Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: https://www.science-gate.com/IJAAS.html

A hybrid CNN-LSTM deep learning framework for enhanced Android malware classification

Altyeb Taha [1,*], Ahmed Hamza Osman [2], Yakubu Suleiman Baguda [2]

[1]*Department of Information Technology, Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Jeddah 21911, Saudi Arabia*
[2]*Department of Information Systems, Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Jeddah 21911, Saudi Arabia*

ARTICLE INFO

Article history:
Received 8 August 2025
Received in revised form
1 November 2025
Accepted 27 April 2026

Keywords:
Android malware
Deep learning
Hybrid model
CNN
LSTM

ABSTRACT

Smartphones increasingly store sensitive data, making them attractive targets for cyberattacks. Among mobile platforms, Android is especially vulnerable due to its open-source nature and widespread use. Malicious applications threaten user privacy, compromise data, and damage system integrity, highlighting the need for effective detection methods. This paper introduces a hybrid deep learning model for Android malware classification that integrates Convolutional Neural Networks (CNNs) for feature extraction with Long Short-Term Memory (LSTM) networks for capturing sequential patterns. The proposed model is evaluated on two benchmark datasets, Drebin and AndroZoo, achieving accuracies of 98.30% and 97.25%, respectively, and outperforming existing machine learning approaches.

1. Introduction

The recent advancements in smartphones and mobile applications have significantly transformed our daily interactions and routines. Increasingly, people are becoming more dependent on smartphones to perform many activities. Examples of smartphone services provided via apps include E-health, online education, E-commerce, social networking, and many other services. Thus, smartphones have become a necessary component of our daily lives and have played a critical role in them. The number of smartphone users worldwide is projected to grow steadily from 2024 to 2029, with an expected increase of approximately 1.5 billion users (Liu et al., 2025). This reflects a consistent upward trend in smartphone adoption observed in recent years. Android dominates 90% of the smartphone operating system market (Enichen et al., 2025). The following motives are the primary reasons for this dominance: First, its open-source architecture empowers developers to customize and deploy it freely. Second, Google oversees Android's ongoing maintenance and enhancements. Third, it provides a robust API designed specifically for building enterprise-grade Android applications (Prasad et al., 2024). Although smartphones make life easier for people, they are also susceptible to malware infection and cyberattacks due to online services and social networks. There is a notable surge in mobile-related cyber threats, encompassing phishing schemes and mobile banking malware. The number of malware infections increases by 100 million per year, involving malicious software, adware, or potentially unwanted applications.

Google Play's permission-based security model is designed to restrict applications from accessing sensitive information without explicit user authorization (Hu, 2024). This permission system notifies the users about the resources that will be accessed by the app before the installation. Before proceeding with the installation, an agreement must be explicitly approved by the smartphone's user. However, Google Play's scheme cannot completely protect the users since they prefer to accept the declaration without thoroughly inspecting the authorization. As a result, most users of smartphones are vulnerable to harm because they overlook the permission risk, which leads to app exploitation. Therefore, research into Android malware is important to protect the apps from attacks.

Numerous machine learning approaches have been proposed for Android malware classification, as discussed in studies such as Alazab et al. (2020), Su

* Corresponding Author.
Email Address: aaataha@kau.edu.sa (A. Taha)
https://doi.org/10.21833/ijaas.2026.04.024
Corresponding author's ORCID profile:
https://orcid.org/0000-0001-9086-3085

et al. (2016), Fereidooni et al. (2016), Wang et al. (2017a), and other related research. However, with the increase in the amount and complexity of malware apps, the classification of Android malware using these classical methods becomes more challenging.

Deep learning creates data abstraction at a large scale of data by utilizing numerous processing layers. It is a branch of machine learning that utilizes neural networks modeled on the construction and role of the human brain. In this context, each artificial neuron acts as a mathematical abstraction of its biological counterpart. The inputs to the neuron are combined with a related weight, added, and an output result is produced using an activation function. One of deep learning's key benefits over other machine learning algorithms is its capability to perform feature engineering by itself without being prompted to do so. To enable quick learning, a deep learning system will examine the data for features that correlate and integrate them.

In comparison to traditional Android malware classification approaches, malware detection based on deep learning methods can gain knowledge from the significant features of Android malicious apps. This enables them to find sophisticated novel malware as well as variants of classic malware apps. The most well-known, effective, and commonly applied deep learning approaches are LSTM and CNN. Although deep learning has the capacity to enhance Android malware classification, the performance of a single network may be limited when dealing with more complicated challenges. Numerous researchers have considered the employment of multiple deep learning networks in combination.

This study's primary goal is to enhance the classification of Android malicious applications. To attain this objective, we introduced an intelligent hybrid classification model based on leveraging the benefits of deep learning approaches. The proposed method identifies Android malware apps by leveraging the convolutional layers' capacity to select valuable information and understand the inner representation of Android apps, as well as LSTM layers' efficacy in recognizing short-term and long-term dependencies. Modern machine learning approaches for classifying Android malware applications were compared to the proposed model in the evaluation process. Our tests show that using LSTM models in combination with CNN is a successful strategy for classifying Android malware and significantly improves classification performance. The suggested approach offers the following major contributions:

- An intelligent hybrid approach that integrates two convolutional blocks and an LSTM block, optimized to capture both spatial and sequential characteristics of Android application behavior. Unlike previous studies that either use CNN or LSTM individually, our model exploits CNN's strength in extracting local feature patterns and LSTM's capacity for learning temporal dependencies, leading to improved malware classification accuracy.
- Two publicly available Android malware datasets were employed to assess the achievement of the suggested method.
- The efficiency of the suggested model for Android malware categorization is illustrated by comparing its effectiveness with other machine learning techniques.

The remainder of this research is arranged as follows: A brief overview of current studies on the use of modern machine learning algorithms in Android malware classification is provided in Section 2. The suggested intelligent hybrid deep learning approach is introduced in detail in Section 3. The findings of the conducted experiments and discussions of our investigation are outlined in Section 4. The conclusion of the research, along with some suggestions for more research are shown in Sections 5 and 6.

2. Literature survey

Due to the rapidly increasing number of malicious Android apps, many researchers have suggested machine learning-based methods for categorizing Android malicious apps. This section presents related research.

Static methods for classifying Android malicious apps depend on obtaining syntactic characteristics without running the application. The pattern of the used permissions and application programming interface (API) calls is one of the two aspects collected based on static analysis of the Android app. Among other features, the most frequently used features in Android are permissions. In addition to having relatively low computing overheads, static techniques offer the benefit of not requiring restricted execution environments. Based on API calls and Android permissions, Alazab et al. (2020) proposed a scheme for classifying Android malicious apps. Additionally, they divided programs into three categories—ambiguous, hazardous, and disruptive using the frequency distribution of API calls and permissions. The researchers used a dataset that contains 13,719 malicious programs and 14,172 benign ones. The suggested solution was built using the random tree, J48, k-Nearest Neighbors (KNN), Random Forest (RF), and Naive Bayes (NB) algorithms. With an accuracy of 94.30%, the random forest achieved the highest performance.

Su et al. (2016) presented DroidDeep, which included several characteristics acquired through static analysis. These comprised permissions that were requested, permissions that were utilized, API calls, activities, and elements of an app. The attributes were then entered into the deep learning algorithm. The researchers state that the approach aids in learning the usual attributes and minimizes the number of attributes. To categorize the apps, a Support Vector Machine (SVM) based detector was

created. DroidDeep employed a dataset of 3986 goodware and 3986 malware apps and attained 97.5% accuracy. To categorize the Android malware, Fereidooni et al. (2016) employed five separate feature sets: harmful behaviors determined by the authors, suspicious API calls, Android permissions, commands of the system, and intents. They employed a dataset of 11,187 goodware and 18,677 malware APKs. The following algorithms were used: Adaptive boosting (AdaBoost), decision trees, deep learning, KNN, RF, Logistic Regression (LR), Extreme gradient boosting (XGBoost), SVM, and NB. However, only XGBoost's complete results were provided. Based on the results, XGBoost had an accurate rate of 97%.

Using a collection of features that includes network addresses, API requests, and requested permissions, Wang et al. (2017b) suggested a technique for classifying Android malicious apps. The researchers employed a dataset of 4403 goodware and 3982 malware apps to train KNN, random forest, and J48. The benign programs come from the Google Play store, whereas the malicious apps dataset was composed of the Drebin (Arp et al., 2014) and Gnome Project (Zhou and Jiang, 2012) datasets. The proposed technique has a 98.2% accuracy rate. Bai et al. (2020) proposed the FAMD framework, which makes use of permissions and n-gram opcodes. FAMD employed a dataset that comprised malware programs from Drebin as well as 5666 benign applications. Categorical boosting (CatBoost) was employed as the ML model for the categorization of Android malware apps, and it performed better than random forest, XGBoost, and KNN. The researchers, based on permissions and a 5-gram sequence, reported the highest accuracy percentage of 96.21%.

McLaughlin et al. (2017) used a distinct method by considering the byte-code of the app as text. The authors employed a Convolutional Neural Network as a classifier and evaluated the system on a data set including 863 benign and 1260 malicious. The highest attained accuracy is 98%. TFDroid is a framework that Lou et al. (2019) suggested. Along with topics and permissions, they utilized data flows as part of the used features. To extract concepts from the applications, TFDroid employed Latent Dirichlet Allocation. The SVM classifier that was trained on 5161 good apps and 3427 bad apps has a 95.32% accuracy rate.

Talha et al. (2015) utilized a dataset that included 6909 malicious applications and 1853 good apps. They presented Apk Auditor, a permission-based framework that computes a malicious score for all permissions based on the number of malicious apps that employ a specific permission divided by the number of malicious applications in the dataset. Logistic regression classifier has an accuracy of 88.28%, which may be unreliable because of the data set's class imbalance.

We previously presented an AI-based scheme for categorization of Android malicious apps based on permission attributes in our previous research (Altaher, 2017). The evolving clustering approach was improved to include a dynamic mechanism for adjusting the radius and centers of the grouped attributes. The evaluation outcomes indicate that the suggested scheme identifies Android malware with 90% accuracy. Altaher and Barukab (2017) proposed an ANFIS model integrated with fuzzy clustering for the categorization of Android malicious apps. The outcomes show that the suggested scheme reached a 91% classification accuracy. Taha and Malebary (2021) introduced a combined method for Android malware identification by merging fuzzy logic with the LightGBM algorithm. The fuzzy clustering approach is used to generate significantly meaningful groups of permissions for Android apps that represent certain properties of the Android apps. The LightGBM algorithm utilizes the permissions of the app and associates FCM clusters as inputs and outputs the categorization findings as goodware or malware apps after completing the training process. We utilized a dataset that included 5560 malicious Android apps and 23,453 benign Android apps. The highest accuracy attained is 94.63%. We proposed an ensemble learning method for Android malware classification in our earlier research (Taha and Barukab, 2022). There are two phases in the suggested method. First, the base learner consists of several machine learning methods, such as DT, LR, AdaBoost, SVM, and XGBoost algorithms. To categorize the classification likelihoods from the basic learner, a meta learner is employed, which uses genetic algorithms to tune the RF model's configuration. 5560 Android malicious apps and 9476 benign apps made up the dataset used to test the suggested technique. The suggested scheme has the best accuracy (94.15%).

Taha et al. (2021) proposed an ensemble of several classifiers that utilize fuzzy integrals for the identification of Android malware. To combine and integrate the classification outcomes from many classifiers, including Random Forest, Light-GBM, XGBoost, AdaBoost, and Decision Tree, it uses Choquet fuzzy integral as a combination method. The introduced scheme also uses dynamic fuzzy measures to take into consideration the cohesion between each potential subset of models as well as the data's dynamic character in each model. This makes it possible for the suggested strategy to merge the classification results with several algorithms. The suggested method's accuracy was 95.08% according to testing findings utilizing the dataset, which included 9476 Android benign apps and 5560 Android malware apps.

Yeboah and Baz Musah (2022) employed a deep learning-based model consisting of a 1-dimensional convolutional neural network (1D CNN) to automate the detection of Android malware. Their suggested approach automatically extracts features from semantically embedded n-grams of raw static operation code (opcodes) sequences to determine the maliciousness of a binary file. Using an Android dataset comprising 4951 malware and 2477 benign

samples, their approach achieved an accuracy of 98%. Kumar et al. (2023) utilized a Long Short-Term Memory (LSTM) approach to detect Android malware. The proposed approach aims to detect Android malware by scrutinizing structures of system actions, API calls, or other active communication generated by Android apps. Their proposed scheme achieved an accuracy of 96.65%.

3. Long short-term memory and convolutional neural network

Deep learning has become a potentially significant approach in the machine learning area; it has been effectively utilized in many classification and prediction problems (LeCun et al., 2015). The suggested intelligent hybrid scheme for Android malware classification combines the advantages of two neural network models. CNNs excel in sequence modeling due to their ability to extract local features across temporal or spatial positions by applying convolution operations between the input sequence and learnable filters, making it suitable for forming the Android permissions sequence. LSTM may dynamically infer long-period associations in a group of unrelated data points (Tang et al., 2021).

3.1. Long short-term memory (LSTM)

The LSTM is a form of recurrent neural network (RNN). Because of its distinctive design, it is an effective deep learning technique. RNN typically struggles with the issue of declining gradients, which occurs while learning large data series. LSTM can tackle this problem by providing a memory cell that determines when to forget specific information. The input, hidden, and output layers comprise an LSTM network. The memory cells in its hidden layers are the basic features of the LSTM. Instead of conventional neuron nodes, memory cells contain memory locations. Four gates make up each memory cell: the input, forget, candidate, and output gates. The input attributes are classified as either needing to be maintained or discarded by the forget gate. The memory cells in the LSTM organization are revived by the input gate, and the output gate is always in charge of controlling the hidden state. Additionally, by employing dedicated memory cells and gating methods, LSTM efficiently counteracts the vanishing and exploding gradient problems that standard RNNs face during training (Aldhyani and Alkahtani, 2022). Fig. 1 displays the construction of the LSTM component. The LSTM maps the vector of input sequence X = (X_1, X_2, X_n) with output likelihood vector by determining the network unit activations utilizing the following formulas (Graves et al., 2013), repeated from t = 1 to n.

$$i_t = \sigma(W_{ix}X_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i) \quad (1)$$
$$f_t = \sigma(W_{fx}X_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f) \quad (2)$$
$$o_t = \sigma(W_{ox}X_t + W_{oh}h_{t-1} + W_{oc}c_{t-1} + b_o) \quad (3)$$
$$c_t = f_t * c_{t-1} + i_t * tanh\,(W_{cx}X_t + W_{ch}h_{t-1} + b_c) \quad (4)$$
$$h_t = o_t * tanh\,(c_t) \quad (5)$$
$$y_t = W_{yh}h_{t-1} + b_y \quad (6)$$

W denotes the collection of weight matrices; for instance, $W_{(ix)}$ specifies the weights connecting the input gate to the input. The symbol σ refers to the logistic sigmoid method. The forget gate f_t chooses the proportion of the former cell state to discard, the input gate decides how much new information to incorporate into each unit, and the output gate manages the amount of the cell's interior state that is revealed. Because all three gates update their activations at every time step, the network can learn to capture temporal patterns across multiple steps.

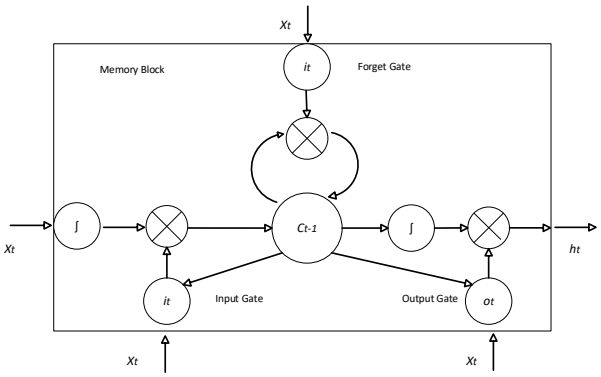

Fig. 1: LSTM unit structure

3.2. Convolutional neural network (CNN)

CNN is an improved deep learning method that is utilized successfully in many categorization tasks (Wang et al., 2017a). The convolution layer is the most essential part of CNN; it utilizes a filter to identify attributes from the input data. Because time series consist of data varying along only one temporal axis, their convolutional kernels are one-dimensional rather than the two-dimensional filters used for images. After convolution, the resulting feature maps are typically fed through nonlinear activation methods like the Rectified Linear Unit (ReLU). By applying these activations, the network injects nonlinearity into the model, qualifying it to capture composite, nonlinear relationships in real-world data. Following these two processes, the new features pass through several active functions and filters. The features created provide useful data that can help with classification or prediction tasks. CNN can benefit from more layers. A pooling layer, for example, is employed to minimize the number of parameters, whereas a dropout layer helps avoid overfitting. A primary advantage of applying CNNs to time series data is that their convolutional layers slide learned filters along the temporal axis, extracting features across every time step. This enables CNN to gain knowledge about the attributes that are constant over time (Li et al., 2020).

4. The proposed CNN-LSTM for Android malware classification

This section explains the suggested novel deep learning architecture for Android malware

detection; the suggested model combines the distinct properties of LSTM and CNN to achieve better performance. Previous research showed that integrating LSTM with CNN for time series classification significantly improves its performance (Karim et al., 2017).

As explained in Fig. 2, the model consists of two convolutional blocks, one LSTM block, and finally the fully connected layer. The architecture of each block is composed of a pair of one-dimensional convolutional (Conv1D) layers, a layer for max-pooling, a batch-normalization layer, and a layer for dropout. The Conv1D layers have 128 filters each, and each filter uses a kernel window size of 10. The activation function used in both layers of Conv1D is the rectified linear unit (ReLU). These activation functions play a crucial role in enhancing the neural networks' expressive power and their capability to approximate between different layers (Yarotsky, 2017). A max-pooling layer with a pool size of two is applied immediately after the Conv1D layer to emphasize the important feature by choosing the highest activation within each two-element window. By retaining only the peak value from each region, this pooling operation downsamples the feature map, shrinking its spatial dimensions and speeding up training by concentrating on the most significant signals (Zheng et al., 2020). The network's hyperparameters were then optimized empirically through iterative trials of different settings to determine the combination that achieved the best performance.

As summarized in Table 1, the proposed CNN–LSTM architecture includes a dropout layer with a rate of 0.20 to mitigate overfitting, following the strategy outlined by Kiranyaz et al. (2021). Dropout operates by randomly deactivating a subset of neurons during training, thereby reducing the likelihood of co-adaptation among neurons. This mechanism effectively excludes the corresponding parameters of inactive units from updates, enhancing the model's generalization ability. Incorporating batch normalization within the convolutional layers helps address the internal covariate shift, which stabilizes the learning process by maintaining activation distributions across mini-batches within a controlled range. This not only prevents exploding gradients but also accelerates convergence during training (Kim and Panda, 2021; Kiranyaz et al., 2021).

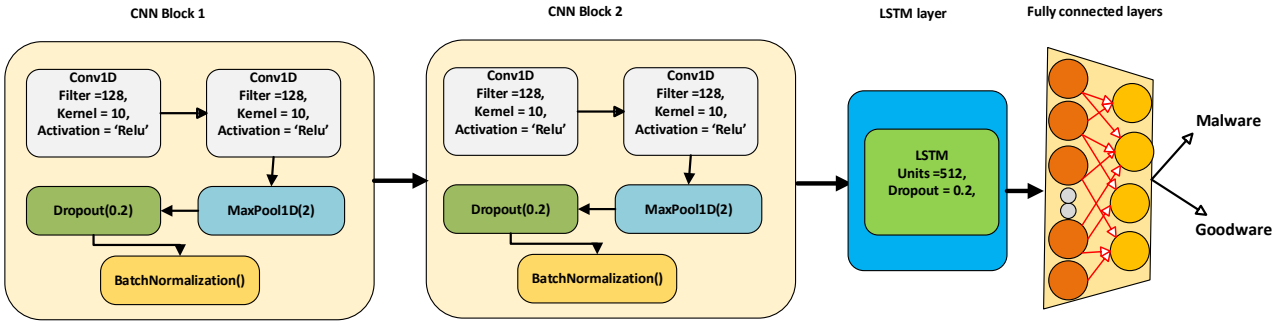

Fig. 2: The suggested scheme for Android malware identification based on CNN-LSTM

Table 1: CNN-LSTM model's parameters

Parameters	Values
Rate of learning	0.001
Number of epochs	500
Size of batch	512
Size of kernel	10
Size of CNN filter	128
Number of LSTM units	512
Decay	
Rate of dropout layer	0.20
Optimization function	Adam

The final classification is achieved through a fully connected (dense) layer, which aggregates attributes extracted by former layers. Prior to this stage, a flattening procedure converts the multi-dimensional attribute maps into a one-dimensional vector suitable for input into the dense layer. The architecture begins with a dense layer of 512 units activated by ReLU, followed by the aforementioned dropout layer. A subsequent dense layer with four neurons, each representing one arrhythmia class, uses a SoftMax activation to output class probabilities.

The training process employs sparse categorical cross-entropy as the loss function, optimized using the Adam algorithm with a learning rate of 0.001 and a weight decay coefficient of 1×10^{-6}. The model is trained for 500 epochs with a batch size of 512, ensuring stable convergence and effective parameter learning.

4.1. Datasets

The First dataset consists of 9476 Android benign apps and 5560 malicious apps. It was developed in Drebin research (Arp et al., 2014), and it contains 49 distinct malware families represented by the Android malware applications: examples of these families include GingerMaster, DroidKungFu, FakeInstaller, and Goldmaster. To prepare for the assessment process, the dataset is pre-processed and divided into testing and training datasets. The dataset uses Android permissions to differentiate between goodware and malicious apps.

The second dataset consists of 2000 malware and goodware apps obtained from the AndroZoo dataset (Allix et al., 2016). AndroZoo is a continuously expanding repository of Android applications gathered from several sources of apps, such as the

official Google Play Store. Over 12M samples are now present in the dataset, which is updated regularly. The features used in the dataset to discriminate between the goodware and malware include the Android permissions and intents.

Data must be formatted correctly for use in machine learning and deep learning algorithms. Some specific Machine Learning and Deep Learning classifiers require data in a specific format; for instance, the Random Forest method does not tolerate null values; hence, one of the most crucial difficulties relating to these concerns is how to address these limitations in data pre-processing.

To preprocess and structure the data effectively, we employed the Python libraries Pandas and NumPy, following standard practices in malware detection studies (Mahdavifar et al., 2022). Both datasets—Drebin and AndroZoo—underwent a series of data cleaning and normalization steps. Specifically, missing values were removed, duplicate entries were eliminated, and Min-Max normalization was applied to scale all features to a uniform range. While the class distributions in both datasets were relatively balanced, stratified sampling was utilized during the train-test split to preserve label proportions and ensure fair evaluation across all classes.

4.2. Performance evaluation metrics

The suggested approach for Android malware classification is evaluated based on several performance assessment measures such as the precision, area under the curve, accuracy, f-measure, and recall.

The proportion of data that was correctly categorized divided by the total number of classification alternatives is the explanation of accuracy. Accuracy can be expressed utilizing Eq. 7:

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \quad (7)$$

Precision is an assessment of how often the machine learning method is precise. Precision may be described using Eq. 8:

$$Precision = TP/(TP + FP) \quad (8)$$

True positive rates are measured using recall. The formula for recall is as Eq. 9:

$$Recall = TP/(TP + FN) \quad (9)$$

Recall and Precision's harmonic average is referred to as F-Measure as Eq. 10:

$$F - measure = ((2 * Precision * Recall)) / ((Precision + Recall)) \quad (10)$$

When coupled, recall and precision (i.e., F-score) are crucial measures employed with balanced and unbalanced data. While recall indicates the ratio of appropriate results, precision reflects the validity of the relevance of the results scale and proximity to the intended answer. The Precision-Recall curve, which is reliable even in an unbalanced data set, provides a full view of the classifier's accuracy (Zhou et al., 2021).

The Receiver Operating Characteristic (ROC) curve's Area Under the Curve (AUC) quantifies a classifier's ability to differentiate between classes across all decision thresholds (Tharwat, 2021). On the ROC plot, the true positive rate, as shown in Eq. 11, and the false positive rate, as shown in Eq. 12, are charted against each other. The AUC determines the two-dimensional area beneath this curve, between (0, 0) and (1, 1). Because it does not rely on choosing a specific cutoff point, AUC is often favored over accuracy as a performance metric. Models with stronger discriminative power achieve AUC scores approaching 1.

$$TPR = TP/(TP + FN) \quad (11)$$
$$FPR = FP/(FP + TN) \quad (12)$$

5. Results and discussion

To evaluate the capability of our proposed model to differentiate between malicious and benign Android apps, we benchmarked it against five established machine learning methods: Decision Tree (DT), KNN, LR, Gradient Boosting (GBoosting), and AdaBoost. All six classifiers were trained and tested on the same two datasets to ensure a consistent and fair comparison of their performance. The proposed model is lightweight enough to be deployed on cloud-based security analysis platforms or edge devices with reasonable computational capabilities.

As shown in Table 2 and Fig. 3, the proposed hybrid CNN–LSTM model significantly outperforms all baseline classifiers on the Drebin dataset. It achieves an accuracy of 98.30%, a precision of 98.84%, a recall of 97.86%, and an F1-score of 97.87%, indicating both high detection capability and robustness in distinguishing between benign and malicious Android applications. These results demonstrate the model's ability to effectively capture complex patterns within static features, leveraging the combined strengths of convolutional and recurrent layers. In contrast, the AdaBoost classifier records the weakest performance on the Drebin dataset, with an accuracy of 91.66%, precision of 91.63%, recall of 91.66%, and F1-score of 91.63%. This suggests that AdaBoost, which relies on sequentially combining weak learners, may not be well-suited to capturing the high-dimensional, non-linear patterns present in Android malware data. A similar performance pattern is observed on the AndroZoo dataset, where the proposed CNN–LSTM approach again demonstrates superior results. It achieves an accuracy of 97.25%, precision of 97.94%, recall of 96.55%, and an F1-score of 97.29%, consistently outperforming traditional machine learning models. The model's strong performance across two distinct datasets highlights

its generalizability and adaptability to real-world malware detection scenarios. Once more, AdaBoost yields the lowest results on the AndroZoo dataset, with an accuracy of 92.68%, precision of 92.88%, recall of 92.24%, and F1-score of 92.51%, as depicted in Table 2 and Fig. 4. These findings further emphasize the effectiveness of deep learning-based architectures in handling complex malware detection tasks and suggest that traditional ensemble methods may fall short in capturing the intricate behavioral characteristics of modern Android malware.

Table 2: Achievement comparison of the suggested scheme and other machine learning algorithms using Drebin and AndroZoo datasets

Dataset	Algorithm	Accuracy	Precision	Recall	F-measure	AUC
Drebin Dataset	DT	95.17	95.22	95.17	95.19	94.80
	LR	93.82	93.89	93.82	93.84	93.60
	KNN	94.62	94.75	94.62	94.56	94.20
	GBoosting	94.92	94.95	94.92	94.93	96.80
	AdaBoost	91.66	91.63	91.66	91.63	92.80
	The proposed approach	98.30	98.84	97.86	97.87	98.60
AndroZoo dataset	DT	95.43	95.45	95.45	95.43	94.80
	LR	92.88	92.92	92.68	92.87	93.60
	KNN	94.37	94.67	94.32	94.37	94.20
	GBoosting	94.79	94.81	94.79	94.79	96.80
	AdaBoost	92.68	92.88	92.24	92.51	92.80
	The proposed approach	97.25	97.94	96.55	97.29	98.9

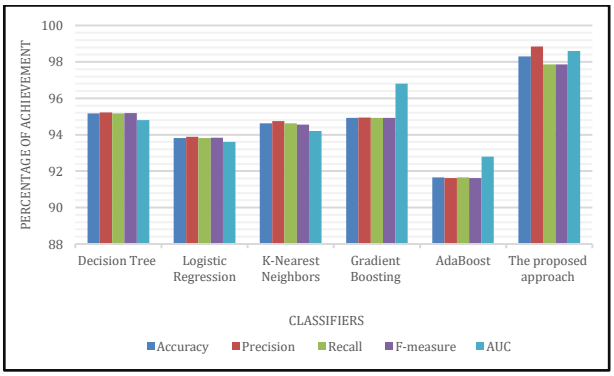

Fig. 3: Performance comparison of suggested scheme and other machine learning algorithms using Drebin dataset

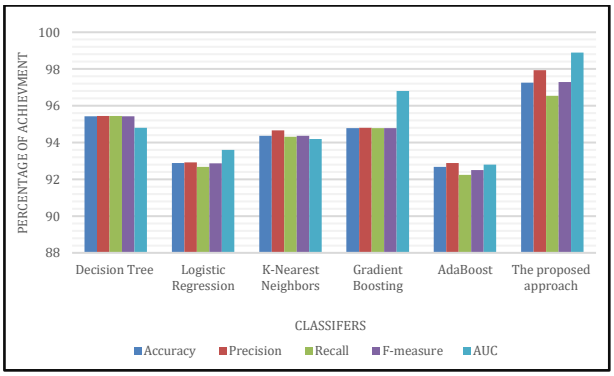

Fig. 4: Performance comparison of suggested scheme and other machine learning algorithms using AndroZoo dataset

Fig. 5 and Fig. 6 compare the area under the curve (AUC) curves of our proposed framework against several baseline machine learning classifiers on two public malware repositories. These plots show the trade-off between true positive rate (sensitivity) and false positive rate (1 – specificity) across varying discrimination thresholds. Consistent with the performance metrics reported in Table 2, our method's ROC curves lie markedly closer to the ideal upper-left corner, indicating superior detection capability. Our model achieved an AUC score of 0.986 on the Drebin dataset and 0.989 on the AndroZoo dataset, underscoring its exceptional ability to distinguish malicious applications from benign ones.

Such high AUC values reflect both robust sensitivity and low false alarm rates, confirming that the proposed approach not only accelerates convergence during training but also attains state-of-the-art classification accuracy. Collectively, these outcomes validate the effectiveness and generalizability of our technique across diverse Android app collections. Precision–recall curves are a standard evaluation metric for classifier performance, especially in contexts where class imbalance is a concern. In these plots, recall is mapped along the horizontal axis, while precision (the positive predictive value) is shown on the vertical axis. Fig. 7 and Fig. 8 compare the precision–recall trajectories of our proposed Android-malware detector against those of several benchmark learning algorithms across two distinct datasets.

Since an ideal classifier's curve hugs the plot's top-right corner, demonstrating both high recall and high precision, the closer a method's trace comes to this region, the better its overall balance between identifying true positives and avoiding false alarms. In both datasets, the curve for our approach remains nearest to the top-right boundary, demonstrating its superior ability to distinguish malicious from benign applications. Moreover, the corresponding areas under these precision–recall curves further affirm that our technique consistently outperforms alternative models in terms of both sensitivity and specificity across varying decision thresholds.

To assess its performance against existing techniques, we present a comparative evaluation in Table 3. The proposed CNN–LSTM hybrid architecture delivers a marked improvement in Android malware classification by combining the strengths of both networks: CNN layers efficiently extract the most discriminative features from application data, while LSTM units capture long-term dependencies among those features. This integration enables the model to discern complex patterns and interrelations within Android app characteristics, thereby enhancing its ability to accurately identify malicious software.

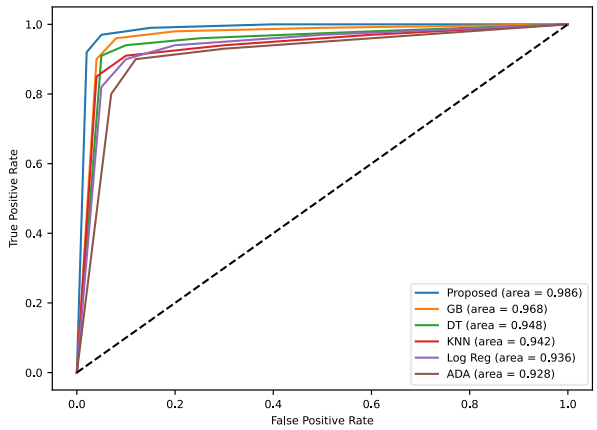

Fig. 5: AUC curve of suggested method and other machine learning techniques using Drebin dataset

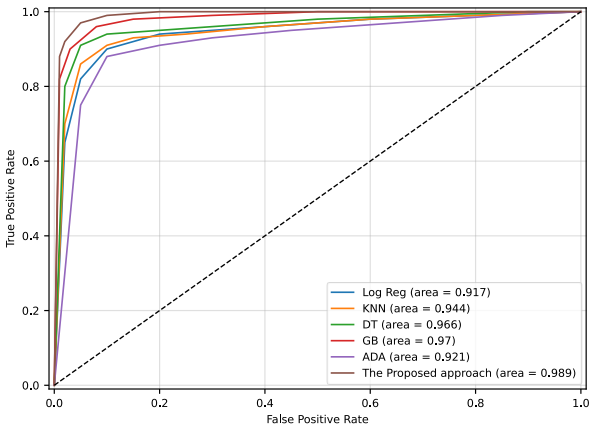

Fig. 6: AUC curve of suggested method and other machine learning classifiers using AndroZoo dataset

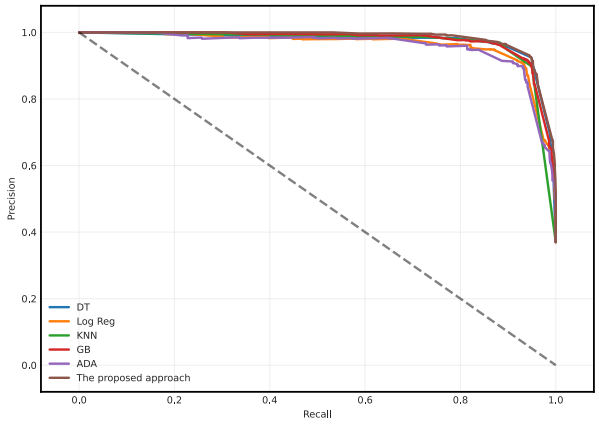

Fig. 7: Precision-recall curve of proposed method and other machine learning techniques using Drebin dataset

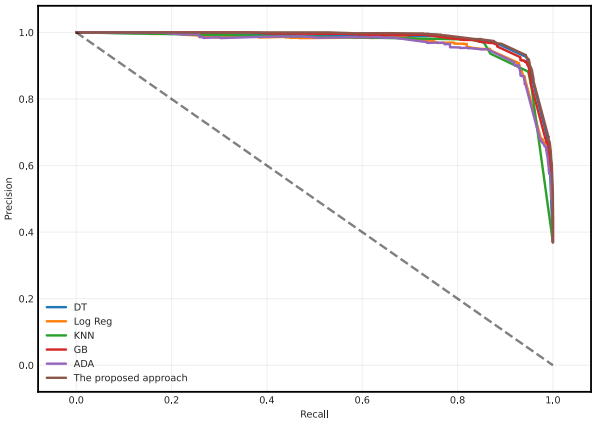

Fig. 8: Precision-recall curve of suggested method and other machine learning classifiers using AndroZoo dataset

Table 3: Comparison of the suggested approach to previous works

Approach	Accuracy	Key Features
Alazab et al. (2020)	94.30%	Integrates Android permissions and API calls, applying three different feature grouping strategies to enhance malware detection and classification accuracy.
Fereidooni et al. (2016)	97%	Leverages multiple static features (e.g., permissions, system commands, and intents) and utilizes XGBoost and deep learning models to detect malware in Android applications.
Wang et al. (2017b)	98.2%	Employs a parallel detection model using eight distinct static feature types, including API calls and network addresses, to identify malicious behavior in Android apps.
Bai et al. (2020)	97.40%	Combines n-gram opcode sequences and permissions, applies dimensionality reduction, and uses CatBoost to achieve robust malware classification.
McLaughlin et al. (2017)	98%	Applies deep CNN to raw opcode sequences derived from disassembled applications to perform effective malware classification through static code analysis.
Lou et al. (2019)	95.32%	Utilizes a topic modeling approach with data flow analysis and permissions to train machine learning models for Android malware detection.
Talha et al. (2015)	88.28%	Implements a permission-based static analysis framework (Apk Auditor) that calculates a maliciousness score based on permission frequency in malware samples.
Altaher (2017)	90%	Uses an evolving hybrid neuro-fuzzy classifier trained on Android permission features to dynamically detect malware.
Altaher and Barukab (2017)	91%	Proposes a hybrid ANFIS model enhanced with fuzzy clustering, using permission-based features for accurate categorization of Android malicious applications.
Taha and Malebary (2021)	94.63%	Combines fuzzy c-means clustering with LightGBM to classify Android malware using permissions as core features; enhances performance via fuzzy logic.
Taha and Barukab (2022)	94.15%	Utilizes ensemble learning integrating multiple base classifiers (e.g., DT, SVM, GBM) with genetic algorithm-based optimization to enhance malware classification based on permissions.
Taha et al. (2021)	95.08%	Employs an ensemble framework with fuzzy integral fusion to combine decisions from multiple classifiers, improving classification robustness using permission-based features.
The proposed approach	98.30%	Introduces a hybrid deep learning framework combining CNN for local feature extraction and LSTM for temporal pattern modeling, applied to permission and intent features from Drebin and AndroZoo datasets.

6. Conclusions

In recent years, Android malware has surged in both prevalence and sophistication, representing a serious risk to users' privacy and information security. Consequently, there is a pressing demand for highly reliable, AI-driven detection mechanisms capable of guarding against these threats. In this paper, we introduce an intelligent hybrid scheme that merges the strengths of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks for more effective Android malware categorization. The CNN module is utilized for automatically obtaining discriminative attributes

from app data, while the LSTM component models the sequential, long-term dependencies inherent in malware behaviors. We assessed our framework based on two publicly available benchmarks: Drebin and AndroZoo, using accuracy, precision, recall, F1-score, and AUC as our primary metrics. The proposed method demonstrates superior performance on the Drebin dataset, achieving an accuracy of 98.30%, a precision of 98.84%, a recall of 97.86%, and an F-measure of 97.87%. On the AndroZoo dataset, our approach consistently yields the highest results, attaining 97.25% accuracy, 97.94% precision, 96.55% recall, and a 97.29% F-measure. For comparison, we also applied a range of traditional machine learning classifiers to the same data, and our hybrid approach consistently outperforms these baselines. For future research, we plan to enrich our system by fusing static analysis features with insights from dynamic behavior profiling to achieve higher malware detection accuracy. Moreover, we plan to extend our framework by integrating attention mechanisms and SHAP (SHapley Additive exPlanations) values to provide meaningful insights into how the model makes decisions, particularly which permissions or features contribute most to malware detection.

List of abbreviations

AdaBoost	Adaptive boosting
AI	Artificial intelligence
ANFIS	Adaptive neuro-fuzzy inference system
API	Application programming interface
APK	Android application package
AUC	Area under the curve
CatBoost	Categorical boosting
CNN	Convolutional neural network
Conv1D	One-dimensional convolutional layer
DT	Decision tree
F-measure	Harmonic mean of precision and recall
FCM	Fuzzy c-means
FN	False negative
FP	False positive
FPR	False positive rate
GBoosting	Gradient boosting
KNN	K-nearest neighbors
LightGBM	Light gradient boosting machine
LR	Logistic regression
LSTM	Long short-term memory
NB	Naive Bayes
ReLU	Rectified linear unit
RF	Random forest
RNN	Recurrent neural network
ROC	Receiver operating characteristic
SHAP	SHapley additive explanations
SVM	Support vector machine
TN	True negative
TP	True positive
TPR	True positive rate
XGBoost	Extreme gradient boosting

Acknowledgment

This project was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, Saudi Arabia, under grant no. (IPP:1027-830-2025). The authors, therefore, acknowledge with thanks DSR for technical and financial support.

Compliance with ethical standards

Conflict of interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

Alazab M, Alazab M, Shalaginov A, Mesleh A, and Awajan A (2020). Intelligent mobile malware detection using permission requests and API calls. Future Generation Computer Systems, 107: 509–521. https://doi.org/10.1016/j.future.2020.02.002

Aldhyani THH and Alkahtani H (2022). Attacks to automatous vehicles: A deep learning algorithm for cybersecurity. Sensors, 22(1): 360. https://doi.org/10.3390/s22010360 **PMid:35009899 PMCid:PMC8749531**

Allix K, Bissyandé TF, Klein J, and Le Traon Y (2016). AndroZoo: Collecting millions of Android apps for the research community. In the Proceedings of the 13th International Conference on Mining Software Repositories, Austin, USA: 468–471. https://doi.org/10.1145/2901739.2903508

Altaher A (2017). An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and permission-based features. Neural Computing and Applications, 28: 4147-4157. https://doi.org/10.1007/s00521-016-2708-7

Altaher A and Barukab O (2017). Android malware classification based on ANFIS with fuzzy c-means clustering using significant application permissions. Turkish Journal of Electrical Engineering and Computer Sciences, 25(3): 2232-2242. https://doi.org/10.3906/elk-1602-107

Arp D, Spreitzenbarth M, Hübner M, Gascon H, and Rieck K (2014). Drebin: Effective and explainable detection of Android malware in your pocket. In the Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, USA: 23-26. https://doi.org/10.14722/ndss.2014.23247

Bai H, Xie N, Di X, and Ye Q (2020). FAMD: A fast multifeature Android malware detection framework, design, and implementation. IEEE Access, 8: 194729–194740. https://doi.org/10.1109/ACCESS.2020.3033026

Enichen EJ, Heydari K, Li B, and Kvedar JC (2025). Platform matters -- Differences in COVID data collected from Android and iOS app users. npj Digital Medicine, 8: 307. https://doi.org/10.1038/s41746-025-01734-8 **PMid:40413335 PMCid:PMC12103540**

Fereidooni H, Conti M, Yao D, and Sperduti A (2016). ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. In the 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), IEEE, Larnaca, Cyprus: 1–5. https://doi.org/10.1109/NTMS.2016.7792435

Graves A, Mohamed AR, and Hinton G (2013). Speech recognition with deep recurrent neural networks. In the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, Vancouver, Canada: 6645–6649. https://doi.org/10.1109/ICASSP.2013.6638947

Hu X (2024). TySA: Enforcing security policies for safeguarding against permission-induced attacks in Android applications. IEEE Access, 12: 165026–165041. https://doi.org/10.1109/ACCESS.2024.3487852

Karim F, Majumdar S, Darabi H, and Chen S (2017). LSTM fully convolutional networks for time series classification. IEEE Access, 6: 1662–1669. https://doi.org/10.1109/ACCESS.2017.2779939

Kim Y and Panda P (2021). Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. Frontiers in Neuroscience, 15: 773954. https://doi.org/10.3389/fnins.2021.773954 **PMid:34955725 PMCid:PMC8695433**

Kiranyaz S, Avci O, Abdeljaber O, Ince T, Gabbouj M, and Inman DJ (2021). 1D convolutional neural networks and applications: A survey. Mechanical Systems and Signal Processing, 151: 107398. https://doi.org/10.1016/j.ymssp.2020.107398

Kumar M, Singh S, Pilania U, Arora G, and Jain M (2023). LSTM-based approach for Android malware detection. Procedia Computer Science, 230: 679–687. https://doi.org/10.1016/j.procs.2023.12.123

LeCun Y, Bengio Y, and Hinton G (2015). Deep learning. Nature, 521: 436–444. https://doi.org/10.1038/nature14539 **PMid:26017442**

Li P, Abdel-Aty M, and Yuan J (2020). Real-time crash risk prediction on arterials based on LSTM-CNN. Accident Analysis & Prevention, 135: 105371. https://doi.org/10.1016/j.aap.2019.105371 **PMid:31783334**

Liu X, Xu L, Xie H, and Prybutok V (2025). An integrated model of smartphone continuance intention: Income effect. Information Systems Management, 42(4): 507-524. https://doi.org/10.1080/10580530.2025.2479733

Lou S, Cheng S, Huang J, and Jiang F (2019). TFDroid: Android malware detection by topics and sensitive data flows using machine learning techniques. In the 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), IEEE, Kahului, USA: 30-36. https://doi.org/10.1109/INFOCT.2019.8711179

Mahdavifar S, Alhadidi D, and Ghorbani AA (2022). Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder. Journal of Network and Systems Management, 30: 22. https://doi.org/10.1007/s10922-021-09634-4

McLaughlin N, Martinez del Rincon J, Kang B, Yerima S, Miller P, Sezer S, Safaei Y, Trickel E, Zhao Z, Doupé A, and Ahn GJ (2017). Deep Android malware detection. In the Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, ACM, Scottsdale, USA: 301–308. https://doi.org/10.1145/3029806.3029823

Prasad A, Chandra S, Uddin M, Al-Shehari T, Alsadhan NA, and Ullah SS (2024). PermGuard: A scalable framework for Android malware detection using permission-to-exploitation mapping. IEEE Access, 12: 50728–50743. https://doi.org/10.1109/ACCESS.2024.3523629

Su X, Zhang D, Li W, and Zhao K (2016). A deep learning approach to Android malware feature learning and detection. In the 2016 IEEE Trustcom/BigDataSE/ISPA, IEEE, Tianjin, China: 244-251. https://doi.org/10.1109/TrustCom.2016.0070

Taha A and Barukab O (2022). Android malware classification using optimized ensemble learning based on genetic algorithms. Sustainability, 14(21): 14406. https://doi.org/10.3390/su142114406

Taha A, Barukab O, and Malebary S (2021). Fuzzy integral-based multi-classifiers ensemble for Android malware classification. Mathematics, 9(22): 2880. https://doi.org/10.3390/math9222880

Taha AA and Malebary SJ (2021). Hybrid classification of Android malware based on fuzzy clustering and the gradient boosting machine. Neural Computing and Applications, 33: 6721–6732. https://doi.org/10.1007/s00521-020-05450-0

Talha KA, Alper DI, and Aydin C (2015). APK Auditor: Permission-based Android malware detection system. Digital Investigation, 13: 1–14. https://doi.org/10.1016/j.diin.2015.01.001

Tang D, Tang L, Shi W, Zhan S, and Yang Q (2021). MF-CNN: A new approach for LDoS attack detection based on multi-feature fusion and CNN. Mobile Networks and Applications, 26: 1705–1722. https://doi.org/10.1007/s11036-019-01506-1

Tharwat A (2021). Classification assessment methods. Applied Computing and Informatics, 17(1): 168–192. https://doi.org/10.1016/j.aci.2018.08.003

Wang X, Zhang D, Su X, and Li W (2017a). Mlifdect: Android malware detection based on parallel machine learning and information fusion. Security and Communication Networks, 2017: 6451260. https://doi.org/10.1155/2017/6451260

Wang Z, Yan W, and Oates T (2017b). Time series classification from scratch with deep neural networks: A strong baseline. In the 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, Anchorage, USA: 1578–1585. https://doi.org/10.1109/IJCNN.2017.7966039

Yarotsky D (2017). Error bounds for approximations with deep ReLU networks. Neural Networks, 94: 103–114. https://doi.org/10.1016/j.neunet.2017.07.002 **PMid:28756334**

Yeboah PN and Baz Musah H (2022). NLP technique for malware detection using 1D CNN fusion model. Security and Communication Networks, 2022: 2957203. https://doi.org/10.1155/2022/2957203

Zheng Z, Chen Z, Hu F, Zhu J, Tang Q, and Liang Y (2020). An automatic diagnosis of arrhythmias using a combination of CNN and LSTM technology. Electronics, 9(1): 121. https://doi.org/10.3390/electronics9010121

Zhou QM, Zhe L, Brooke RJ, Hudson MM, and Yuan Y (2021). A relationship between the incremental values of area under the ROC curve and of area under the precision-recall curve. Diagnostic and Prognostic Research, 5: 13. https://doi.org/10.1186/s41512-021-00102-w **PMid:34261544 PMCid:PMC8278775**

Zhou Y and Jiang X (2012). Dissecting Android malware: Characterization and evolution. In the 2012 IEEE symposium on security and privacy, IEEE, San Francisco, USA: 95–109. https://doi.org/10.1109/SP.2012.16