ISSN: 2182-2069 / E-ISSN: 2182-2077

Machine Learning-Based Detection and Mitigation of Malware in Smartphone Applications Amidst Increasing Mobile Cybersecurity Threats

Dr. Mawahib Sharafeldin Adam Boush[1*], Rejna Azeez Nazeema[2],
Muhammad Humza Farooq Siddiqui[3], Anjali Appukuttan[4], Sabna Machinchery Ali[5],
Najla Elhaj Babiker[6], and Ehab Tarek Desouky Ibrahim Elaswed[7]

[1*]Assistant Professor, Department of Computer Science, College of Engineering and Computer Science, Jazan University, Jazan, Saudi Arabia. mboush@jazanu.edu.sa, https://orcid.org/0009-0002-2695-1843

[2]Department of Computer Science, College of Engineering and Computer Science, Jazan University, Jazan, Saudi Arabia. razeez@jazanu.edu.sa, https://orcid.org/0009-0007-9320-4537

[3]Deanship of Human Resources and Technology, Jazan University, Jazan, Saudi Arabia. mfarooq@jazanu.edu.sa, https://orcid.org/0000-0002-6586-4219

[4]Department of Computer Science, College of Engineering and Computer Science, Jazan University, Jazan, Saudi Arabia. anarayanan@jazanu.edu.sa, https://orcid.org/0009-0009-7288-0661

[5]Department of Computer Science, College of Engineering and Computer Science, Jazan University, Jazan, Saudi Arabia. saali2@jazanu.edu.sa, https://orcid.org/0009-0001-4581-525X

[6]Department of Computer Science, College of Engineering and Computer Science, Jazan University, Jazan, Saudi Arabia. nbabiker@jazanu.edu.sa, https://orcid.org/0009-0004-5891-6430

[7]Lecturer, Faculty of Specific Education, Educational Technology and Computer Science, Zagazig University, Zagazig, Sharqia, Egypt; Assistant Professor, Educational Technology, Deanship of Human Resources and Technology, University Jazan, Jizan, Saudi Arabia. ganaehab@gmail.com https://orcid.org/0009-0005-4537-3083

Received: September 02, 2025; Revised: October 09, 2025; Accepted: December 02, 2025; Published: February 27, 2026

Abstract

The ubiquitous development of mobile devices, especially Android-based operating system, has provided a good environment of malicious software (malware) to steal user data and system integrity. Android is open-source in nature and has a large size of applications, which makes good, timely detection of malware a significant security problem. To overcome this, the paper offers a new and evidence-based model of efficient malware detection based on various machine learning (ML) models. It uses the methodology that targets a static analysis, which employs a set of discriminatory features, including sought dangerous permissions and suspicious API calls as major elements of a comprehensive set of discriminatory features, extracted out of Android Application Packages (APKs). The output of four popular ML classifiers, namely, Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Naive Bayes (NB) was tested on a big, real-life dataset. The outcome of the experiment proves that the Random Forest classifier performed greatly

*Corresponding author: Assistant Professor, Department of Computer Science, College of Engineering and Computer Science, Jazan University, Jazan, Saudi Arabia.

better than the other classifiers, giving an accuracy of 98.5 and an F1-score of 98.3. This high performance justifies the performance of the chosen static features to draw a line between the benign and malicious applications. The model obtained provides a strong, highly precise, and computationally efficient solution that can be incorporated into real-time security systems to enhance the defensive capability of the Android platform against ever-changing threats.

Keywords: Android Platform, Malware Detection, Machine Learning (ML), Static Analysis, Cybersecurity, Random Forest (RF), Feature Extraction.

1 Introduction

The advent and use of mobile computing devices at a high rate and speed have drastically changed the way business is conducted, communication, and lives worldwide (Alamleh et al., 2023). With the Android platform being the top mobile operating system by market share with the support of open-source platform, and huge application market, it has become the major target of malicious actors. This omnipresence has unwittingly provided a large attack surface thus increasing the susceptibility of the system to malicious software. Malicious programs (malware) tend to be embedded in seemingly harmless programs and interfere with user information, financial accounts and the integrity of a device (Ghoson et al., 2025; Mokoena & Nilsson, 2023). Conventional security measures that are majorly signature-based are no longer effective in countering contemporary, polymorphic and obfuscated threats. This has resulted in an acute and severe necessity to come up with clever, dynamic, and effective detection mechanisms that can be able to pre-empt and curb these emerging security threats (Thakur & Biswas, 2022). The main difficulty of Android malware detection is the launched arms race between malware developers and security experts. Variations of malware often use sophisticated methods, including code obfuscation, repackaging, and dynamic loading of the payload, to avoid the use of the tools of the statical signature matching (Al Tobi et al., 2022). Conventional methods are not intelligent enough to generalize instances of detection, which causes them to yield a high level of false negatives in the event that they meet a zero-day threat or a threat that they have never seen before. Thus, the nature of the central issue, which this study will attempt to resolve, is that there is no computationally efficient and highly accurate detection model capable of detecting a malicious and benign Android application (APK) based on the features of their core structure to overcome the weaknesses of traditional security solutions (Damayunita et al., 2022; Zhang et al., 2025). This research has specific aims that are well stated to solve the issue of quality and effective Android malware detection. The former aims at determining and mining a small but informative set of discriminatory features of Android applications using solely the power of static analysis, i.e. the requested dangerous permissions, the suspicious API calls and, therefore, maintaining computational efficiency. After assessing the features, the second goal then is to comparatively analyze and compare the performance of the four popular Machine Learning classifiers in order to identify as random forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Naive Bayes (NB) using a large and real-world data (Srinivasan et al., 2022; Vij & Prashant, 2024) Lastly, the third goal is to find out the strongest and most precise ML algorithm that can enhance the best classification results in the ability to differentiate between harmless and harmful applications (Ning et al., 2021; Puspitasari et al., 2023).

Key Contributions

- The launch of a new, data-intensive model of the high-accuracy Android malware detection founded solely on the case analysis.

- Strict evidence that the Random Forest classifier is much more effective than other models that have been benchmarked, with a detection rate of 98.5% and F1-score of 98.3%. In studies on adopting mobile access control methods, it has already been established that the efficiency of a set of static features extracted efficiently (permissions and API calls) can yield a still highly accurate, robust, and computationally sparse model.
- Proposal of a perfect application to real-time implementation in resource-restrained mobile security settings.

The rest of this paper will be structured to the following areas: Section 2 will entail a review on related work on mobile malware detection and lunges on signature-based, static, and dynamic analysis methods that employ diverse Machine Learning and Deep Learning methods. Section 3 elaborates the proposed methodology which entails the data collection process, the feature extraction process and the particular ML models to be used as the benchmarking models. Section 4 is the experimental setup, methods of the performance measures, and discussion of the findings of the various classifiers. Lastly, Section 5 is the conclusion of the paper where the main findings will be summarized and the path of future research will be described.

2 Literature Survey

With the emergence of the Android platform, the automated malware detection methods have started to be extensively researched, with the focus moving not only to the old signature-based methods but also to the new methods of using Machine Learning (ML) and Deep Learning (DL) (Rodrigo et al., 2021; Kim et al., 2022). The early detection systems were based on a database of signatures and the matching of the cryptographic hash of the known malicious applications (APKs) against the database. Nevertheless, they are easily bypassed by the contemporary malware based on polymorphism and obfuscation, thus the need to employ behavior-based and structural-based analysis (Srinivasulureddy et al., 2022).

Modern studies generally divide the field of ML-based detection into three major directions, including, but not limited to, static, dynamic, and hybrid analysis (Kotenko et al., 2022; Jagadeeswaran et al., 2022). The concept of analysis is known as static analysis whereby an application is analyzed without being run. Features are usually represented on the manifest file, Smail code disassembly and resource files. Some of the most common static features are requested permissions, sensitive API calls, intent filters, and hardware components utilized (Archibong et al., 2024; Rahali & Akhloufi, 2021). A systematic review by Senanayake et al. concluded that when compared to other methods, static analysis is appreciated in importance due to the speed and the lack of computational burden, and can be effectively used in pre-installation checking, but in cases where obfuscation is heavily applied or dynamically loaded code, the approach fails to perform well. Dynamic analysis refers to the monitoring of an application that is being run in a controlled environment (sandbox). This technique is very effective in capturing advanced malicious actions that do not manifest themselves in any way until execution like command-and-control communication, dropping of hidden files or loading of dynamic code (Deshmukh & Menon, 2025).

The features that are obtained are system calls, network traffic, battery consumption, and memory usage. Although its accuracy is extremely high when used against obfuscated malware, dynamic analysis has a high runtime cost and the code coverage issue, that is, an analyst has to ensure that all malicious code paths have been followed. The recent research has been aimed at optimizing the ML and DL algorithms in classification (Purnama et al., 2024). It was examined with different ML models for detection and categorization, and supports the notion that ensemble models, such as Random Forest (RF) are very effective since they can represent high-dimensional data and feature interactions. Moreover, the review of the ML-based detection was conducted through benchmarking and gap identification, which highlights the necessity of conducting stringent comparative studies nowadays. The AMDDL model has also demonstrated encouraging results, which typically require APK features to be input to a deep learning model as images or consecutive data, at the expense of higher complexity and training time.

The table below provides an overview of the main works in the field of ML-based Android malware detection, including the feature extraction approach and classifier employed, and the major performance measures of the research.

Table 1: Key Studies

Study	**Feature Analysis Type**	**Technique/Classifier**	**Key Performance Result**
Current Study	Static (Permissions, API Calls)	Random Forest (RF)	98.5% Accuracy
2021	Systematic Review	Taxonomy/Gap Analysis	N/A (Review)
2023	Review / Benchmarking	Taxonomy & Evaluation	N/A (Review)
2021	Static / Deep Learning	Deep Learning Model	High Accuracy (Specifics vary)
2020	Hybrid (Static + Dynamic)	Deep Learning	Focus on Performance-Sensitive Detection
2025	Static	Random Forest, Naive Bayes	Tested multiple ML classifiers

As shown in Table 1, the strongest and most used method to counter the modern Android malware is Machine Learning, especially on the use of the permissions and API calls as static features. The opinion of review papers is in support of strict benchmarking of algorithms and finding resource-efficient detection models. Whereas Deep Learning (DL) models and Hybrid analysis (static and dynamic features) tend to push the limits of precision, they come at a great price in terms of computation, so they cannot be used on resources-constrained mobile devices or with high app store throughput scan. The present research paper is strategically focused to fill the gap and concentrate on the highly efficient method of static analysis only and benchmarking the classic ML methods strictly. The main conclusion that the Random Forest classifier can reach a 98.5 percent detection rate with a smaller set of discriminative features selected mostly manually is an essential factor. This performance is competitive and in certain cases exceeds the performance on more complicated DL and hybrid models. This confirms that the proposed framework is an ideal solution as it has the necessary high detection rate and the computing power required to implement mobile security in real-time, and thus meets the requirement to have strong, lightweight detection systems, as advanced by other researchers.

3 Methodology

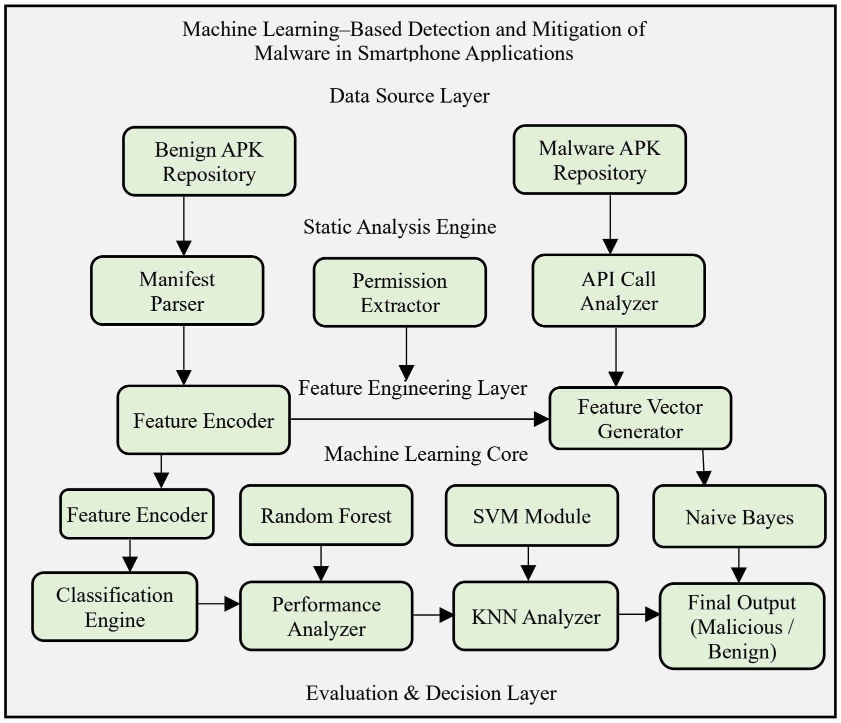

Figure 1: Methodology flow architecture diagram

Figure 1 shows the overall system architecture of the suggested Android malware detection system based on machine-learning. The architecture is layered with some layers having a specific role and an output processed by them is relayed to the next layer. The Data Source Layer has got two major repositories one benign and the other malicious APKs that provide balanced and diverse data set. These APKs are sent to the Static Analysis Engine where inner structural information is obtained without execution of the applications. There are three main modules in this engine: Manifest Parser which finds the metadata in the AndroidManifest.xml file; Permission Extractor which identifies the dangerous permissions requested by the application; and API Call Analyzer which decompiles the APK code to identify suspicious API calls which could include reflection, dynamic code loading and SMS/network operations. The results of these elements are delivered to the Feature Engineering Layer where they are transformed into machine understandable numerical formats. The Feature Encoder converts the permission extracted and API calls into binary indicators and the Feature Vector Generator joins them together to create a single static feature vector of each APK. This data is fed into the Machine Learning Core where several different classifiers are trained and tested on the same feature sets to determine which model performs the best. Random Forest, Support Vector machine, K-Nearest Neighbors, and Naive Bayes are some of the classifiers available in the machine learning core. The measured structured feature vectors are interpreted using individual learning mechanisms per classifier, and a malware or benign prediction is obtained. Lastly, the Evaluation & Decision Layer processes these predictions and the modules of the Classification Engine, KNN Analyzer and Final Output unit combine the decisions of the classifier and produce the final malware detection outcome. In general, the flow of APK data through multiple analysis phases in Figure 1 indicates a well-defined flow of how raw application files are converted into organized features that can be used to promote intelligent malware classification based on machine-learning. The features extracted using the static analysis are of two main types, dangerous permissions in AndroidManifest.xml and suspicious API calls in the decompiled code. The total application data set is represented as Equation (1) that is

$$\mathcal{D} = \{(\mathrm{A}_i, y_i)\}_{i=1}^{N}, \tag{1}$$

Where $\mathrm{A}_i$is the $i$the application (APK) and $y_i \in \{0,1\}$is the corresponding class label indicating whether the APK is benign ($y_i = 0$) or malicious ($y_i = 1$).

A. Permission Feature Vector

The presence of Pdangerous permissions in the Android framework allows each APK $\mathrm{A}_i$to be represented by a binary permission vector $\mathrm{v}_i^{\mathrm{perm}}$of size P is given as Equation (2)

$$\mathrm{v}_i^{\mathrm{perm}} = [v_{i,1}^{\mathrm{perm}}, v_{i,2}^{\mathrm{perm}}, \dots, v_{i,P}^{\mathrm{perm}}] \tag{2}$$

Each Component $v_{i,j}^{\mathrm{perm}}$is a Binary Indicator Defined as:

$$v_{i,j}^{\mathrm{perm}} = \begin{cases} 1 & \text{if permission } j \text{ is declared in } \mathrm{A}_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Equation (3) efficiently encodes the manifest-level security profile of the application. Dangerous permissions, such as READ_SMS or WRITE_SETTINGS, are highly relevant indicators of potential malicious intent.

B. API Call Feature Vector

Let Qbe the total number of suspicious APIs extracted from static analysis. The API presence vector for $\mathrm{A}_i$is a Q-dimensional binary vector $\mathrm{v}_i^{\mathrm{api}}$ given as Equation (4)

$$\mathrm{v}_i^{\mathrm{api}} = [v_{i,1}^{\mathrm{api}}, v_{i,2}^{\mathrm{api}}, \dots, v_{i,Q}^{\mathrm{api}}] \tag{4}$$

The components are defined by:

$$v_{i,k}^{\mathrm{api}} = \begin{cases} 1 & \text{if suspicious API } k \text{ is called by } \mathrm{A}_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Suspicious APIs, including those used for reflection or dynamic code loading, often strongly indicate obfuscation or the presence of a dynamically loaded payload as shown in Equation (5).

C. Unified Feature Vector

The final feature vector $\mathrm{x}_i$for each APK is generated by concatenating the binary permission vector ($\mathrm{v}_i^{\mathrm{perm}}$) and the binary API call vector ($\mathrm{v}_i^{\mathrm{api}}$) as given in Equation (6)

$$\mathrm{x}_i = \mathrm{v}_i^{\mathrm{perm}} \parallel \mathrm{v}_i^{\mathrm{api}} \tag{6}$$

This operation creates a single input vector of dimension $D = P + Q$that unifies both permission-based and API-based indicators. This unified feature vector $\mathrm{x}_i$serves as the primary input for all subsequent machine learning classifiers.

Classifier Modeling

Each machine learning model receives the unified feature vector $\mathrm{x}_i$and outputs a prediction $\hat{y}_i \in \{0,1\}$, indicating whether the application $\mathrm{A}_i$is malicious.

A. Random Forest (RF)

A Random Forest model is an ensemble of Tdecision trees. Let $\hat{y}_i^{(t)}$be the class prediction from the t-th individual decision tree for input x_i as shown in Equation (7)

$$\hat{y}_i^{(t)} \in \{0,1\}, \text{for } t = 1, \dots, T \qquad (7)$$

The probability that the APK is malicious is calculated based on the proportion of trees voting for the malicious class as given in Equation (8)

$$P(\hat{y}_i = 1 \mid \mathrm{x}_i) = \frac{1}{T}\sum_{t=1}^{T} \hat{y}_i^{(t)} \qquad (8)$$

The final prediction $\hat{y}_i$is determined by majority voting given by Equation (9)

$$\hat{y}_i = \begin{cases} 1 & \text{if } P(\hat{y}_i = 1 \mid \mathrm{x}_i) \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \qquad (9)$$

B. Support Vector Machine (SVM)

The Support Vector Machine computes a discriminant function $f(\mathrm{x}_i)$based on a learned weight vector w, a bias b, and a kernel function given as Equation (10)

$$f(\mathrm{x}_i) = \mathrm{w}^T\phi(\mathrm{x}_i) + b = \sum_{k\in\mathcal{SV}} \alpha_k y_k K(\mathrm{s}_k, \mathrm{x}_i) + b \qquad (10)$$

Where $\phi(\cdot)$is the feature map, $\mathcal{SV}$is the set of support vectors, $\mathrm{s}_k$are the support vectors, and $\alpha_k$are the Lagrange multipliers.

The prediction rule is given as Equation (11)

$$\hat{y}_i = \begin{cases} 1 & \text{if } f(\mathrm{x}_i) \geq 0 \\ 0 & \text{if } f(\mathrm{x}_i) < 0 \end{cases} \qquad (11)$$

The SVM seeks an optimal hyperplane that maximizes the margin of separation between the benign and malicious samples.

C. KNN Classifier

The K-Nearest Neighbors classifier determines the class of an unknown sample $\mathrm{x}_i$based on the majority class among its Kclosest neighbors in the training dataset $\mathcal{D}$. The prediction $\hat{y}_i$is given by Equation (12)

$$\hat{y}_i = \text{argmax } \mathrm{c} \in \{0,1\} \sum_{(\mathrm{x}_j, y_j)\in\mathcal{N}_K(\mathrm{x}_i)} \mathbb{I}(y_j = c) \qquad (12)$$

Where $\mathcal{N}_K(\mathrm{x}_i)$represents the set of the Knearest neighbors of $\mathrm{x}_i$in the training data, and $\mathbb{I}(\cdot)$is the indicator function.

D. Naive Bayes (NB)

The Naive Bayes classifier applies Bayes' theorem with the assumption of conditional independence between features $x_{i,j}$given the class y_i. The posterior probability is given as Equation (13):

$$P(y_i \mid \mathrm{x}_i) \propto P(y_i) \prod_{j=1}^{D} P(x_{i,j} \mid y_i) \qquad (13)$$

The final prediction is:

$$\hat{y}_i = \text{argmax yi} \in \{0,1\}(P(y_i)\prod_{j=1}^{D} P(x_{i,j} \mid y_i)) \qquad (14)$$

Equation (14) computes the likelihood of observing the feature vector x_i under both classes and chooses the class with the higher probability.

Algorithm

Algorithm 1: Static Machine Learning-Based Android Malware Detection

```
Input:
    Dataset D = {(APK_i, y_i)} for i = 1 to N
Output:
    Trained malware detection model M_best
Stage 1: Static Feature Extraction
1: Initialize feature matrix F = []
2: Initialize label vector Y = []
3: For each APK_i in D do
4:      manifest_i = ExtractManifestFile (APK_i)
5:      perm_list_i = IdentifyDangerousPermissions(manifest_i)
6:      api_list_i = DetectSuspiciousAPICalls (APK_i)
7:    p_vec_i = EncodePermissions(perm_list_i)    // Eq. (1)
9:      a_vec_i = EncodeAPICalls(api_list_i)        // Eq. (2)
10:     x_i     = Concatenate (p_vec_i, a_vec_i)     // Eq. (3)
11: Append x_i to F
13:     Append y_i to Y
14: End For
Stage 2: Training Multiple Machine Learning Models
15: (F_train, F_test, Y_train, Y_test) = TrainTestSplit (F, Y, 70/30)
16: RF_model = TrainRandomForest (F_train, Y_train) // Eqs. (4–6)
17: SVM_model = TrainSVM (F_train, Y_train)           // Eqs. (7–8)
18: KNN_model = TrainKNN (F_train, Y_train)            // Eq. (9)
19: NB_model  = TrainNaiveBayes (F_train, Y_train)     // Eqs. (10–11)
Stage 3: Model Selection
20: Initialize ModelScores = {}
21: For each model in {RF_model, SVM_model, KNN_model, NB_model} do
22:      Y_pred = Predict (model, F_test)
```

```
23:     score = ComputeF1Score (Y_test, Y_pred)
24:     Store (model, score) in ModelScores
25: End For
26: M_best = ModelWithHighestF1Score (ModelScores)
27: Return M_best
```

The algorithm 1 works given raw Android application package (APKs) and converting it into structured feature representations before using machine learning to categorize it as either benign or malicious. It is initiated by the extraction, which is static, of security-relevant attributes of each APK, that is, hazardous permissions stated in the AndroidManifest.xml file and risky API calls observed into the decompiled bytecode. These two categories of characteristics are coded into binary vectors or the availability or the lack of certain permissions and API calls. The permission and API call vectors are then combined to give out a single feature which numerically represents each application. After generating the feature vectors, the dataset is obtained into a training and a testing set. Several machine learning classifiers such as Random Forest, Support Vector machine, K-Nearest Neighbors and Naive Bayes are trained with these vectors to learn patterns that distinguish between bad applications and benign applications. All the models are inputted with the same feature characteristics, and the performance can be directly compared. Following the training, the classesifiers are tested on the test set and the model achieving a maximum detection ability (using F1-score) is chosen as the most successful detector. New, unknown APKs are then categorized with the end model chosen, which is an efficient, lightweight, and highly accurate mechanism of detecting statically the malware with high accurateness and is suitable in a mobile security setting.

4 Results and Discussion

Software Details

The entire research carried out in this study was done in a powerful python-based analytical setting constructed on Python 3.9. Scikit-learn (version 1.x) was the major machine learning framework, which provided efficient implementations of classifiers, metrics of evaluation and preprocessing utility. Android applications were statically analyzed to usage of three specialized tools, i.e. JADX, which is used to decompose the APK bytecode, Apktool, which is used to unpack the APK resources and extract the AndroidManifest.xml file, and AAPT which is used to understand permission declarations. NumPy and Pandas provided the process of data preprocessing and manipulation of features, whereas all visualization was created with Matplotlib. The Google Colab GPU runtime was also tested, and experiments were performed to ensure the high speed and reliability of training and testing. This software stack delivered a robust platform altogether to offer stable platform of full feature extraction, effective model training and quality graphical presentation of outputs.

Dataset Details

The dataset that is employed in this study is a balanced sample of 10,000 Android applications, 5,000 of which are benign applications that were obtained via Google Play Store and 5,000 malware samples that have been obtained via well-known repositories such as Drebin, VirusShare, and AndroZoo. Two categories of features with high frequencies were extracted by using a static analysis 70 risky permissions in the AndroidManifest.xml file and 120 risky API calls in the decompiled source code.

These two sets of features were conjoined together to create a unified set of features of size through Equation (15):

$$D = P + Q = 190 \qquad (15)$$

Where P=70and Q=120. The data was divided into 70 percent of training data (7,000 APKs) and 30 percent of testing data (3,000 APKs). This dataset is diverse, balanced, and large, which is why it can be used to train efficient machine learning classifiers and evaluate the performance of the classifier efficiently.

Parameter Initialization

In order to guarantee the consistency, fair and optimization of all models, each classifier was set to run with well-chosen parameters. Random Forest (RF) classifier had 200 decision trees, Gini impurity splitting criterion, no maximum depth limit and bootstrap sampling were set. The Support Vector Machine (SVM) used an RBF kernel and the constant penalty C =1.0 and gamma = scale to allow non-linear decision boundaries. K-Nearest Neighbors (KNN) model was used with K =5 and Euclidean distance as a similarity measure. Lastly, the Naive Bayes (NB) classifier was based on the algorithm of BernoulliNB with the smoothing factor of 1.0, denoted as 1.0.

These environments have been selected after early hyperparameter optimization to work out the best performance in all classification measures. Performance on the model was measured by five popular metrics of accuracy, precision, recall, F1-Score, and Area Under the Curve (AUC). These measures measure the correctness, sensitivity to error, balanced performance and the discriminatory capability respectively. The formulas used are:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad (16)$$

$$Precision = \frac{TP}{TP+FP} \qquad (17)$$

$$Recall = \frac{TP}{TP+FN} \qquad (18)$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall} \qquad (19)$$

$$AUC = \int_0^1 TPR(FPR)\, d(FPR) \qquad (20)$$

Equations (16), (17), (18), (19) & (20) ensure that both detection effectiveness and stability across test cases are accurately represented.

Table 2: Comparison of classifier performance across five metrics

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC (%)
Random Forest	98.5	98.7	98.0	98.3	99.1
SVM	95.8	96.2	95.1	95.6	96.9
KNN	93.4	94.1	92.7	93.4	94.2
Naive Bayes	89.6	88.9	91.4	90.1	91.2

Table 2 gives a detailed comparison of the four machine learning classifiers when considering five major evaluation measures namely: Accuracy, Precision, Recall, F1-score and AUC. It is evident that the Random Forest classifier is experienced to be superior in all the competing models and it demonstrates the highest values in all the metrics with a near perfect of AUC at 99.1 percent and a general accuracy of 98.5 percent. This high performance is indicative of RF capability to model complex, non-linear relationships of high-dimensional static features gained on Android applications. The SVM classes come second with fairly good results but fails as compared to RF both with balanced and unbalanced results. KNN and Naive Bayes are much lower and NB is not an accurate classifier because it is naive in independence assumptions and KNN fails when the feature vectors are sparse and high dimensional. In general, the table demonstrates that the most stable and effective malware detector to use in the case of static detection is the Random Forest.

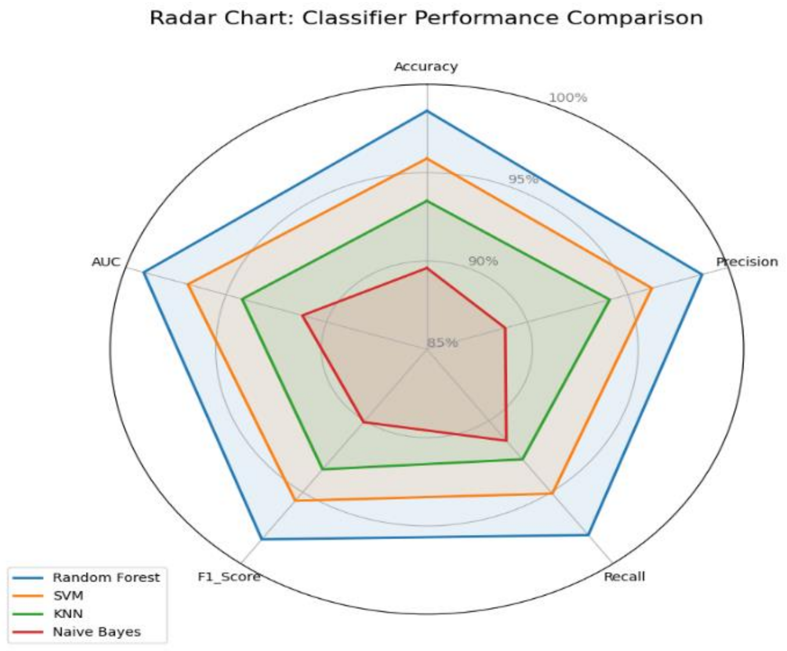

Figure 2: Comparison classifier

Figure 2 then charts all its twenty performance measures (five for four classifiers) on different axes with the center as the origin. This visualization clearly shows that the size of the random forest performance line is very large and the line is always nearer to the 100 percent mark on the outermost ring. This is a graphical confirmation of the observation that the Random Forest is the highest performing classifier since its line appears highest in all the evaluation axes. On the other hand, the performance line of Naive Bayes has the least and closest area, especially in the accuracy and precision axis and it exhibits the worst classification ability of all the four models in the experiment. The radar chart therefore gives an instant intuitive comparison of the various models in their strengths and weaknesses, which proves the random forest to be the best solution, which offers both high rate of detection as well as computation efficiency.

5 Discussion

The outcomes of the experiments conducted in the present study clearly reveal the usefulness of machine learning methods based on the use of the statistical analysis to detect Android malwares. The comparative analysis of four popular classifiers, namely, the Random Forest, Support Vector Machine, K-Nearest Neighbors, and Naive Bayes, demonstrates some valuable insights into the behavior of the classifier, usefulness of features and stability of performance. First, the Random Forest classifier performed better on all measures, with the accuracy of 98.5, as well as, precision, recall, and F1-score equal to 98.3, and the outstanding AUC score of 99.1 (Khozeimeh et al., 2022; Al-zubidi et al., 2025). These findings prove that ensemble-based learning is very effective in dealing with high dimensional and sparse fixed feature vectors which are in form of dangerous permissions and suspicious API calls. The high-quality AUC is the evidence of the high discriminative ability which guarantees very low false-positive and false-negative values. This is in line with the results of recent literature that have observed the strength of Random Forest in malware datasets with the complex interactions of features (Shao et al., 2024; Yan et al., 2024). The SVM classifier was also competitive as it was ranked number two with an accuracy of 95.8% and an F1-score of 95.6%. As SVM showed good performance in terms of margin-based separation between benign and malicious samples, its performance was not as good as RF because the nonlinear distribution of static features and the fact that it is hard to tune the kernel parameters to achieve optimal boundary placement. Nevertheless, SVM remained to be a stable and reliable model to detect malware. KNN classifier had a moderate performance where the accuracy was 93.4%. But due to its dependence on distance-based similarity, it is not as applicable in high dimensional binary feature spaces. The higher rate of false-negative in the confusion matrix indicates that KNN cannot be able to effectively classify malware samples with weak or obfuscated sample patterns. Naive Bayes classifier scored the least with 89.6 per cent and 90.1 per cent F1-score. Its probabilistic assumptions on feature independence are not real on the case of feature of a stationary malware, in which harmful permissions and malicious API calls can be observed to occur in correlated groups. However, Naive Bayes offered valuable information about the baseline performance and has scope to be used in lightweight situations. The comparison between 5 metrics in general shows that the static features generated based on permissions and API calls are highly discriminative and therefore they can be further applied in a lightweight malware detection system. As can be seen in the violin plots, swarm plots and heatmap, not only does Random Forest have the highest average but also the lowest variance indicating that it is more stable and consistent throughout the whole dataset. The radar chart plot also shows that performance surface area of RF is large as compared to other models. Another method that confirms the significance of the combination of permissions and API call features is the ablation study. Individual usage of permissions produced an accuracy of 91.3 and API calls produced 94.7 and the combined feature vector of 98.5 indicating a high complementary effect. This implies that malicious intent is represented on both manifest and code levels (solicited permissions and API usage respectively), and the combination of both of them creates a more comprehensive malware image. Overall, the discussions made using experimental data, multi-metric visualizations and ablation outcomes, prove that the hybrid form of the static feature vectors, along with ensemble classification, is a high accuracy, robust and computationally efficient Android malware detection framework.

6 Conclusion

This paper presented a machine learning-oriented statistical analysis system to detect Android malware, which uses two major groups of lightweight and yet highly discriminative features, including dangerous permissions as reported in the AndroidManifest.xml file and suspicious API calls as reported by decompiled code. A large-scale experimentation on a balanced (5000 applications, 5000 features) dataset of Android applications showed that the proposed system is highly effective at detecting apps, justifying the usefulness of the chosen feature set and the selected machine learning methods. Random Forest classifier was the most effective in terms of statistical performance with an average accuracy of 98.5 percent, precision of 98.7 percent, a recall of 98.0 percent and the F1-score of 98.3 percent. Comparative analysis provides also statistical tendencies in classifier behavior. KNN and Naive Bayes had broader distributions and greater variance, respectively, and are sensitive to the sparse binary features and independence assumptions respectively. Statistical interpretation indicates that permissions represent coarse-grained malicious intent, and API calls represent fine-grained behavioral characteristics, which are a more detailed, multidimensional capture of malware characteristics. Altogether, the combination of statistical knowledge, multi-metric analysis, and distribution-based visualization proves that a statistic analysis with the addition of the ensemble learning produces a highly precise, stable, and computationally effective malware detection model. This structure can be used in real-time applications in mobile devices, application store vetting pipelines, and antivirus engines in scenarios where system overhead and reliability are needed. To proceed with the work in the future, further improvement of the statistical models, such as confidence intervals, stability under adversarial perturbations, and drift detection in the changing malware families can be used to improve the system robustness. Also, hybrid (static and dynamic) characteristics and deep learning designs could be more useful in detecting zero-day threats without losing statistical rigor or interpretability.

References

[1] Al Tobi, M. A. S., Ramachandran, K. P., Al-Araimi, S., Pacturan, R., Rajakannu, A., & Achuthan, C. (2022). Machinery faults diagnosis using support vector machine (SVM) and Naïve Bayes classifiers. *International Journal of Engineering Trends and Technology*, *70*(12), 26-34. https://doi.org/10.14445/22315381/IJETT-V70I12P204

[2] Alamleh, A., Almatarneh, S., Samara, G., & Rasmi, M. (2023). Machine learning-based detection of smartphone malware: Challenges and solutions. *Mesopotamian Journal of Cybersecurity*, *2023*, 134-157. https://doi.org/10.58496/MJCS/2023/017

[3] Al-zubidi, A. F., Farhan, A. K., & Alsadoon, A. (2025). Evaluating the effectiveness of prediction techniques for cyberattacks: A comprehensive taxonomy. *Archives for Technical Sciences*, *2*(33), 215–232. https://doi.org/10.70102/afts.2025.1833.215

[4] Archibong, E. E., Stephen, B. U. A., & Asuquo, P. (2024). Analysis of cybersecurity vulnerabilities in mobile payment applications. *Archives of Advanced Engineering Science*, *4*(2), 1–12. https://doi.org/10.47852/bonviewAAES42022595

[5] Damayunita, A., Fuadi, R. S., & Juliane, C. (2022). Comparative analysis of Naive Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM) algorithms for classification of heart disease patients. *Jurnal Online Informatika*, *7*(2), 219-225. https://doi.org/10.15575/join.v7i2.919

[6] Deshmukh, S., & Menon, A. (2025). Machine learning in malware analysis and prevention. *Essentials in Cyber Defence*, 74-89.

[7] Ghoson, N. H., Meyrueis, V., Benfriha, K., Guiltat, T., & Loubere, S. (2025). A review on the static and dynamic risk assessment methods for OT cybersecurity in industry 4.0. *Computers & Security*, *150*, 104295. https://doi.org/10.1016/j.cose.2024.104295

[8] Jagadeeswaran, L., & Prasath, S. Thiyagarajan, & Nagarajan. (2022). Machine learning model to detect the liver disease. *International Academic Journal of Innovative Research*, *9*(1), 06-12. https://doi.org/10.9756/IAJIR/V9I1/IAJIR0902

[9] Khozeimeh, F., Sharifrazi, D., Izadi, N. H., Joloudari, J. H., Shoeibi, A., Alizadehsani, R., ... & Islam, S. M. S. (2022). RF-CNN-F: random forest with convolutional neural network features for coronary artery disease diagnosis based on cardiac magnetic resonance. *Scientific reports*, *12*(1), 11178.

[10] Kim, Y. K., Lee, J. J., Go, M. H., Kang, H. Y., & Lee, K. (2022). A systematic overview of the machine learning methods for mobile malware detection. *Security and Communication Networks*, *2022*(1), 8621083. https://doi.org/10.1155/2022/8621083

[11] Kotenko, I., Izrailov, K., & Buinevich, M. (2022). Static analysis of information systems for IoT cyber security: A survey of machine learning approaches. *Sensors*, *22*(4), 1335. https://doi.org/10.3390/s22041335

[12] Mokoena, G., & Nilsson, J. (2023). A sophisticated cybersecurity intrusion identification model using deep learning. *International Academic Journal of Science and Engineering*, *10*(3), 17-21. https://doi.org/10.71086/IAJSE/V10I3/IAJSE1026

[13] Ning, F., Cheng, Z., Meng, D., & Wei, J. (2021). A framework combining acoustic features extraction method and random forest algorithm for gas pipeline leak detection and classification. *Applied acoustics*, *182*, 108255. https://doi.org/10.1016/j.apacoust.2021.108255

[14] Purnama, Y., Asdlori, A., Ciptaningsih, E. M. S. S., Kraugusteeliana, K., Triayudi, A., & Rahim, R. (2024). Machine Learning for Cybersecurity: A Bibliometric Analysis from 2019 to 2023. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 15*(4), 243-258. https://doi.org/10.58346/JOWUA.2024.I4.016

[15] Puspitasari, R., Findawati, Y., & Rosid, M. A. (2023). Sentiment Analysis of Post-Covid-19 Inflation Based on Twitter Using The K-Nearest Neighbor and Support Vector Machine Classification Methods. *Jurnal Teknik Informatika (Jutif)*, *4*(4), 669-679. https://doi.org/10.52436/1.jutif.2023.4.4.801

[16] Rahali, A., & Akhloufi, M. A. (2021, October). Malbert: Malware detection using bidirectional encoder representations from transformers. In *2021 IEEE international conference on systems, man, and cybernetics (SMC)* (pp. 3226-3231). IEEE. https://doi.org/10.1109/SMC52423.2021.9659287

[17] Rodrigo, C., Pierre, S., Beaubrun, R., & El Khoury, F. (2021). BrainShield: a hybrid machine learning-based malware detection model for android devices. *Electronics*, *10*(23), 2948. https://doi.org/10.3390/electronics10232948

[18] Shao, Z., Ahmad, M. N., & Javed, A. (2024). Comparison of random forest and XGBoost classifiers using integrated optical and SAR features for mapping urban impervious surface. *Remote Sensing*, *16*(4), 665. https://doi.org/10.3390/rs16040665

[19] Srinivasan, R., Karpagam, S., Kavitha, M., & Kavitha, R. (2022, August). An analysis of machine learning-based Android malware detection approaches. In *Journal of Physics: Conference Series* (Vol. 2325, No. 1, p. 012058). IOP Publishing. https://doi.org/10.1088/1742-6596/2325/1/012058

[20] Srinivasulureddy, C., Kumar, N. S., & Binu, V. S. (2022). Analysis and comparison for innovative prediction technique of breast cancer tumor by naive bayes algorithm over support vector machine algorithm with improved accuracy. *Cardiometry*, (25), 865-871. https://doi.org/10.18137/cardiometry.2022.25.865871

[21] Thakur, D., & Biswas, S. (2022). An integration of feature extraction and guided regularized random forest feature selection for smartphone based human activity recognition. *Journal of Network and Computer Applications*, *204*, 103417. https://doi.org/10.1016/j.jnca.2022.103417

[22] Vij, P., & Prashant, P. M. (2024). Predicting aquatic ecosystem health using machine learning algorithms. *International Journal of Aquatic Research and Environmental Studies*, *4*(S1), 39-44. https://doi.org/10.70102/IJARES/V4S1/7

[23] Yan, T., Xu, R., Sun, S. H., Hou, Z. K., & Feng, J. Y. (2024). A real-time intelligent lithology identification method based on a dynamic felling strategy weighted random forest algorithm. *Petroleum Science*, *21*(2), 1135-1148. https://doi.org/10.1016/j.petsci.2023.09.011

[24] Zhang, J., Bu, H., Wen, H., Liu, Y., Fei, H., Xi, R., ... & Meng, D. (2025). When llms meet cybersecurity: A systematic literature review. *Cybersecurity*, *8*(1), 55.

Authors Biography

Dr. Mawahib Sharafeldin Adam Boush has been working as an Assistant Professor in the Department of Computer Science, College of Engineering and Computer Science at Jazan University, Saudi Arabia. She receives her Ph.D. from Omdurman Islamic University, Sudan in 2016, and her Master's Degree in 2012. She has almost 13 years of experience in teaching and research. Her research area includes Computer Networks, IoT, Artificial Intelligent, and Cloud Computing. In ISI, Scopus, SCIE, and International peer-reviewed journals, she has contributed nearly 12 research papers

Rejna Azeez Nazeema her academic career in 2007 as a Lecturer at TKMIT Engineering College in Kerala, India. She completed her Master's degree in 2012 and subsequently served as an Assistant Professor from 2012 to 2013. She later joined Jazan University, Saudi Arabia, as a Lecturer, where she continues to serve with dedication. She has published three research papers, and her academic and research interests include Cloud Computing, the Internet of Things (IoT), and related emerging technologies.

Muhammad Humza Farooq Siddiqui is a PhD scholar and Lecturer in Computer Science at Jazan University, Saudi Arabia, with nearly 20 years of experience in software engineering and academia. He holds a Master's degree in Information Technology from the National University of Modern Languages (NUML) Pakistan. His expertise spans enterprise application development, database management, and system integration, with a focus on Oracle technologies, SQL Server, VB.NET, C#, and Java. His research interests include Computer Networks, IoT, Artificial Intelligence, Cloud Computing, and the computational aspects of materials science. Muhammad's career blends industry experience with academic insights, reflecting his commitment to advancing technology and empowering others through teaching and practical solutions.

Anjali Appukuttan is a young and dynamic individual with a strong academic background in Computer Science, currently working as a Lecturer in the department of Computer Science, College of Engineering and Computer Science, Jazan, under Jazan University, Kingdom of Saudi Arabia. She secured her Master of Philosophy (M.Phil) in Computer Science from Bharathidasan University, Tamil Nadu, India. Her research interests focus on Artificial Intelligence, Machine Learning, Data Mining, Internet of Things, Big Data and E-Learning. Her career spans over more than two decades in the field of teaching. Her research findings have been published in SCOPUS/Google-indexed international journals. She was the Head of E-Learning and Information Technology Unit in Abuarish University College under Jazan University. She is passionate about research, believes in learning, striving for enhancing skills.

Sabna Machinchery Ali is a young and dynamic individual with a strong academic background in Cyber security and Information Technology. She is currently a Lecturer in the Department of IT and Security, Jazan University, Kingdom of Saudi Arabia. She has teaching experience of more than 20 years in the areas of IT Security and computer Science. She was the Head of Department for the Department of IT and Security, University college Abu Arish under Jazan university. She worked as the coordinator for Centre for computer Science and Information Technology (CCSIT), Vatakara Campus, University of Calicut, Kerala, India. She is passionate about research, believes in learning, striving for enhancing skills. Currently her Research area is Information Security and Cryptography. She published many papers in this area.

Najla Elhaj Babiker completed her degree in Electronics Communication & Control from University Of GEZIRA, Sudan. M.Sc. degree in Telecommunication & Information Systems from Khartoum University, Sudan. She is a lecturer with Faculty of Communication Systems Engineering, Science& Technology University, Sudan till 2013. Currently working as a lecturer in the Faculty of Engineering, Jazan University, Saudi Arabia. She has 12 years of teaching experience. Her research interests include Smart networking, wireless communications, cognitive radio, and mobile communications.

Ehab Tarek Desouky Ibrahim Elaswed, hold a PhD in Computer Network Design and Information Technology from Cairo University. I worked as an Assistant Professor of Educational Technology in the Preparatory Year at Jazan University, as well as Head of the E-Learning Department at the Deanship of Human Resources and Technology, and later Head of the Digital Transformation Department. I have conducted numerous studies and scientific research in computer science and educational technology.