Ibn Al-Haitham Journal for Pure and Applied Sciences
Journal homepage: jih.uobaghdad.edu.iq
PISSN: 1609-4042, EISSN: 2521-3407
IHJPAS. 2026; 39(2)

Classification of Hybrid Malware on Android: The Significance of Feature Importance in Decision Tree Analysis

Nabaa Kareem Mhalhal[1*]
[1]Directorate of Education, Miysan Governorate, Miysan, Iraq.
*Corresponding Author

Received:2/ September/ 2025
Accepted:31/December/2025
Published: 20/April/2026
doi.org/10.30526/39.2.4293

Abstract

Malicious applications provide a growing and significant threat to Android users, developers, and application platforms. Experts have endeavored to create novel detection methodologies due to the ongoing advancement in the sophistication of malware and the escalating intensity of its damaging assaults. Within the framework of these initiatives, the detection of malware encounters a significant impediment due to the lack of clean and balanced datasets. This research seeks to develop a model proficient in identifying and categorizing various forms of hybrid malware on the Android platform, utilizing the Decision Tree (DT) model as the primary analytical instrument. The purpose of study is employing Machine Learning (ML) techniques for the categorization of network traffic related to malware programs. It presents ML approaches utilizing DT, K-Nearest Neighbors (K-NN), Naive Bayes (NB), and Logistic Regression (LR) for predicting network virus traffic. It conducts experiments on the CICAndMal2017 dataset. The technique comprises many essential stages, such as data processing, which includes the raw data that is refined through cleansing and transformation from text format to numerical format. To address the problem of class imbalance among application categories, the oversampling approach was employed to ensure equal representation of all malware types in the dataset, followed by feature engineering. The features were assessed via the Random Forest (RF) model to determine the permissions and behaviors that most significantly impact application classification. This facilitated comprehension of the principal variables behind harmful acts. The findings indicated that the DT model was proficient in prediction and showed superior performance relative to other models. The last classification reports confirmed that the model achieved a good balance between precision and recall in the classification of both secure applications and various malware types. The model's performance was measured by its accuracy, F1-score, recall, and precision; it achieved a score of 99.99% in all measures utilized.

Keywords: Machine Learning, Classification, Android Malware, Decision Tree.

1.Introduction

The widespread adoption of smartphones has led to a significant increase in cybersecurity challenges, particularly regarding malware that targets mobile devices. The vast diversity of the Android system and its open-source nature can be used to explain this phenomenon since it has made Android one of the prime targets of malicious actors[1]. Evidence indicates that out of all malware targeting smartphones, more than 99 percent of them target the Android devices in particular. This has led to the emergence of more than "19 million" malware programs designed to target this system and, hence, acts as a major target for cyber threats in the mobile ecosystem [2]. The increasing spread of malware on the Android platform can be attributed to two major

factors, which include the large number of mobile devices distributed around the world and the comparative permissiveness of application distribution systems. Many malware programs require internet connectivity to communicate with Command and Control (C&C) servers to receive tasks, receive upgrades, or exfiltrate data [3]. This program makes extensive use of the most frequently used and recognized network protocols; it helps bypass the detection provided by traditional firewalls. To compensate for this challenge, security experts may study, record, and investigate the information exchanges between the hosts to come up with a normal performance of the network[3]. This methodical approach will assist in identifying behavioral anomalies, such as irregular bandwidth usage inflows and outflows, DDoS attacks, and various forms of malicious activities [4]. It is possible to look at a lot of network traffic and find suspicious domains to detect malware infections before any signs of them show up. This means that the attackers need to get in touch with the command and control servers, which are on the networks. Early signs of infections can speed up the response and reduce the damage that comes with them [5]. Traffic classification is very important in this analysis because it helps figure out what kinds of applications and programs run on the network. This kind of classification helps network operators and internet service providers keep the network running smoothly [6]. There are three main ways to classify traffic:

- Port-based: These methods use the port numbers to figure out what kind of traffic it is [7].
- Payload-based approaches: These look at the data's contents to figure out what the application is [8].
- Machine learning-based approaches [7]: These are the most common and hopeful ways to go. They fix the problems with the two methods above by assuming that programs send information in standard ways that can be used to tell the difference between traffic. To find these patterns, flow statistics like the average packet size, flow lengths, total packets in a single flow, and use of the TCP protocol are employed [8].

Machine Learning (ML) is one of the branches of Artificial Intelligence (AI) [9]. This approach entails having systems learn through data, identify trends, andmake judgments with little human intervention[10]. The enabling factors have multiplied to the extent of the fast proliferation of the extensive use of this industry, including both the volume and type of data, the ready access to low-cost and powerful computing capabilities, and the efficiency of data storage systems [11]. This development has led to the creation of models that are capable of handling large and complex data sets and providing quick and accurate results; thus, this has created great interest in this field [12]. The paper focuses on the effect of ML methods on the detection of malware on Android devices. We aim to determine how different strategies can be effective in achieving the best accuracy in detecting malware.

This project will be directed toward creating an ML model that will detect and classify hybrid malware on the Android operating system. Hybrid malware refers to the program which combines the characteristics of most types of malicious ,programs such as adware and ,scareware from SMS malware, as well as distinguishing them from legitimate applications [13]. Multi-phase research method was utilized to ensure the quality and effectiveness of the ,model focusing on the application of the Decision Tree (DT) as the major algorithm due to its interpretability and ability to handle complex judgments.

1.1. Research Gaps and Study Contributions

Despite the fact that recent articles have investigated Android malware detection methods based on the application of different ML and DL methods, the majority of them did not investigate the particular features that have the largest impact on the classification outputs. Not much focus has been placed on the relationship between network traffic properties and feature relevance when it comes to accuracy and interpretability. In order to fill this research gap, this research paper offers a clear, feature-based method of classifying hybrid Android malware. The key contributions are:

- Determination of the most valuable network traffic features that improve classification performance.
- Identification of the best machine learning model to detect and classify malware based on four classifiers (DT, NB, KNN, and LR).
- Comparison with other studies conducted using the same dataset to prove the usefulness of the proposed method and provide evidence of improvements.

The research question can be justified by these contributions and offers the basis for future research in interpretable and reproducible malware detection. In terms of the rest of the research paper, the second section presents the results of other related studies that use the same data as the articles referenced. The third section describes the description of the research methods used in this study, including the dataset and the data processing methods, as well as providing four ML models used in the study. The last two sections are dedicated to the results and key findings, and the final section is the summary of the research and future work.

2.Related Works

Android malware has encouraged researchers to develop new detection methods because of the growing concern over Android malware [14]. Special input was provided by Fallah and Bidgoly, who benchmarked other supervised and unsupervised ML algorithms to detect Android malware by analyzing network traffic. The results of the research demonstrate the effectiveness of these approaches to malware detection in the case of their application with appropriate feature selection and a sufficient amount of data [3].

Manh and Cho (2024) present a unique framework that uses an enriched function-call graph alongside GraphSAGE neural networks to select both semantic and structural features of Android applications. Their process is designed to identify malicious trends that are not detected by traditional detection systems. It was found that the model achieved an accuracy of 99.03% and a recall of 99.19% in detection and recall, correspondingly, on contemporary malware samples [15].

In their 2024 paper, " Android traffic malware analysis and detection using ensemble classifier," Mohanraj and Sivasankari. presented a proposed network traffic analysis framework known as STAR that uses a packet parser to derive features of the HTTP and TCP flows. The method uses the SPRINT ensemble classifier to classify Android traffic into five different classes, which include Adware, Banking, SMS, Riskware, and Benign. The reported investigation stated that the SPRINT model had a classification accuracy of 99.5%, and comparative assessments of DNN and RNN models gave 98% and 96%, respectively[16].

In an attempt to overcome the more sophisticated obfuscation mechanisms, Shakya and Dave (2022) showed a dynamic system call-based threat detection model for Android malware. They executed the applications under test in an emulator, and by logging the system-level behavior they obtained, they fed it into training ML models. In their experiments, it was found that a K-Nearest Neighbor (K-NN) model had an F1 score of 0.85 in identifying malware, while a DT model had an F1 score of 0.73 in classifying family [17].

Grace and Sughasiny (2022) created a hybrid model for Android malware detection and classification. They implemented an Aquila optimizer to select the best features from application log files, based on which they classified through a hybrid Long Short-Term Memory (LSTM) and Support Vector Machine (SVM) model. The authors claim a high accuracy of their model at 97 %, compared to other methods currently available, which include individual LSTM, SVM, Random Forest (RF), and Naive Bayes (NB) classifier [12].

3.Methodology

This section outlines the methodology adopted in this study, including the dataset description, preprocessing procedures, applied ML models, and the proposed framework. It also presents the experimental setup that specifies the implementation environment and evaluation metrics to

ensure the reproducibility of results.

3.1. Experimental Setup

All the experiments were performed in a Jupyter Notebook environment, which is packaged with Anaconda. Python (version 3.11.5) was used together with Scikit-learn (1.7.1), Pandas (2.0.3), and NumPy (1.24.3) to implement. The data was split in a way that 80 % of the examples were used as training and the last 20 % for test data. An oversampling method was used to balance the samples in each class because of the imbalance in the classes. All the classifiers, that is, Decision Tree (DT) and Naive Bayes (NB), K-Nearest Neighbors (KNN), and Logistic Regression (LR) have been trained using the same parameter values and random seeds to ensure reproducibility. Accuracy, precision, recall, F1 -score, and the confusion matrix were used to measure performance. All experiments were conducted on a workstation with an Intel 7 core processor, 16GB of RAM, and the Windows 11 operating system.

3.2. Dataset Description

The data used were made available by the Canadian Institute for Cybersecurity, which is known as CICAndMal2017; it is regarded as a realistic and efficient dataset (consisting of more than four thousand samples of malware, including sources like VirusTotal and Contagiodumpst [3]. Moreover, they have collected six thousand harmless applications published in 2015, 2016, and 2017 on the Google Play Store [3]. Secondly, they installed more than 5,500 software applications and 429 malicious malware samples on real Android devices to create a fully realistic environment. Finally, the team connected these Android smartphones to a hotspot computer to capture network traffic files (PCAP) using the TCPDUMP software [3]. The CIC administered network data in three discrete conditions to overcome automated malware that cheats dynamically based on analysis by exploiting time teams:15 minutes after rebooting, immediately after malware installation, and 15 minutes before rebooting. The CIC provided evidence in the form of CSV files [3]. Malware category labeling in CICAndMal2017 has different levels. In the first layer, you can either declare CSV files as benign, malicious, or benign [3]. In the second layer, three specific types of malware exist, including adware, a type of malware that aims at maximizing the number of impressions or clicks on irritating advertising banners [18]. Scareware refers to a type of malware that aims at scaring a user, thus encouraging them to purchase unnecessary programs. SMS malware is malware that sends SMS messages without the user [14]. The purchaser of the malware can use the compromised machines as a premium channel for the SMS services [19]. It should be stressed that these types of malware have the potential to engage in similar malicious actions. For instance, they can exfiltrate users' financial information and transmit it to command-and-control servers [3].

3.3. CICAndMal2017 Pre-processing

This paper uses a multi-step scientific approach to ensure that data is carefully prepared prior to using ML models. The strategy focuses on transforming raw information into a form that can be learned by models, with accuracy as indicated in the subsequent stages.

3.3.1 Data Obtaining and Pre-processing

The data were obtained via the CICAndMal2017.csv file. To maintain consistency at the first stage, a data-cleaning procedure was performed by standardizing all the column names by converting them to a lowercase format and substituting spaces with underscores. The strategy enables access to variables with ease and minimizes program errors that occur due to misnamed columns.

3.3.2 Initial Data Processing

This phase involved addressing the essential concerns contained in the data.

- Categorical encoding: The column of the dependent variable, label, was subjected to a label-encoding method to transform the textual values into numeric codes. As an illustration, the terms Benign and Adware were coded as 0 and 1 respectively.
- Addressing absent data: All columns with object-type data were analyzed, and the missing values were rectified by substituting them with an alternative value (N) prior to encoding.

- Equilibrating the classes: To address the significant disparity between the data classes (the quantity of benign programs surpasses that of harmful programs), the oversampling approach was employed to equilibrate the dataset. This stage sought to balance the number of instances across all classes, hence preventing the model from exhibiting bias towards the most prevalent class. The class column has four categories, with the following sample counts: {Adware = 147443, Scareware = 117083, SMS = 67397, Benign = 23708}. Subsequent to the data balancing procedure, the quantity of samples in each category reached 147443.

3.3.3 Feature Engineering

To improve interpretability, the data were divided into independent variables (features) and a dependent one (label). A Random Forest (RF) classifier was used to measure the importance of features using the Gini impurity measure. The top ten features were identified as the most influential and consequently were further analyzed after ranking features in terms of their significance. Training of classification modelswas then retrained with these features, and the effect on accuracy and computational efficiency was evaluated. This procedure identified the permissions and behavioral qualities that are most suggestive of malicious behavior and minimized redundancy while enhancing model transparency[20].

3.3.4 Data Segmentation

The dataset was partitioned into two subgroups following processing:
Training set: Allocated for model training, it comprises 80% of the data. The test set is intended to assess the model's performance on novel data and comprises 20% of the dataset. The train-test-split function utilized the random-state argument to provide a random and reproducible splitting procedure, hence providing consistent outcomes.

3.3.5 Assessment of performance

A uniform set of measurements was employed to assess the models' performance.

- Confusion matrix: The matrix provided detailed information on correct and incorrect predictions in all classes [25].
- Classification report: This report included accuracy, recall, and F1-score values - metrics important to go through with the strict assessment of model performance. Figure 1 provides the key actions of the methodology used[25]. **Figure 1** illustrates the primary steps of the employed methodology.

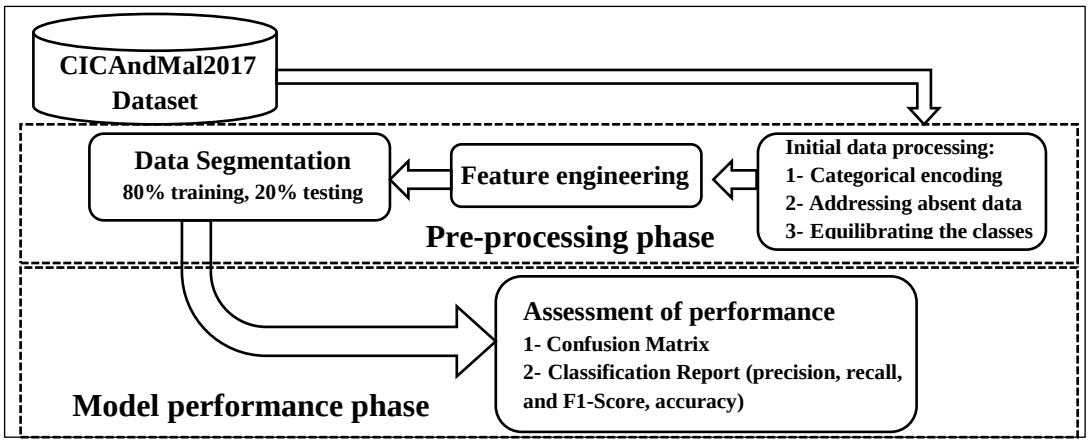

Figure 1. Architecture of the methodology

The classification report utilized a set of four measures, which improve the credibility of the evaluation process and are accuracy in Equation (1) [21], precision in Equation (2) [22], recall in Equation (3) [23], and the F1-score in Equation (4) [10]. Precision, in this case, means the rate of positive predictions being correct, and recall is often defined as sensitivity, hit rate, or the true positive rate [24]. The F1-score is the harmonic mean of recall and precision [25]. TP refers to the number of cases that are correctly classified as positive [26]; FP refers to the number of cases that

are falsely classified as positive [27]; FN refers to the number of cases that are falsely classified as negative [28]; and TN refers to the number of cases that are correctly classified as negative [29].

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad \textbf{(1)}$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad \textbf{(2)}$$

$$Recall = \frac{TP}{TP+FN} \quad \textbf{(3)}$$

$$\text{F1} - \text{score} = \frac{2\ \text{Precision} * \text{Recall}}{\text{Precision} * \text{Recall}} \quad \textbf{(4)}$$

3.4. Models

The current study focuses on complementing Android malware detection through the enhancement of the preprocessing and feature treatment procedures. Traditional ML classifiers, e.g; DT, NB, K-NN, and ,LR were used, and performance improvements were achieved through techniques like data balancing, feature selection, and normalization. This study does not come up with a hybrid model; instead, it highlights the role of preprocessing and feature relevancy in detection accuracy and interpretability.

3.4.1. Decision Tree (DT)

This model employs input variables to generate decision rules that forecast a response variable. The decision rules are represented as nodes that partition the feature space into areas referred to as sub nodes. Each sub-node is divided successively until a specific requirement is met. Each endpoint of these structures is designated as a leaf and assigned a fixed score value (C), which signifies the mean of the response variable values at that endpoint. The formulation for a data set comprising (n) observations and (m) input variables is represented in Equation (5) [8].

$$f(x)= \{C\_q(x)\ (q: R\text{^}m \rightarrow 1,2,\ldots,t,\ C \in R\text{^}m)\} \quad \textbf{(5)}$$

In this context, q(x) delineates decision rules inside a tree that assign data samples to certain leaf indices, (t) denotes the total count of leaves in the tree, and C-(q(x)) signifies the score weights attributed to the tree's leaves [8].

3.4.2 Naive Bayes (NB)

It is a variant of the probabilistic classification technique derived from Bayes' Rule. The technique presumes that the properties, or features, are independent of one another conditional on the class. The NB classifier is a highly scalable algorithm for high-dimensional datasets and operates in linear time for making predictions. Consequently, it is an appropriate method for text classification tasks such as spam detection, information retrieval, and text categorization [13].

3.4.3 K-Nearest Neighbors (K-NN)

The K-NN algorithm is a well-liked nonparametric method for regression and classification. Financial modeling, picture interpolation, and visual identification are just a few of the many research disciplines that have found success with the K-NN approach [30].

3.4.4 Logistic Regression (LR)

It is a classification method that is based on and adapts to data using the logit function (or logistic curve). Multinomial logistic regression is designed to predict the outcomes of a multi-class problem based on a specified collection of features, which may include real-valued, binary-valued, or categorical-valued variables. It is proficient at managing a substantial array of characteristics [13].

4.Results and Discussions

The comprehensive study of the data revealed various outcomes regarding feature significance and model efficacy.

4.1 Feature Importance

The examination of feature significance utilizing the RF model indicated that some permissions and behaviors provide much greater predictive capacity than others in the classification of applications. The paramount features, listed in descending order of significance, were:

1.Timestamp.

2.Fwd_avg_bytes/bulk.
3.Down/up-ratio.
4.Unnameddi_o.
5.Source-ip.
The results suggest that timestamp, fwd-avg-bytes/bulk, and down/up-ratio are key markers of the application's character, determining its maliciousness. **Figure 2** shows the top feature importance.

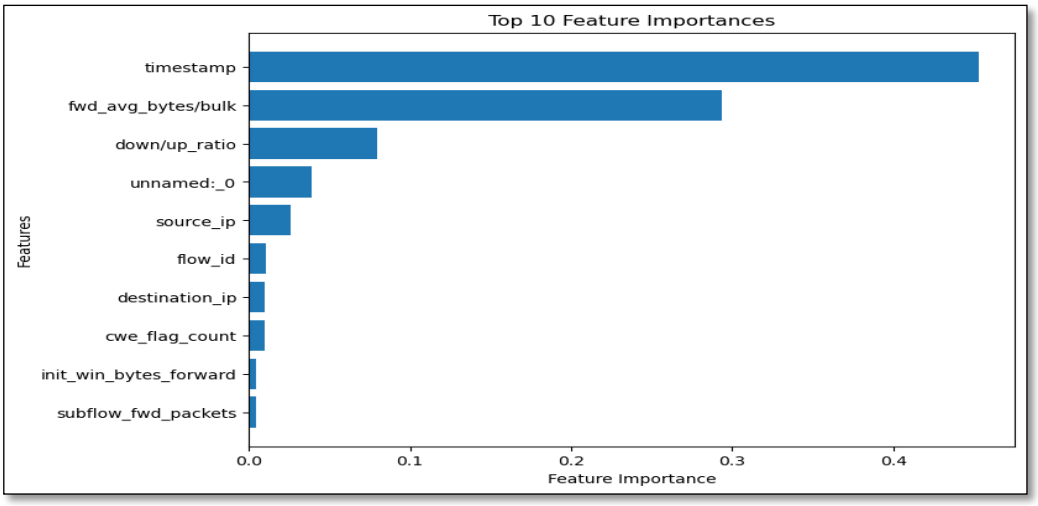

Figure 2. Top 10 Feature Importance

4.2 Models Performances

The efficacy of four categorization models was evaluated using the test set. The confusion matrices and classification reports indicated a distinct variance in performance among the models. The confusion matrix is a table that shows how well a classification model works. Each row displays the actual classes of the data, while each column displays the predicted classes from the model. The numbers in the cells show how many instances fit into each combination of actual and predicted classes [5]. **Figure 3a** shows the confusion matrix for DT. There are four classes in this matrix: (a) Adware, (b) Scareware, (c) SMS malware, and (d) Benign, which means that the applications are not harmful. Let's go through the matrix one row at a time to see how well the model works for each class. The model correctly found 4,770 cases of Adware (Actual). It accurately guessed that (0) instances of Adware were Scareware, SMS malware, or Benign. The model does almost everything right here. Actual Scareware: The model accurately predicted 29,541 cases of Scareware. It did not make any wrong guesses for this class. Again, this is a very excellent result. SMS malware (Actual): The model correctly identified 23,494 cases of SMS malware. However, it got four predictions wrong and incorrectly labeled four real SMS malware cases as Android adware. This percentage is a very small number, which means that the model is still doing very well on this class. Benign (Actual): The model accurately detected 13,317 occurrences of benign applications. It did not mistakenly label any benign applications as harmful; this is an important part of a beneficial security model. The confusion matrix for NB is depicted in **Figure 3b**. The NB classifier faced significant challenges in correctly identifying malware categories. For adware, 4,274 applications were correctly classified, while 44 were misclassified as scareware, 152 as SMS malware, and 300 as benign. Mislabeling 300 adware applications as benign indicates missed threats. For scareware, 21,836 applications were correctly identified, with 2,039 misclassified as adware and 5,013 as benign, highlighting a substantial risk if scareware is treated as harmless. For SMS malware, 17,644 applications were correctly predicted, while 1,775 were labeled as adware and 3,607 as benign. Regarding benign applications, the NB model correctly classified 4,796 samples, while 2,455 were misclassified as scareware and 6,066 as SMS malware. Overall, the high misclassification

rates demonstrate that the NB struggles with correlated features and complex nonlinear interactions in malware datasets, resulting in numerous false positives and false negatives. Overall review: Compared to a strong classification model, the NB model does not perform very well. There are too many wrong predictions, such as when real malicious applications are wrongly classified as benign and benign applications are wrongly classified as malicious. This test shows that the system is not very reliable. In addition, the model is not suitable for a critical job like threat detection because it has a low accuracy rate and a high rate of both false positives and false negatives. The best confusion matrix would have many high numbers along the main diagonal (correct predictions) and many zeros or very low numbers everywhere else. This matrix shows that there are important values off the diagonal, especially for the benign class.

Figure 3c is a confusion matrix for a K-NN model. The model only correctly identified 450 adware applications as actual adware. However, it made many mistakes, wrongly classifying 2,444 as scareware, 1,566 as SMS malware, and 310 as benign. Many mistakes occurred, particularly in misclassifying items into incorrect categories, such as incorrectly labeling them as benign. Scareware (Actual): The model found 21,752 scareware applications. This is a good number, but many of them were wrongly labeled: 1,146 as adware, 5,133 as SMS malware, and 1,510 as benign. It is a serious mistake to misclassify scareware as benign. SMS malware (Actual): Only 12,201 SMS malware applications were found to be real. The model got many of them wrong: 946 were classified as adware, 8,635 as scareware, and 1,716 as benign. A lot of SMS malware is wrongly labeled as scareware. Benign (Actual): The model only correctly identified 2,342 applications as benign. It put a lot of them in the wrong category: 418 as adware, 5,922 as scareware, and 4,635 as SMS malware. The high rate of false positives, which flags benign applications as malicious, could significantly negatively impact the user experience. Final thoughts: The K-NN model does not work well for any of the classes. The model is not reliable and is likely to misclassify items because there are so many off-diagonal values. The model struggles to distinguish between various types of malware, and crucially, it frequently misclassifies both malicious and benign applications. This level of performance is not satisfactory enough for a security-sensitive application like malware detection.

The confusion matrix for LR is depicted in **Figure 3d**. The numbers on the main diagonal (from top-left to bottom-right) show the correct predictions, and the numbers off the diagonal show the incorrect ones. The model only correctly identified 12 adware applications as adware. However, it got them all wrong, indicating that 3,620 were scareware, 1,041 were SMS malware, and 97 were benign. This classification significantly harms the model's performance. Scareware (Actual): The model correctly found 25,779 scareware applications. Even though this is the most accurate prediction count, it still makes many mistakes, calling 21 of them adware, 3,288 of them SMS malware, and 453 of them benign. It is very dangerous for security to mistake harmful applications for safe ones. SMS malware (Actual): There were only 7,361 SMS malware applications that were correctly predicted. A lot of them were put in the wrong category: 30 were put in the adware category, 15,661 were put in the scareware category, and 446 were put in the benign category. A big problem is that many things are wrongly labeled as scareware. Benign (Actual): The model only correctly identified 273 applications as benign. It got a lot of them wrong, calling 23 of them adware, 9,854 of them scareware, and 3,167 of them SMS malware. This study shows that there are a lot of false positives. This means that harmless applications are wrongly marked as harmful; this makes the user experience very bad. Final thoughts: the LR model does a terrible job of classifying mobile applications. The model has a challenging time correctly identifying applications in almost every category. Because there are so many misclassifications, especially when benign applications are labeled as malicious and malicious applications are labeled as benign, it is not useful for any real-world application, especially the one related to security. The best model would have high values along the diagonal and very low values everywhere else. This is not the case here.

Upon examining the performance of four distinct ML models, DT, NB, K-NN, and LR were

examined for their performance in classifying Android applications, revealing that their methodologies varied significantly. The decision tree model was characterized by exceptional predictive accuracy, making very few mistakes, and was consistent throughout the four categories on whichit was tested: adware, scareware, SMS-based malware, and benign applications. The rest of the three classifiers, namely, NB, K-NN, and LR, exhibited significantly worse performance, which can be explained by the fact that they were limited in their theory and were especially susceptible to the inherent properties of the dataset. The Naive Bayes algorithm, which is based on the assumption of feature independence, is unable to reflect the complexity of interdependencies that define malware data, with permissions, API calls, and behavioral attributes being strongly correlated with one another. Thus, this model produced high rates of false positives - benign applications that were incorrectly detected as malicious and false negatives malicious applications that were falsely identified as benign. K-NN, because of distance measures and feature scaling, did not perform well in the high-dimensional feature space of the experiment, where correlated features undermined its discriminative power, and it frequently misclassified benign and malicious samples. Logistic regression, which is a linear model, was only suitable for those datasets that were linearly separable, though Android malware features exhibit nonlinear relationships that cannot be modeled by a simple decision boundary, resulting in inaccurately classified data, especially between the adware and benign classes. Together, the capacity of the decision tree to represent nonlinear relationships and the ability to deal with feature dependencies put the decision tree at a decisive edge over the alternative models, making it a more accurate as well as a naturally interpretable approach to protecting applications in security-relevant situations, such as the identification of Android malware.

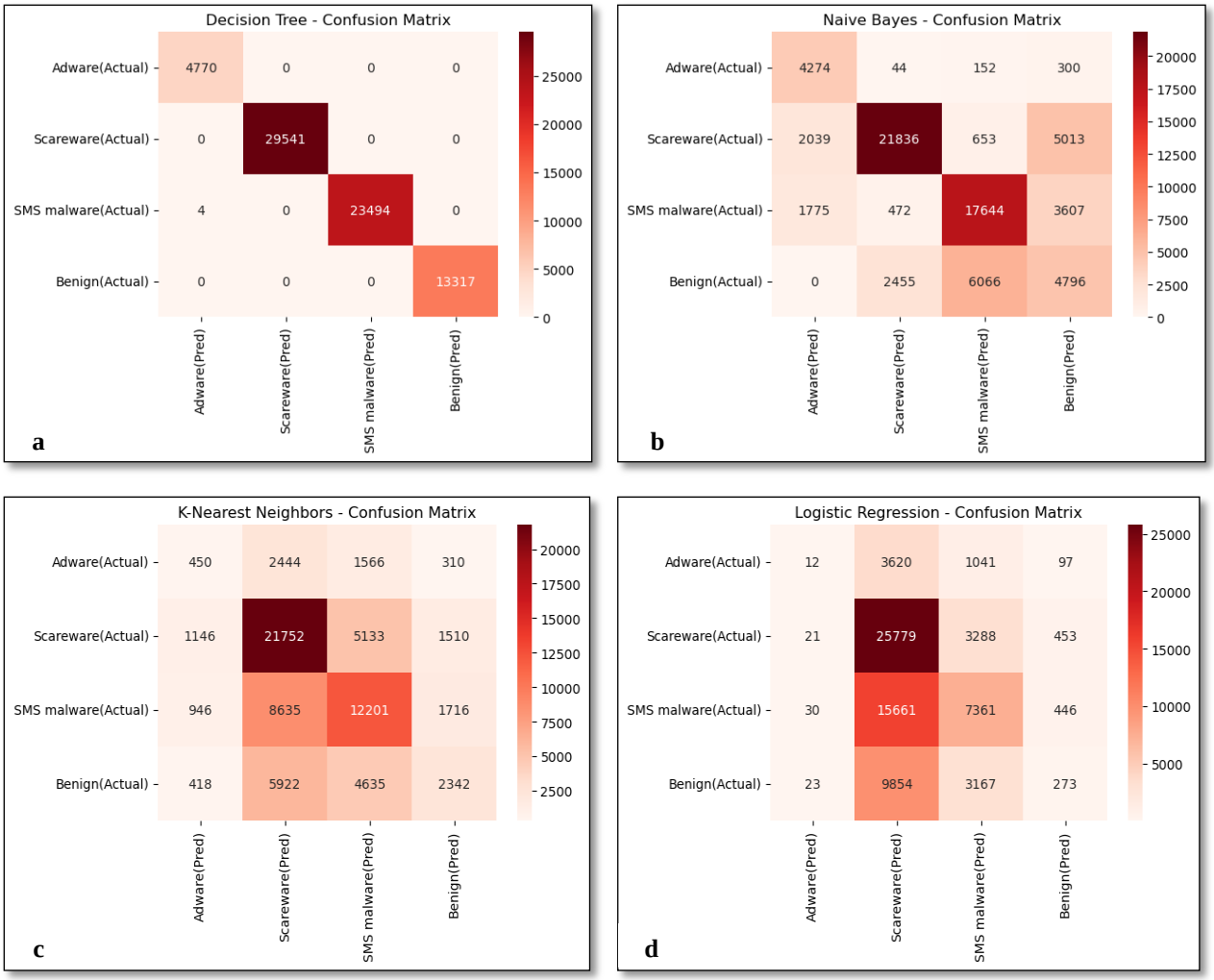

Figure 3. Confusion Metrix for models: (a) DT, (b) NB, (c) K-NN, (d) LR

Table 1 presents the classification report results, which corroborate the findings derived from the confusion matrix.

Table 1. Classification report for models

Models	Metrics			
	Precision (%)	**Recall(%)**	**F1-score (%)**	**Accuracy (%)**
DT	99.99	99.99	99.99	99.99
NB	53	90	66	68
K-NN	15	9	12	52
LR	0.14	0.00	0.00	47

4.3. Comparative Analysis

The proposed model's performance was methodically compared against established approaches from prior research, with efficacy and accuracy as the major metrics of measurement. **Table 2** indicates that prior investigations employing various approaches demonstrated significant results. In particular, Refs [31] employed the DL, which attained accuracies of 0.84% and 0.98%, respectively. Refs [1, 3, 17, 12] also documented accuracies of 0.97%, 0.96%, 0.90%, and 0.85%, respectively, utilizing an ensemble ML model. The recommended model (DT) achieved 99.99% in all measures used, which proves the accuracy of the processing procedures and their importance in the work of the classification model. The excellent performance provided by the DT model may be explained by a number of reasons. In contrast to NB, K-NN, and LR, DT is able to learn nonlinear relationships among features (without the need for transformations) and model interactions among permissions, API calls, and behavioral features of Android applications. Furthermore, DT can process high-dimensional data efficiently and without scaling features, and it offers a clear hierarchical structure that can help to understand the courses of decision-making. These benefits will allow DT to achieve greater accuracy and reduce misclassifications for all types of malware, making it specifically relevant to security-sensitive applications like malware detection.

Table 2. Comparative analysis with relevant studies and the suggested model

Ref	Year	Model	Metric
Fallah and Bidgoly [3]	2019	SVM RF NB KNN	F1-score 80% F1-score 66.49% F1-score 80% F1-score 54.92%
Elayan and Mustafa [31]	2021	GRU	Accuracy 98%
Skaka and Dave [17]	2022	Ensemble ML	Accuracy 90.4
Gracea and Sughasiny[12]	2022	Ensemble ML	Accuracy 85%
The proposed	-	DT	Accuracy 99.99% F1-score 99.99% Recall 99.99% Precision 99.99%

5.Conclusions

The network traffic malware detection provides a solid methodology to identify malicious samples based on behavioral signatures, hence putting malicious actions at bay before they cause damage. This study has made a comparative evaluation of traditional machine-learning classifiers, including DT, NB, K-NN, and LR in the multi-class Android malware classification. The decision-tree model proved to be the most successful model, which can be explained by the fact that it captures non-linear interactions between feature variables and application behavior. A feature-importance analysis revealed that permissions related to internet access, telephony status, and SMS messaging are key indicators of malevolence, and as a consequence, this guarantees that model adjudications are readable and understandable. On the other hand, the comparatively poor results of logistic regression and naive Bayes underscore the weakness of models based on simplified assumptions. All these results point to the fact that judicious preprocessing and feature

engineering significantly increase the efficacy of a classifier. To conclude, the decision-tree paradigm can provide a reliable and understandable approach to Android malware detection and classification, which holds the potential to be practiced in real operational malware defense systems.

Future research may pursue several avenues to augment this study based on the existing findings. First, employing sophisticated methodologies: deep learning (DL) architectures, like Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), may be utilized to examine sequential data streams, such as API requests or data flows. Second, incorporating behavioral and dynamic attributes: the dataset may be augmented to encompass dynamic characteristics gathered throughout the application's functionality (such as energy usage, processor velocity, and network engagement). This would offer a more thorough perspective on the application's activity. Third, broadening the categorization parameters: the study may be extended to encompass various malware types or forecast further application attributes, thereby enhancing the precision and efficacy of security systems.

Conflict of Interest

The authors declare that they have no conflicts of interest.

Funding

No funding.

References

1. Smmarwar SK, Gupta GP, Kumar S. Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: A comprehensive review. Telemat. Inform. Rep. 2024;14:100130. https://doi.org/10.1016/j.teler.2024.100130 .
2. Pathak A, Barman U, Shanta T. Machine learning approach to detect android malware using feature-selection based on feature importance score. J. Eng. Res. 2025;13(2):712–20. https://doi.org/10.1016/j.jer.2024.04.008 .
3. Fallah S, Bidgoly AJ. Benchmarking Machine Learning Algorithms for android malware detection. Jordanian. J. Comput. Inf. Technol. 2019;05(03):216–30.
4. Mbunge E, Muchemwa B, Batani J, Mbuyisa N. A review of deep learning models to detect malware in Android applications. Cyber secur. Appl. 2023;1:100014. https://doi.org/10.1016/j.csa.2023.100014 .
5. Smmarwar SK, Gupta GP, Kumar S. Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: A comprehensive review. Telemat. Inform. Rep. 2024;14:100130. https://doi.org/10.1016/j.teler.2024.100130 .
6. Aamir M, Iqbal MW, Nosheen M, Ashraf MU, Shaf A, Almarhabi KA, et al. AMDDLmodel: Android smartphones malware detection using deep learning model. PLoS One. 2024;19:1–16. http://dx.doi.org/10.1371/journal.pone.0296722 .
7. Mhalhal NK, Behadil, SF. Mobility Prediction Based on Deep Learning Approach Using GPS Phone Data. Ibn Al-Haitham. J. Pure. Appl. Sci. 2024;37(4):423–38, https://doi.org/10.30526/37.4.3916 .
8. Habeeb MA, Khaleel YL. Enhanced Android Malware Detection through Artificial Neural Networks Technique. Mesopotamian. J. Cybersecur. 2025;5(1):62–77. https://doi.org/10.58496/MJCS/2025/005.
9. Alhogail A, Alharbi RA. Effective ML-Based Android Malware Detection and Categorization. Electron. 2025;14(8), https://doi.org/10.3390/electronics14081486 .
10. Ali SF, Abdulrazzaq MR, Gaata MT. Learning Techniques-Based Malware Detection: A Comprehensive Review. Mesopotamian. J. Cybersecur. 2025;5(1):273–300, https://doi.org/10.58496/MJCS/2025/018 .
11. Shatnawi AS, Jaradat A, Yaseen TB, Taqieddin E, Al-Ayyoub M, Mustafa D. An Android Malware Detection Leveraging Machine Learning. Wirel. Commun. Mob. Comput. 2022;2022, https://doi.org/10.1155/2022/1830201 .

12. Gracea M, Sughasiny M. Malware detection for Android application using Aquila optimizer and Hybrid LSTM-SVM classifier. EAI Endorsed. Trans. Scalable. Inf. Syst. 2023;10(1):1–11, https://doi.org/10.1007/978-3-319-59162-9_20 .
13. Islam R, Islam M, Saha S, Jamal M, Masud A. Internet of Things and Cyber-Physical Systems Android malware classification using optimum feature selection and ensemble machine learning. IoT. CPS. 2023;3:100–11. https://doi.org/10.1016/j.iotcps.2023.03.001 .
14. Mbunge E, Muchemwa B, Batani J, Mbuyisa N. A review of deep learning models to detect malware in Android applications. Cybersecur. Appl. 2023;1:100014. https://doi.org/10.1016/j.csa.2023.100014 .
15. Minh MV, Xuan C Do. A Novel Approach for Android Malware Detection Based on Intelligent Computing. Comput. Mater. Contin. 2024;81(3):4371–96. https://www.techscience.com/cmc/v81n3/59055 .
16. Mohanraj A, Sivasankari K. Android traffic malware analysis and detection using ensemble classifier. Ain. Shams. Eng. J. 2024 Dec;15(12):103134. https://doi.org/10.1016/j.asej.2024.103134 .
17. Shakya S, Dave M. Analysis, Detection, and Classification of Android Malware using System Calls. 2022; https://arxiv.org/abs/2208.06130 .
18. Ksibi A, Zakariah M, Almuqren L, Alluhaidan AS. Deep Convolution Neural Networks for Image-Based Android Malware Classification. Comput. Mater. Contin. 2025;82(3):4093–116, https://doi.org/10.32604/cmc.2025.059615 .
19. Gómez A, Muñoz A. Deep Learning-Based Attack Detection and Classification in Android Devices. Electron. 2023;12(15), https://doi.org/10.3390/electronics12153253 .
20. Abuthawabeh MKA, Mahmoud KW. Android malware detection and categorization based on conversation-level network traffic features. Proc - 2019 Int. Arab. Conf. Inf. Technol. ACIT 2019. 2019:42–7, https://doi.org/10.1109/ACIT47987.2019.8991114 .
21. Giannakas F, Kouliaridis V, Kambourakis G. A Closer Look at Machine Learning Effectiveness in Android Malware Detection. Inf. 2023;14(1), https://doi.org/10.3390/info14010002.
22. Jo J,Cho J, Moon J. A Malware Detection and Extraction Method for the Related Information Using the ViT Attention Mechanism on Android Operating System. Appl. Sci. 2023;13(11), https://doi.org/10.3390/app13116839.
23. Atacak İ,Kılıç K, Doğru İA. Android malware detection using hybrid ANFIS architecture with low computational cost convolutional layers. Peer. J. Comput. Sci. 2022;8.
24. Aboshady D, Ghannam N, Elsayed E, Diab L. The Malware Detection Approach in the Design of Mobile Applications. Symmetry (Basel). 2022;14(5):839. https://www.mdpi.com/2073-8994/14/5/839.
25. Alkahtani H, Aldhyani THH. Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices. Sensors. 2022;22(6):1–26, https://doi.org/10.3390/s22062268.
26. Kavalcı Yılmaz E, Bakır R. Advanced Android Malware Detection: Merging Deep Learning and XGBoost Techniques. Bilişim. Teknol. Derg. 2025;18(1):45–61, https://doi.org/10.17671/gazibtd.1553548 .
27. Odat E, Yaseen QM. A Novel Machine Learning Approach for Android Malware Detection Based on the Co-Existence of Features. IEEE Access. 2023;11:15471–84, https://doi.org/10.1109/ACCESS.2023.3244656 .
28. Alabrah AA. Novel Neural Network Architecture Using Automated Correlated Feature Layer to Detect Android Malware Applications. Mathematics. 2023;11(20):1–14, https://doi.org/10.3390/math11204242 .
29. Vasani V, Bairwa AK, Joshi S, Pljonkin A, Kaur M, Amoon M. Comprehensive Analysis of Advanced Techniques and Vital Tools for Detecting Malware Intrusion. Electron. 2023;12(20):1–30, https://doi.org/10.3390/electronics12204299 .
30. Sudesh K, Shersingh , Siddhant , Karan V. Malware Classification Using Machine Learning Models. Procedia. Comput. Sci. 2024;235:1419–28. https://doi.org/10.1016/j.procs.2024.04.133 .
31. Elayan ON, Mustafa AM. Android malware detection using deep learning. Procedia Comput Sci. 2021;184(2019):847–52. https://doi.org/10.1016/j.procs.2021.03.106.