 0.986462 | 0.988772 | 0.981053 | 0.984897 | 0.009114 | 80.043 |
| | 200 features | 0.986598 | 0.988402 | 0.981736 | 0.985058 | 0.009424 | 25.056 |
| | 300 features | 0.987110 | 0.988937 | 0.982342 | 0.985628 | 0.008990 | 28.099 |
| | 500 features | 0.986564 | 0.988700 | 0.981357 | 0.985014 | 0.009176 | 34.318 |
| | 1000 features | 0.986427 | 0.988696 | 0.981053 | 0.984860 | 0.009176 | 35.581 |
| Random forest | Without feature reduction | 0.985643 | 0.990553 | 0.977416 | 0.983940 | 0.007626 | 410.918 |
| | 200 features | 0.986086 | 0.989286 | 0.979689 | 0.984464 | 0.008680 | 26.052 |
| | 300 features | 0.985814 | 0.990030 | 0.978325 | 0.984143 | 0.008060 | 32.892 |
| | 500 features | 0.985268 | 0.988144 | 0.979007 | 0.983554 | 0.009610 | 39.334 |
| | 1000 features | 0.985848 | 0.990256 | 0.978174 | 0.984178 | 0.007874 | 55.863 |
| AdaBoost | Without feature reduction | 0.957407 | 0.964897 | 0.939523 | 0.952041 | 0.027962 | 1197.214 |
| | 200 features | 0.958839 | 0.963286 | 0.944524 | 0.953813 | 0.029450 | 20.893 |
| | 300 features | 0.959146 | 0.962455 | 0.946116 | 0.954215 | 0.030194 | 26.718 |
| | 500 features | 0.957509 | 0.957509 | 0.940205 | 0.952184 | 0.028334 | 38.433 |
| | 1000 features | 0.957407 | 0.964897 | 0.939523 | 0.952041 | 0.027962 | 68.769 |
| Naive bayes | Without feature reduction | 0.824137 | 0.775236 | 0.857901 | 0.814476 | 0.203484 | 13.823 |
| | 200 features | 0.815680 | 0.764390 | 0.853429 | 0.806460 | 0.215202 | 0.197 |
| | 300 features | 0.817078 | 0.766633 | 0.853202 | 0.807604 | 0.212474 | 0.203 |
| | 500 features | 0.821102 | 0.770724 | 0.857522 | 0.811809 | 0.208692 | 0.239 |
| | 1000 features | 0.823353 | 0.774280 | 0.857370 | 0.813709 | 0.204476 | 0.530 |
| Decision tree | Without feature reduction | 0.975208 | 0.970562 | 0.974460 | 0.972507 | 0.024180 | 88.788 |
| | 200 features | 0.974355 | 0.969016 | 0.974157 | 0.971580 | 0.025482 | 3.736 |
| | 300 features | 0.974253 | 0.969434 | 0.973475 | 0.973475 | 0.025110 | 5.289 |
| | 500 features | 0.975447 | 0.970293 | 0.975294 | 0.975294 | 0.024428 | 9.415 |
| | 1000 features | 0.975344 | 0.970428 | 0.974915 | 0.972666 | 0.024304 | 12.959 |
| XGBoost | Without feature reduction | 0.980835 | 0.983024 | 0.974233 | 0.978608 | 0.013764 | 1438.581 |
| | 200 features | 0.981210 | 0.982669 | 0.975445 | 0.979044 | 0.014074 | 55.962 |
| | 300 features | 0.982062 | 0.983587 | 0.976430 | 0.979995 | 0.013330 | 81.079 |
| | 500 features | 0.980596 | 0.982350 | 0.974384 | 0.978351 | 0.014322 | 133.677 |
| | 1000 features | 0.980971 | 0.982512 | 0.975066 | 0.978775 | 0.014198 | 253.358 |
| K-Nearest-Neighbor | Without feature reduction | 0.932376 | 0.945841 | 0.901326 | 0.923047 | 0.042222 | 1641.555 |
| | 200 features | 0.932035 | 0.945372 | 0.901023 | 0.922665 | 0.042594 | 74.515 |
| | 300 features | 0.932172 | 0.945815 | 0.900872 | 0.922796 | 0.042222 | 94.963 |
| | 500 features | 0.932206 | 0.945748 | 0.901023 | 0.922844 | 0.042284 | 150.967 |
| | 1000 features | 0.932410 | 0.945987 | 0.901250 | 0.923077 | 0.042098 | 285.270 |

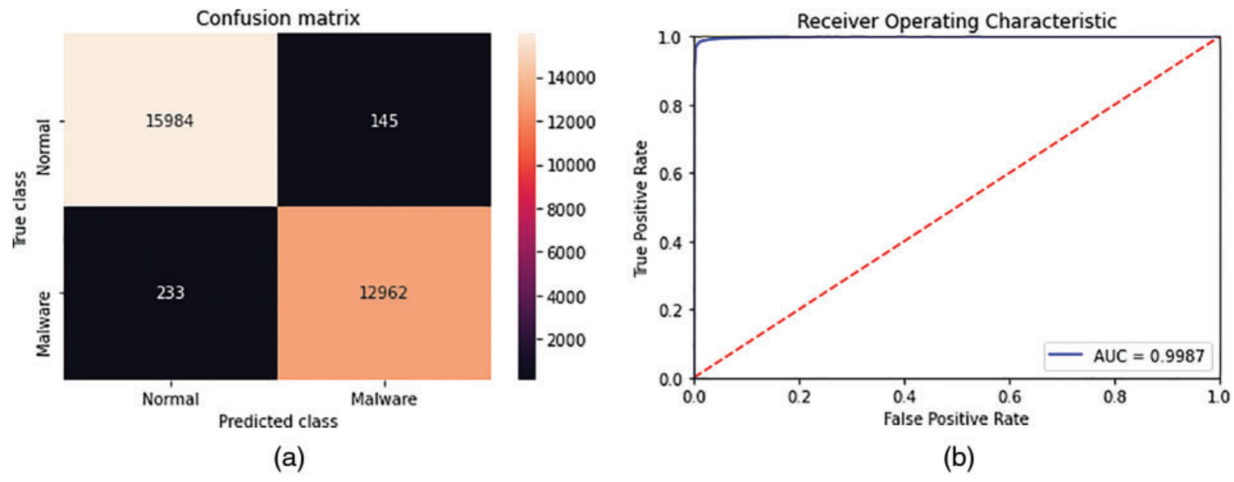

**Figure 3:** ROC curve and confusion matrix of the LightGBM model with the top 300 features. (a) Confusion matrix. (b) ROC curve

Comparing all the results obtained from the different experiments, we can conclude that the LightGBM model with the top 300 features is the best performing model for the Android malware dataset. These comparative results confirm that the feature importance technique based on the Attention mechanism proposed in this paper has shown its ability to improve the classification results of the LightGBM model. The strength of this technique is that it can pay attention to features that play an important role in the classification results and ignore those that are less important. This technique has both reduced the training time and improved the classification results.

### *4.4 Experimental Results of Malware Category Classification*

In this part, we are looking to evaluate our approach in the context of a multiclass classification. Our target is to classify the malware samples of the dataset into 12 malware categories as previously mentioned in the description of the dataset. This experiment consists in conducting a comparative evaluation among (i) the LightGBM model without feature reduction, (ii) the LightGBM model with selection of the best features as we did in the previous experiment, and (iii) the model described by the authors of the dataset paper (deep learning model). The benchmarking comprises 4 measures: precision, recall, and F1-score of each malware category as well as the overall model accuracy.

Table 5 shows the classification results of the different models tested in this comparison. This time the LightGBM model applied to the 500 best features provided the best results with an accuracy of 0.947711 in comparison with 0.946933 for LightGBM without feature reduction, 0.946349 for LightGBM with top 200 features, 0.946960 for LightGBM with top 300 features, 0.947461 for LightGBM with top 1000 features, and 0.93 for the deep learning model of the authors of the dataset.

These results demonstrated once again the effectiveness of the feature importance technique based on the Attention mechanism. This effectiveness is also associated with the choice of the value *n* that determines the number of important features to be selected. Choosing this number carefully is very essential to avoid ignoring some important features or including some features that have a negative effect on the classification results.

**Table 5:** Results of the malware category classification

| | Metrics | Adware | Backdoor | Banker Trojan | Dropper Trojan | File infector | PUA | Ransomware | Riskware | SMS trojan | Scareware | Spy trojan | Trojan | Accuracy | Training speed |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LightGBM without feature reduction | Precision | 0.92 | 0.85 | 0.87 | 0.85 | 0.94 | 0.86 | 0.81 | 0.98 | 0.94 | 0.97 | 0.92 | 0.94 | 0.9469 | 374.56 |
| | Recall | 0.96 | 0.75 | 0.84 | 0.70 | 0.77 | 0.66 | 0.92 | 0.97 | 0.94 | 0.75 | 0.88 | 0.91 | | |
| | F1-score | 0.94 | 0.79 | 0.85 | 0.77 | 0.85 | 0.74 | 0.86 | 0.97 | 0.94 | 0.85 | 0.90 | 0.92 | | |
| LightGBM (200 features) | Precision | 0.92 | 0.86 | 0.87 | 0.85 | 0.94 | 0.87 | 0.81 | 0.98 | 0.94 | 0.97 | 0.92 | 0.94 | 0.9463 | 283.4 |
| | Recall | 0.96 | 0.74 | 0.85 | 0.70 | 0.77 | 0.66 | 0.92 | 0.97 | 0.94 | 0.75 | 0.88 | 0.91 | | |
| | F1-score | 0.94 | 0.79 | 0.86 | 0.77 | 0.85 | 0.75 | 0.86 | 0.97 | 0.94 | 0.85 | 0.90 | 0.92 | | |
| LightGBM (300 features) | Precision | 0.92 | 0.85 | 0.87 | 0.85 | 0.94 | 0.86 | 0.81 | 0.98 | 0.94 | 0.97 | 0.92 | 0.94 | 0.9467 | 289.59 |
| | Recall | 0.96 | 0.74 | 0.85 | 0.70 | 0.78 | 0.66 | 0.92 | 0.97 | 0.94 | 0.75 | 0.88 | 0.91 | | |
| | F1-score | 0.94 | 0.79 | 0.86 | 0.77 | 0.85 | 0.75 | 0.86 | 0.97 | 0.94 | 0.84 | 0.90 | 0.92 | | |
| LightGBM (500 features) | Precision | 0.92 | 0.86 | 0.86 | 0.86 | 0.93 | 0.88 | 0.81 | 0.98 | 0.94 | 0.97 | 0.92 | 0.94 | 0.9477 | 311.01 |
| | Recall | 0.96 | 0.74 | 0.85 | 0.71 | 0.76 | 0.67 | 0.92 | 0.97 | 0.93 | 0.75 | 0.88 | 0.91 | | |
| | F1-score | 0.94 | 0.80 | 0.86 | 0.77 | 0.84 | 0.76 | 0.86 | 0.97 | 0.94 | 0.85 | 0.90 | 0.93 | | |
| LightGBM (1000 features) | Precision | 0.92 | 0.85 | 0.87 | 0.85 | 0.94 | 0.88 | 0.81 | 0.98 | 0.94 | 0.97 | 0.92 | 0.94 | 0.9475 | 350.16 |
| | Recall | 0.96 | 0.75 | 0.85 | 0.70 | 0.78 | 0.66 | 0.92 | 0.97 | 0.94 | 0.75 | 0.88 | 0.91 | | |
| | F1-score | 0.94 | 0.79 | 0.86 | 0.77 | 0.85 | 0.75 | 0.86 | 0.97 | 0.94 | 0.84 | 0.90 | 0.92 | | |
| DiDroid [18] | Precision | 0.935 | 0.721 | 0.759 | 0.85 | 0.909 | 0.677 | 0.798 | 0.963 | 0.917 | 0.836 | 0.924 | 0.895 | 0.93 | – |
| | Recall | 0.929 | 0.643 | 0.759 | 0.686 | 0.789 | 0.682 | 0.944 | 0.967 | 0.886 | 0.764 | 0.835 | 0.896 | | |
| | F1-score | 0.932 | 0.68 | 0.759 | 0.759 | 0.845 | 0.679 | 0.864 | 0.965 | 0.901 | 0.799 | 0.877 | 0.896 | | |

## 4.5 Discussion

In recent years, a number of studies have been conducted with the aim of proposing relevant solutions for the identification and classification of malware in Android mobile environments. Table 6 compares some of these studies, including the one we propose in this paper. To achieve a fair and equitable comparison, we only selected the works that were evaluated using the CCCS-CIC-AndMal-2020 dataset. Different techniques were used in these works, including classical machine learning algorithms such as Random Forest and SVM as well as deep learning algorithms such as CNN and LSTM. The comparison showed that our approach outperformed the other works in terms of binary and multi-class classification accuracy.

The experimental results shown that our proposed approach is very effective. This approach made it possible to minimize the execution time of the model by effectively reducing the dimensionality of the data while improving the accuracy of the classification. The experimental findings have also shown that the LightGBM model performs better than other machine learning algorithms on this type of dataset.

Although the proposed approach has been shown to be effective with the CCCS-CIC-AndMal-2020 dataset and although we believe it can be properly adapted to other types of datasets, it would be more appropriate to test this approach with other Android malware samples. This will help to thoroughly investigate the performance of the Attention mechanism and verify its ability to efficiently analyze other types of features. Another limitation of this approach must be considered in our future work. The data collected in the CCCS-CIC-AndMal-2020 dataset contains static analysis of Android applications. This static analysis is very useful in identifying Android malware due to the large number of features we can extract from the application APK file. However, it is not able to detect complex Android malwares as their malicious actions can only be observed during execution. Therefore, it is important to extend the current approach to support hybrid application analysis that includes static data from the application and dynamic monitoring of its behavior at runtime.

**Table 6:** Comparison with studies using the CCCS-CIC-AndMal-2020 dataset

| Paper reference | Year | Used methods | Accuracy | |
|---|---|---|---|---|
| | | | Binary classification | Multiclass classification |
| Musikawan et al. [30] | 2023 | DNN | 0.9772 | – |
| Batouche et al. [31] | 2021 | Random forest | – | 0.89 |
| Chopra et al. [32] | 2023 | CNN, transfer learning | 0.9719 | – |
| Ullah et al. [33] | 2022 | SVM | 0.9664 | – |
| DiDroid [21] | 2020 | CNN | – | 0.93 |
| Wang et al. [34] | 2023 | Bidirectional LSTM | – | 0.92 |
| Our approach | 2024 | Attention mechanism, LightGBM | **0.9871** | **0.9477** |

## 5 Conclusion

In this study, we presented a solution for Android malware detection. Two recent ML techniques were employed in the solution: the Attention mechanism and the LightGBM classifier. The Attention mechanism was integrated with a neural network to analyze the dataset's features and identify which are important for the classification results. The LightGBM algorithm was chosen for classifying the samples of the dataset based on a set of features selected according to their importance. The advantage of our solution is its ability to reduce the size of the features, subsequently minimizing the execution time, and also improving the accuracy of the classification algorithm. Experimental results demonstrated that the feature importance technique enhanced the classification accuracy from 98.64% (without feature reduction) to 98.71% (after feature selection).

Additionally, we tested the proposed approach on the CCCS-CIC-AndMal-2020 dataset, focusing on static analysis of Android applications. In the future, we plan to investigate other malware datasets and improve our approach in order to support hybrid features obtained from static and dynamic analysis of Android applications. We also intend to develop a new method for determining the key features that characterize the behavior of each malware family so that we could assist security experts in identifying these malicious applications.

**Acknowledgement:** The authors extend their appreciation to the Deanship of Graduate Studies and Scientific Research at Jouf University for funding this work.

**Funding Statement:** This work was funded by the Deanship of Graduate Studies and Scientific Research at Jouf University under Grant No. (DGSSR-2023-02-02178).

**Availability of Data and Materials:** The data that support the findings of this study are openly available at https://www.unb.ca/cic/datasets/andmal2020.html (accessed on 15 January 2024).

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

## References

[1] Annie, "The state of mobile in 2022," 2022. Accessed: Jan. 15, 2024. [Online]. Available: https://www.data.ai/en/insights/market-data/state-of-mobile-2022/

[2] StatCounter, "Mobile operating system market share worldwide," 2022. Accessed: Jan. 15, 2024. [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/worldwide

[3] T. Shishkova and A. Kivva, "Mobile malware evolution 2021," 2022. Accessed: Jan. 15, 2024. [Online]. Available: https://securelist.com/mobile-malware-evolution-2021/105876/

[4] A. Nazir *et al.*, "A deep learning-based novel hybrid CNN-LSTM architecture for efficient detection of threats in the IoT ecosystem," *Ain Shams Eng. J.*, vol. 15, no. 7, pp. 102777, Apr. 2024. doi: 10.1016/j.asej.2024.102777.

[5] A. Nazir *et al.*, "Advancing IoT security: A systematic review of machine learning approaches for the detection of IoT botnets," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 35, no. 10, pp. 101820, Dec. 2023. doi: 10.1016/j.jksuci.2023.101820.

[6] R. H. Hadi, H. N. Hady, A. M. Hasan, A. Al-Jodah, and A. J. Humaidi, "Improved fault classification for predictive maintenance in industrial IoT based on AutoML: A case study of ball-bearing faults," *Processes*, vol. 11, no. 5, pp. 1507, May 2023. doi: 10.3390/pr11051507.

[7] J. Abawajy, A. Darem, and A. A. Alhashmi, "Feature subset selection for malware detection in smart IoT platforms," *Sensors*, vol. 21, no. 4, pp. 1374, Feb. 2021. doi: 10.3390/s21041374.

[8] J. Thiyagarajan, A. Akash, and B. Murugan, "Improved real-time permission based malware detection and clustering approach using model independent pruning," *IET Inf. Secur.*, vol. 14, no. 5, pp. 531–541, Mar. 2020. doi: 10.1049/iet-ifs.2019.0418.

[9] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross and G. Stringhini, "MaMaDroid: Detecting Android malware by building markov chains of behavioral models," *ACM Trans. Priv. Secur.*, vol. 22, no. 2, pp. 1–34, Apr. 2019. doi: 10.1145/3313391.

[10] L. Cai, Y. Li, and Z. Xiong, "JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters," *Comput. Secur.*, vol. 100, no. 7, pp. 102086, Jan. 2021. doi: 10.1016/j.cose.2020.102086.

[11] N. Xie, Z. Qin, and X. Di, "GA-StackingMD: Android malware detection method based on genetic algorithm optimized stacking," *Appl. Sci.*, vol. 13, no. 4, pp. 2629, Jan. 2023. doi: 10.3390/app13042629.

[12] H. Bai, N. Xie, X. Di, and Q. Ye, "FAMD: A fast multifeature Android malware detection framework, design, and implementation," *IEEE Access*, vol. 8, pp. 194729–194740, 2020. doi: 10.1109/ACCESS.2020.3033026.

[13] Z. Liu, R. Wang, N. Japkowicz, D. Tang, W. Zhang and J. Zhao, "Research on unsupervised feature learning for Android malware detection based on restricted Boltzmann machines," *Future Gener. Comput. Syst.*, vol. 120, no. 5, pp. 91–108, Jul. 2021. doi: 10.1016/j.future.2021.02.015.

[14] Y. Wu *et al.*, "DroidRL: Feature selection for Android malware detection with reinforcement learning," *Comput. Secur.*, vol. 128, no. 1, pp. 103126, May 2023. doi: 10.1016/j.cose.2023.103126.

[15] S. Alam, S. A. Alharbi, and S. Yildirim, "Mining nested flow of dominant APIs for detecting Android malware," *Comput. Netw.*, vol. 167, no. 1, pp. 107026, Feb. 2020. doi: 10.1016/j.comnet.2019.107026.

[16] S. Sharma, P. Ahlawat, and K. Khanna, "DeepMDFC: A deep learning based Android malware detection and family classification method," *Secur. Priv.*, vol. 7, no. 2, pp. 23, Oct. 2023. doi: 10.1002/spy2.347.

[17] A. Ananya, A. Aswathy, T. R. Amal, P. G. Swathy, P. Vinod and S. Mohammad, "SysDroid: A dynamic ML-based Android malware analyzer using system call traces," *Clust. Comput.*, vol. 23, no. 4, pp. 2789–2808, Jan. 2020. doi: 10.1007/s10586-019-03045-6.

[18] B. Wu *et al.*, "Why an android app is classified as malware," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 2, pp. 1–29, Mar. 2021. doi: 10.1145/3423096.

[19] A. Martin, R. Lara-Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," *Inf. Fusion*, vol. 52, no. 7, pp. 128–142, Dec. 2019. doi: 10.1016/j.inffus.2018.12.006.

[20] L. Taheri, A. F. Kadir, and A. H. Lashkari, "Extensible Android malware detection and family classification using network-flows and API-calls," in *Int. Carnahan Conf. Secur. Technol. (ICCST)*, Chennai, India, Oct. 2019. doi: 10.1109/ccst.2019.8888430.

[21] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. GAGNON, and F. Massicotte, "DIDroid: Android malware classification and characterization using deep image learning," in *2020 the 10th Int. Conf. Commun. Netw. Secur.*, Tokyo, Japan, Nov. 2020. doi: 10.1145/3442520.3442522.

[22] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. 2014 Netw. Distrib. Syst. Secur. Symp.*, San Diego,CA, USA, 2014. doi: 10.14722/ndss.2014.23247.

[23] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Int. Conf. Detect. Intrusions Malware Vulnerability Assess.*, Bonn, Germany, 2017, vol. 10327, 10.1007/978-3-319-60876-1_12.

[24] K. Allix, T. F. Bissyandé, J. Klein, and Y. L. Traon, "AndroZoo: Collecting millions of android apps for the research community," in *2016 IEEE/ACM 13th Work. Conf. Min. Softw. Repos. (MSR)*, Austin, TX, USA, 2016, pp. 468–471.

[25] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark Android malware datasets and classification," in *2018 Int. Carnahan Conf. Secur. Technol. (ICCST)*, Montreal, QC, Canada, 2018, pp. 1–7. doi: 10.1109/CCST.2018.8585560.

[26] G. Ke *et al.*, "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. 31st Int. Conf. Neural Inform. Process. Syst. (NIPS'17)*, Long Beach, CA, USA, 2017, pp. 3149–3157.

[27] A. Ghourabi, "A security model based on LightGBM and transformer to protect healthcare systems from cyberattacks," *IEEE Access*, vol. 10, pp. 48890–48903, 2022. doi: 10.1109/ACCESS.2022.3172432.

[28] E. Brochu, V. M. Cora, and Nando de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," arXiv preprint arXiv:1012.2599, Dec. 2010.

[29] J. van Hoof and J. Vanschoren, "Hyperboost: Hyperparameter optimization by gradient boosting surrogate models," arXiv preprint arXiv:2101.02289, 2021.

[30] P. Musikawan, Y. Kongsorot, I. You, and C. So-In, "An enhanced deep learning neural network for the detection and identification of Android malware," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 8560–8577, 15 May, 2023. doi: 10.1109/JIOT.2022.3194881.

[31] A. Batouche and H. Jahankhani, "A comprehensive approach to Android malware detection using machine learning," in *Information Security Technologies for Controlling Pandemics*. Cham: Springer, 2021.

[32] R. Chopra, S. Acharya, U. Rawat, and R. Bhatnagar, "An energy efficient, robust, sustainable, and low computational cost method for mobile malware detection," *Appl. Comput. Intell. Soft Comput.*, vol. 2023, pp. e2029064, 2023. doi: 10.1155/2023/2029064.

[33] S. Ullah, T. Ahmad, A. Buriro, N. Zara, and S. Saha, "TrojanDetector: A multi-layer hybrid approach for trojan detection in android applications," *Appl. Sci.*, vol. 12, no. 21, pp. 10755, Jan. 2022. doi: 10.3390/app122110755.

[34] X. Wang, J. Liu, and C. Zhang, "Network intrusion detection based on multi-domain data and ensemble-bidirectional LSTM," *EURASIP J. Inf. Secur.*, vol. 2023, no. 1, pp. 5, Jun. 2023. doi: 10.1186/s13635-023-00139-y.