ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

HYBRID DEEP LEARNING MODEL FOR MALWARE DETECTION IN ANDROID APPLICATIONS

Paramjeet Kaur[1], Dr.Vijay Laxmi[2]
[1]Research Scholar, Department of Computer Applications Guru Kashi University talwandi Sabo(Bathinda)
[2]Professor Faculty of Computer Applications Guru Kashi University talwandi Sabo(Bathinda)

ABSTACT:
With the rapid growth of applications, accurately predicting future traffic patterns is crucial for efficient application management and optimization. This research presents a hybrid deep learning framework that combines CNN (Convolutional Neural Networks) and LSTM (Long Short-Term Memory) networks to enhance the accuracy of network traffic prediction. CNN is leveraged for feature extraction, capturing spatial dependencies within the data, while LSTM is employed to model temporal dependencies, enabling the framework to effectively learn long-term patterns in network traffic. Unlike traditional deep learning models, the proposed hybrid approach improves prediction accuracy by utilizing CNN's capability to extract essential features and LSTM's strength in handling sequential dependencies. The framework significantly outperforms existing deep learning models by addressing the challenges of feature extraction complexity and sequence learning, making it a robust solution for real-time network traffic forecasting. The Kaggle dataset is used to test the suggested model for malware prediction. The accuracy, precision, and recall of the suggested model are evaluated. According to analysis, the suggested model predicts malware with a 98 percent accuracy rate.

Keywords
Malware Detection, LSTM, CNN, PSO

1. Introduction

The use of smartphones has significantly expanded with the advancement of communication technologies. According to the latest data from International Data Corporation, Android's market share achieved 86.6% and 1.3823 billion smartphones were shipped worldwide in the fourth quarter of 2019. Additionally, the report projects that the market share will increase to 87.1% in 2023. Google created the open-source, Linux-based mobile operating system known as Android. Being an open-source operating system has numerous benefits, but as its security approach is reliant on permission tagging, there are certain security issues [1][2]. Many programmers create both official and unofficial apps for Android. Since the generated programs are uploaded to official or additional application repositories without being thoroughly reviewed, it is clear that there is another serious security flaw. Because of the security issues brought on by the big developer community and the operating system, Android is the main focus of malware makers. Unfortunately, because of the popularity of Android, fraudsters have been enticed to embed malware in a number of applications, seriously endangering the security of users. These four categories—hardware-based assaults, kernel-based attacks, hardware abstraction layer-based attacks, and application-based attacks are typically used to evaluate Android malware. These malwares cause a variety of material and non-material harms, including illegal user behavior, enlisting users in botnet attacks, obtaining sensitive user data, generating revenue from data

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

collection, and harming device hardware [3][4]. To prevent these effects, academics and security companies work to develop malware detection systems with high performance rates. Derived from the words “malicious” and “software”, malware is defined as software that gains unauthorized access to a system. Another definition of malware is random code that purposefully alters or destroys the intended functions. Malware includes Trojan horses, spyware, viruses, and other intervening scripts. Malware is also defined as generating massive traffic that contaminates system and data directories, drivers' boot sections, and executable routines, resulting in Denial of Service (DoS) [5][6]. These programs contaminate the memory when the user runs the file and then move on to other directories that are processed later. In operating systems with security flaws, they can take over the system and contaminate other systems on the network. Typically, this type of virus impairs performance. When applying machine learning algorithms to Android malware classification, it is important to balance the efficiency and accuracy of the algorithm. Shallow machine learning algorithms typically create simpler classification models that are easy to implement and fast to run but tend to provide lower accuracy [7][8]. More intricate machine learning models, on the other hand, frequently provide better detection accuracy but have lower efficiency. Many techniques aim to strike a compromise between these two factors. Algorithms including LR (Logistic Regression), Naive Bayes, SVM (Support Vector Machine), kNN (k-Nearest Neighbors), Decision Tree and Random Forest are well-suited for detecting Android malware using shallow machine learning techniques. LR method is a type of generalized linear regression used to estimate the probability of a particular outcome. Its aim is to identify the best model that explains the relationship between a dependent variable and various independent variables. Built on Bayes' Theorem, NB algorithm assumes that the features are conditionally independent of each other, which simplifies classification tasks. It works well in scenarios where this assumption holds true. A supervised learning model called SVM determines the appropriate decision boundary, or hyperplane, to divide the data into distinct classes [9][10]. SVM can also handle non-linear classification through the kernel trick, which allows it to transform data into higher dimensions for better classification. kNN algorithm classifies a sample based on the majority category of its "k" nearest neighbors, assuming that similar data points tend to belong to the same category. This method is based on the principle of proximity in feature space. A non-parametric supervised learning method called a decision tree creates a tree structure from a dataset with labels and features. It is appropriate for both classification and regression problems since it extracts decision rules and uses the tree to categorize or forecast results. An extension of decision trees, Random Forest is an ensemble technique that enhances classification accuracy by combining the predictions of several decision trees [11][12]. The final prediction is based on the majority vote from all trees. In addition to shallow learning techniques, deep learning models show promise for Android malware detection, often providing greater accuracy at the expense of higher computational complexity. DBN (Deep Belief Network) is a probabilistic generative model made up of multiple restricted Boltzmann machines. It learns a joint distribution between observed data and labels through a layer-by-layer training approach, solving the limitations of traditional neural networks when dealing with multi-layer structures [13][14]. CNN is a feedforward neural network with a deep architecture and convolutional layers. In addition to performing shift-invariant classification tasks, such as identifying patterns in photographs or other hierarchical data, it excels at feature representation. One kind of neural network made for

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

sequential data is the recurrent neural network (RNN). It is especially helpful for applications like time-series analysis and speech recognition because it processes data recursively, which enables it to learn intricate, non-linear patterns in sequences. A generator and a discriminator are two competing models that make up the potent unsupervised learning framework known as a GAN (Generative Adversarial Network). While the discriminator tries to discern between produced and genuine data, the generator produces data. Through adversarial training, GANs can produce high-quality outputs from complex distributions. Furthermore, MMML (Multimodal Machine Learning) focuses on integrating and processing data from various sources or modalities. It enables tasks like collaborative learning, representation learning and modality conversion, providing improved flexibility in handling different types of data. Multiple Kernel Learning combines many kernel functions to map and merge features from different sources, enabling more effective data representation in a newly combined space. Graph Embedding techniques solve the problem of efficiently feeding sparse graph data into machine learning models [15][16]. By mapping high-dimensional graph data into low-dimensional dense vectors, this method helps overcome the difficulty of handling sparse and high-dimensional graph structures. Lastly, Representation Learning transforms raw data into a form that machine learning algorithms can effectively process. This method eliminates the need for manual feature extraction, allowing models to automatically learn to use and extract features from the data.

2. Literature Review

Xie Nannan, et al. (2025) presented Andro-BCFL, a novel Android malware detection solution that integrates blockchain and federated learning approaches to improve malware detection speed and efficiency [17]. The framework solved the problem of trust among organizations by utilizing blockchain technology for secure sharing of malware data. Smart contracts played an important role in the system, as they were used to collaboratively manage and update malware databases, facilitating the exchange of malware-related information. For the malware detection process, the system initially employed Locality Sensitive Hashing, which enabled rapid identification and comparison of malware signatures within the blockchain network. This approach helped minimize the impact of malware outbreaks in a timely manner. This two-tiered detection method improved the overall malware identification capabilities of the system. Experimental results demonstrated that Locality Sensitive Hashing achieved a detection accuracy of 93.56% within the blockchain, while Federated Learning boosted the detection accuracy to 97.87%, validating the effectiveness and practicality of the proposed method.

Z. Liu, et al. (2025) suggested a novel approach to address AMD's (Android malware detection) model aging problem [18]. This approach used pseudo-labeled data into the detection model retraining process to lessen the requirement for manual annotation. By identifying their data drift patterns, the strategy offered a novel way to assess the data drift of recently emerged samples. Data drift scores were then generated using these patterns, which aided in deciding how to blend true-labeled and pseudo-labeled data for retraining. This approach maintained the efficacy of malware detection while drastically reducing the resources required for annotation. The efficiency of the suggested models was shown by a number of tests carried out on long-term datasets. The results revealed that the method improved the F-score by approximately 26% when predicting previously unseen malware over a nine-year period.

P. Mishra, et al. (2024) introduced CloudIntellMal, an intelligent malware detection system based

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

on serverless computing, to identify malware that targets Android applications [19]. To store and pre-process logs gathered from end-user devices, this framework made use of popular cloud services including AWS (Amazon Web Services), EC2 (Elastic Cloud Compute), and S3 (Simple Storage Service). It included an optimized feature extraction algorithm that produced feature vectors using the "Bag of n-grams" technique. To find dangerous patterns, the cloud infrastructure used the machine learning technique. The trained decision model was used on EC2 to categorize the apps under observation during the detection phase. Using the Drebin dataset, a framework prototype was created and evaluated, yielding encouraging results.

X. Lu, et al. (2024) presented an Android malware detection model that improved resistance to adversarial attacks by using a denoising graph convolutional network (GCN) and was based on the Android FCG (function call graph) [20]. The model contained techniques for generating vertex feature vectors and simplifying the FCG by decreasing its size. Understanding that adversarial tactics could be used by attackers, the model included a subgraph network (SGN) to reveal the FCG's structural characteristics and gauge the severity of obfuscation attempts. A denoising graph neural network (GNN) was created, and the graph convolution process was subjected to the 1-Lipschitz-based denoising technique. The FCG's feature vectors were extracted by the GCN, and classification was done using an MLP (multilayer perceptron). Experimental results showed that the proposed Android malware detection model outperformed other models in terms of F_1 scores, especially when dealing with many levels of obfuscation attacks, thus validating its effectiveness in combating such adversarial techniques.

L. Huang, et al. (2023) suggested a new malware detection framework, WHGDroid, which was based on a WHG (weighted heterogeneous graph) to detect malware by identifying implicit higher-level semantic connections between Android applications [21]. To thoroughly analyze the applications, the method first extracted five different Android entities and five types of relationships. These entities and their interrelations were then represented in the form of a weighted heterogeneous graph, where the weights reflected the significance of each entity. In order to generate homogenous graphs that solely contained nodes connected to apps, the framework introduced rich-semantic metapaths to implicitly connect the nodes representing apps. To learn the numerical embedding representations of the apps, a graph neural network was used. The method was comprehensively compared with five baseline techniques using large datasets across many reading scenarios. Experimental findings demonstrated that WHGDroid outperformed two leading state-of-the-art methods in every scenario tested.

P. Tarwireyi, et al. (2023) proposed a multi-audio feature-fusion technique to identify Android malware by combining audio features from several viewpoints [22]. This method involved converting Android application package files into waveform audio files in order to extract sixty-three standard audio signal processing features and thirty-nine biologically inspired audio features. MFCC (Mel-Frequency Cepstral Coefficients), GFCC (Gammatone Frequency Cepstral Coefficients), and BFCC (Bark Frequency Cepstral Coefficient) were the sources of the biologically inspired features. Although more research was required, experimental results demonstrated the effectiveness of the audio-based features suggested for malware identification. Using the conventional eXtreme Gradient Boosting (XGBoost) machine learning algorithm on the CICMaldroid 2020 dataset, the suggested approach produced remarkable performance metrics: 98.96% accuracy, 99.65% recall, 99.30% F1-score, and 98.14% AUC.

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

S. K. Smmarwar, et al. (2022) introduced "OEL-AMD" [23], a sophisticated and effective system for detecting Android malware. In order to eliminate superfluous features and efficiently capture statistical traits, this framework made use of statistical feature engineering. Both static and dynamic feature layers were subjected to optimal feature selection using BGWO (Binary Grey Wolf Optimization), a meta-heuristic technique. After that, a number of base learners were trained using adjusted hyperparameters to enhance the ensemble model's capacity for classification via inductive reasoning. The overall performance of the model was then totaled. 83.49% for category classification and 96.95% for binary classification were the greatest classification accuracys attained.

T. Frenklach, et al. (2021) proposed an ASG (app similarity graph)-based static analysis technique for Android apps [24]. The fundamental tenet of this approach was that, as opposed to depending solely on expert-driven features, the secret to categorizing an application's behavior was to find common, reusable building pieces, such as functions. The approach was shown on a fresh VTAz dataset from 2020, a dataset of around 190K apps provided by VirusTotal, and the Drebin benchmark in both balanced and unbalanced configurations. In balanced conditions, the method's accuracy was 0.975, and its AUC score was 0.987. Additionally, the suggested method's analysis and classification time—which ranged from 0.08 to 0.153 seconds per app was noticeably less than those of previous examined studies.

M. K. Alzaylaee, et al. (2020) introduced DL-Droid, a deep learning system that uses stateful input generation and dynamic analysis to identify dangerous Android apps. More than 30,000 programs, both malicious and benign, were used in experiments on actual devices [25]. Additionally, the stateful input generation method's detection performance and code coverage were compared to the stateless approach, which is more frequently employed in deep learning systems. The results showed that DL-Droid outperformed conventional machine learning techniques, with a detection rate of up to 97.8% when utilizing only dynamic features and a rate of 99.6% when combining both dynamic and static data. The findings also emphasized the importance of improved input generation for dynamic analysis, as DL-Droid, with state-based input generation, surpassed existing state-of-the-art detection approaches.

3. Research Methodology

To forecast the malware, a deep learning-based system is created. Compared to current deep learning models, the model framework will be able to predict malware with a high degree of accuracy. The following describes the phases of the suggested model:

3.1. Dataset input and pre-processing

Kaggle is thought to be used to collect data for a dataset, and the pre-processing stage is carried out to remove any missing or inappropriate values.

3.2. Feature Reduction

In order to mitigate the qualities, this step implements the PSO model. The behavior of the particles, such as flocking, swarming, and herding, is taken into consideration when designing this algorithm. Each particle has the ability to alter its course in response to its companion's self or past flying experiences. Due to its own experience, it knows the location of the meal and considers it to be in its personal best position with (P). Furthermore, if a swarm is defined as the global best position (G), the particle is regarded as the finest position [5]. The goal of this method is to recreate this behavior in order to solve real-time issues. Furthermore, the particles receive

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

assistance in creating a swarm. These particles are useful to fly arbitrarily in the solution space at velocity v_i at position x_i and change their places relied on the personal experience, social and cognitive nature. The position and velocity of every particle i at tth generation are defined as:

$$v_i(t+1) = wv_i + c_1r_1\big(P_i(t) - x_i(t)\big) + c_2r_2\big(G(t) - x_i(t)\big) \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

In this, w is employed for inertia weight to regulate the impact of prior velocity; c_1 and c_2 denote the constants, which are utilized to adjust the charm speeds among such social and cognitive elements; and the uniform random values are illustrated with r_1 and r_2 in the range [0, 1].

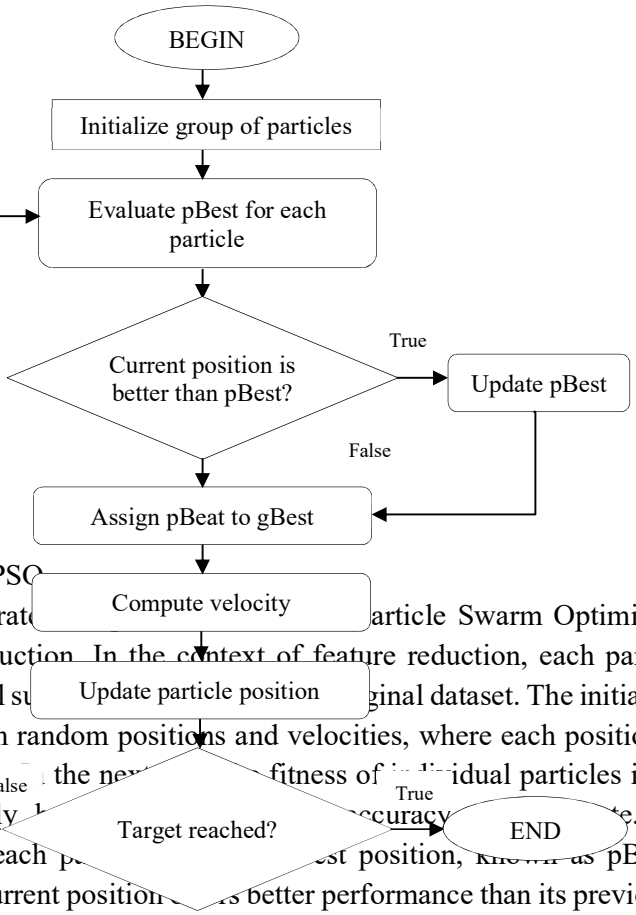


Figure 1: Flowchart of PSO

The flowchart demonstrat article Swarm Optimization) in machine learning for feature reduction. In the context of feature reduction, each particle in the swarm represents a hypothetical su ginal dataset. The initial step is to initialize a group of particles with random positions and velocities, where each position corresponds to a particular set of featu the ne fitness of individual particles is evaluated using a fitness function, usually accuracy e. Specifically, this evaluation determines each p st position, known as pBest. It also checks whether the particle's current position s better performance than its previous pBest. If so, the pBest is updated with the current position. Among all pBest values, the one with the best performance is selected as the global best (gBest), representing the best solution discovered by the entire swarm. The velocity of each particle is influenced by its own pBest, the swarm's gBest, and a random component that promotes exploration of the feature space. These velocities are the mechanism through which the new positions of the particles are obtained, consequently, the feature subsets being considered change. By means of this model, the whole operation is repeated until the condition is met, in which case the particles move towards the most appropriate feature subsets according to the results received. Every iteration, the algorithm verifies that the stopping criterion (reaching a fixed number of iterations, or achieving a desired accuracy) has been met. Once the aim has been achieved, the processing stops, and the best features are selected. This PSO-based technique effectively eliminates redundant information by discarding irrelevant features, thereby enhancing model efficiency and reducing computational costs.

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

3.3. Classification

This phase uses a hybrid deep learning framework that combines CNN and LSTM. A CNN is a large network that repeats and interprets stimuli similarly to how the visual cortex of the brain does. It has several hidden layers and is a deep neural network as well. CNN's output layer frequently uses neural networks for multiclass categorization. CNN uses a feature extractor during the training phase rather than performing it by hand. CNN's feature extractor is composed of distinct neural network types, each of whose weights is established during training. CNN enhances picture identification when its neural network feature extraction is more detailed (has more layers), but at the cost of the learning process complexity that previously made CNN underappreciated and ineffectual. While other neural networks classify features, convolutional neural networks aid in the extraction of the input images' characteristics. This method uses the input image as its starting point. The network traffic is categorized using the feature signals that were recovered. This procedure employed a bi-directional model for classification. The Softmax Regression Layer connects the Forward LSTM and Backward LSTM of Bi-Long Short-Term Memory. Due to the normal LSTM network's lack of future context information and inability to learn all the sequences, the prediction effect is lost when dealing with time series.

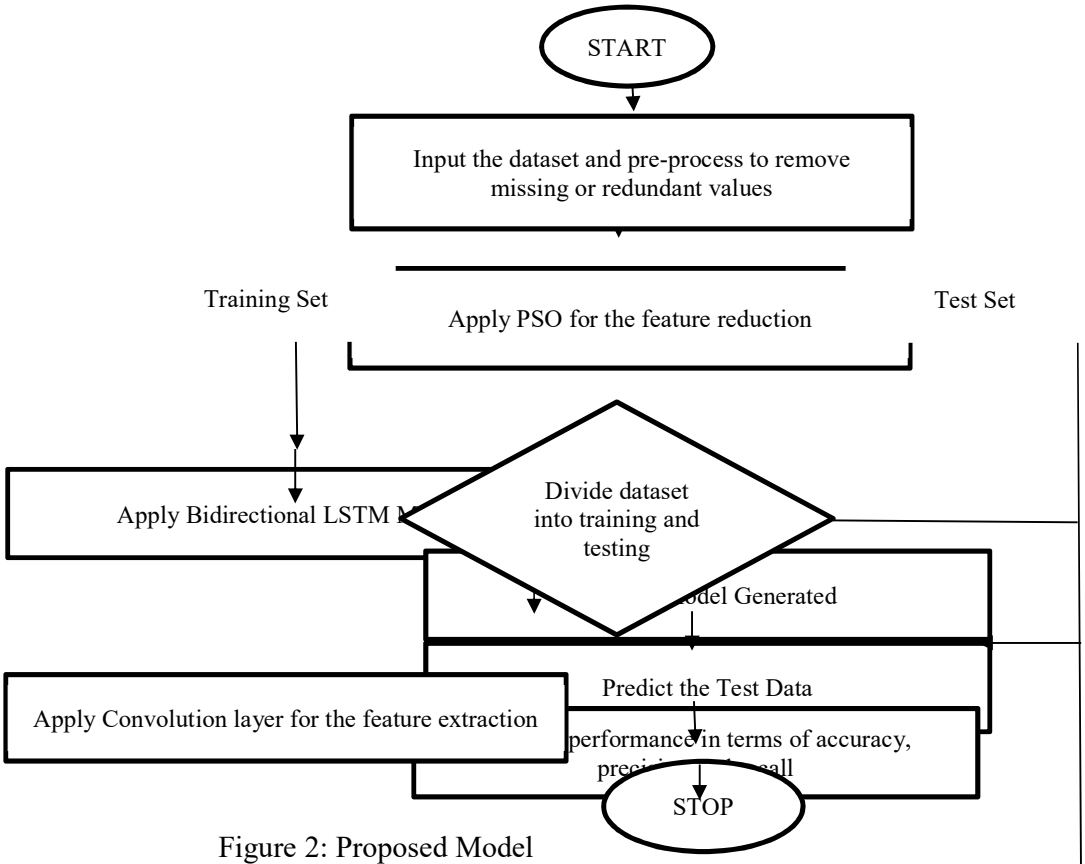


Figure 2: Proposed Model

The process of predicting malware of the proposed model is demonstrated Figure 2. First of all, the data of network traffic are utilized as input in the proposed model and are pre-processed for removing the missing values. After that, the PSO (Particle Swarm Optimization) algorithm is

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

implemented for mitigating the features. The data is split into two sets: training and testing. CNN models are used to extract features from the original collection, while bi-directional models are used afterward. The training system is then developed. The training model is created directly in the latter set. The testing data is then forecasted. Finally, a variety of criteria are used to calculate the performance, including recall, accuracy, and precision. Thus, the outcomes are attained.

4. Result and Discussion

To classify network traffic, the suggested model, which combines CNN and bidirectional LSTM is put into practice. Accuracy, precision, and recall are used to analyze performance using the Kaggle dataset.

4.1. Performance Analysis Parameters

The proposed model is implemented and results of the method is compared in terms of accuracy, precision and recall. The detail description is the parameters is given below: -

Accuracy: The number of correctly classified points divided by the total number of points multiplied by 100 is the definition of accuracy.

$$\text{Accuracy} = \frac{\text{Number of points correctly classified}}{\text{Total Number of points}} * 100 \text{ -(3)}$$

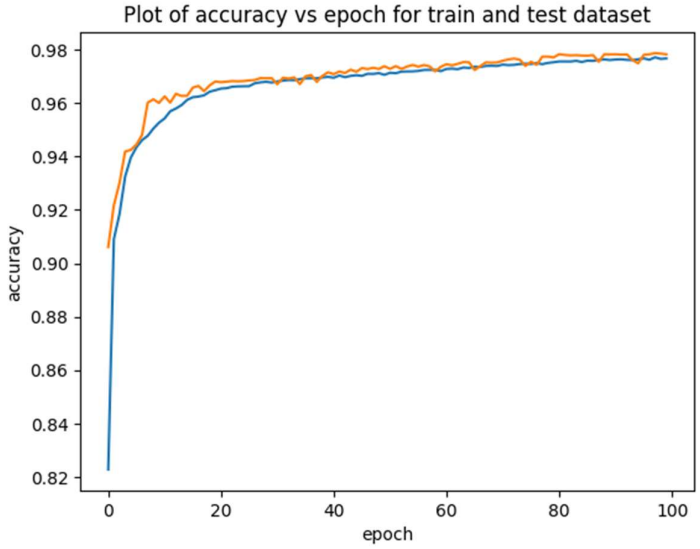
Precision: Precision, also known as positive predictive value, is the percentage of pertinent occurrences among the recovered examples in recognizing patterns, information retrieval, and binary classification.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} - (4)$$

Recall: The percentage of pertinent examples that have been recovered out of all the relevant instances is known as recall.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} - (5)$$

4.3. Results



ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

Figure 3: Accuracy of training and testing

As shown in figure 3, the model training and test accuracy is plotted correspond to different number of epoch values. The model training accuracy is achieved upto 90 percent which represents that model highly trained to generate desired results for prediction.

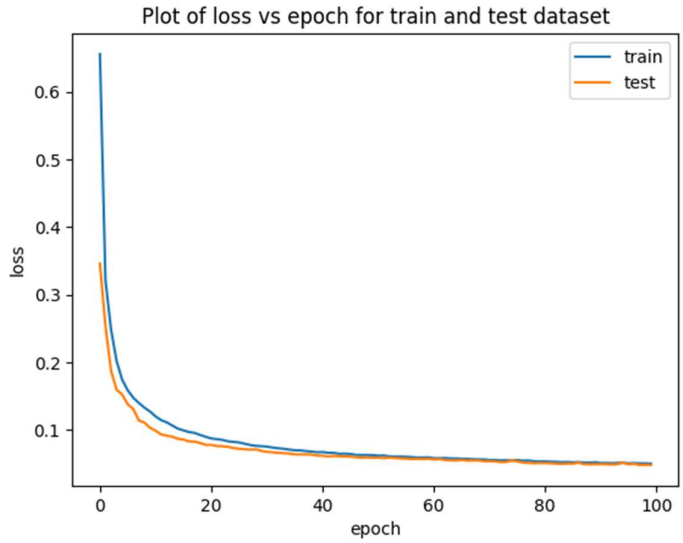


Figure 4: Training and test Model loss

As shown in figure 4, the model training and test loss is representing correspond to number of epoch values. The model loss is reduced to 2 percent for the model training.

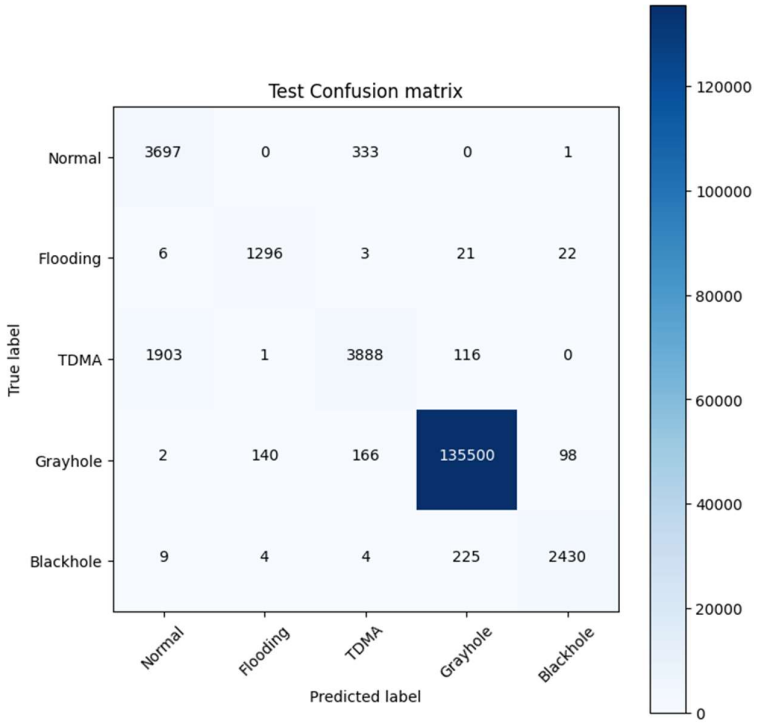


Figure 5: Confusion Matrix

Figure 5 illustrates the confusion matrix between the expected and real labels. Plotting of projected labels against true labels shows the true positive and false positive values.

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

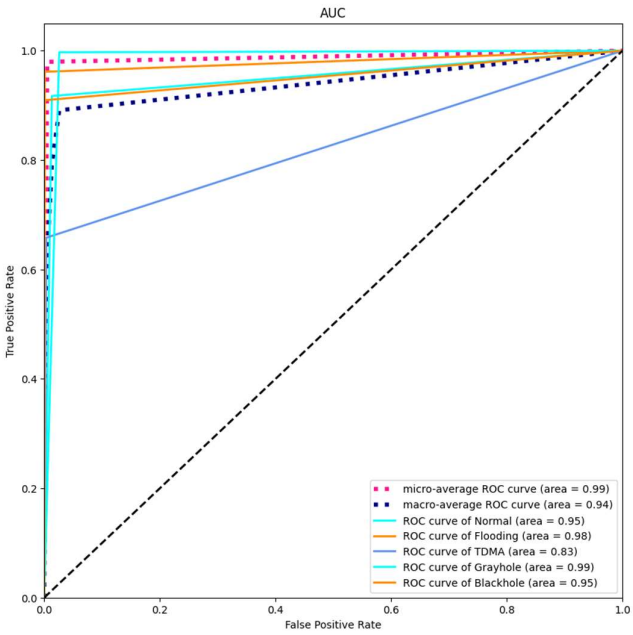


Figure 6: ROC Curve

As shown in figure 6, the AUC-ROC curve is drawn for each class in the dataset. The x axis shows that false positive rate and y axis shows that true positive rate. The ROC curve of each class touch to above 95 percent for the malware detection.

Table 1: Performance Analysis

Models	Accuracy	Precision	Recall
P. Faruki [5]	97.45	97	97
R. B. Hadiprakoso [13]	93.67	93	93
X. Lu, J. Zhao [20]	95.89	95	95
S. K. Smmarwar, [23]	94.12	94	94
Proposed	98	98	98

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

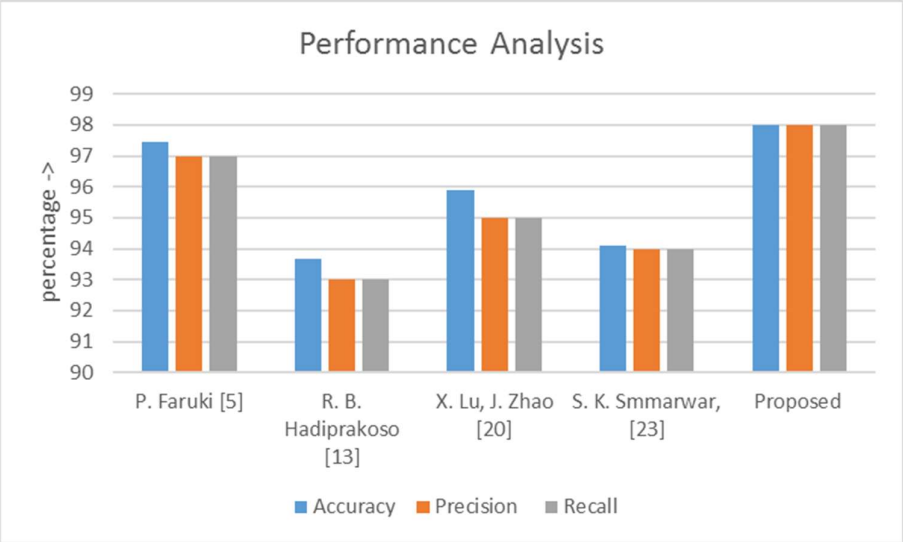


Figure 7: Performance Analysis

As shown in figure 7, the performance analysis of proposed model is compared with existing models. It is analysed that proposed model achieves accuracy of up to 98 percent which is approx. 2 percent higher than existing models.

Conclusion

As malware detection continues to grow rapidly, accurately predicting future patterns is vital for optimizing application management and performance. CNN and LSTM networks are used in this study's hybrid deep learning framework to increase malware prediction accuracy. The CNN component is utilized for feature extraction, capturing spatial relationships within the data, while LSTM effectively models temporal dependencies to identify long-term traffic trends. The suggested method outperforms traditional deep learning models in terms of predicting accuracy by utilizing CNN's capacity to extract important features and LSTM's prowess in managing sequential data. The Kaggle dataset is used to construct the framework, and accuracy, precision, and recall are assessed. According to experimental results, the model outperforms current approaches and reaches an amazing 98% accuracy, proving its usefulness in real-time network traffic forecasting.

References

[1] K. Xu, Y. Li and R. H. Deng, "ICCDetector: ICC-Based Malware Detection on Android," in IEEE Transactions on Information Forensics and Security, vol. 11, no. 6, pp. 1252-1264, June 2016

[2] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma and Z. Liang, "Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 5, pp. 1103-1112, May 2017

[3] Z. Yuan, Y. Lu and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," in Tsinghua Science and Technology, vol. 21, no. 1, pp. 114-123, Feb. 2016

[4] A. Guerra-Manzanares, M. Luckner and H. Bahsi, “Android malware concept drift using system calls: Detection, characterization and challenges”, Expert Systems with Applications, vol. 12, no. 5, pp. 859-867, 21 April 2022

[5] P. Faruki et al., "Android Security: A Survey of Issues, Malware Penetration, and Defenses,"

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

in IEEE Communications Surveys & Tutorials, vol. 17, no. 2, pp. 998-1022, Secondquarter 2015
[6] Q. Han, V. S. Subrahmanian and Y. Xiong, "Android Malware Detection via (Somewhat) Robust Irreversible Feature Transformations," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 3511-3525, 2020
[7] S. H. Moghaddam and M. Abbaspour, "Sensitivity analysis of static features for Android malware detection," 2014 22nd Iranian Conference on Electrical Engineering (ICEE), 2014, pp. 920-924
[8] G. Canbek, S. Sagiroglu and T. Taskaya Temizel, "New Techniques in Profiling Big Datasets for Machine Learning with a Concise Review of Android Mobile Malware Datasets," 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), 2018, pp. 117-121
[9] J. H. Abawajy and A. Kelarev, "Iterative Classifier Fusion System for the Detection of Android Malware," in IEEE Transactions on Big Data, vol. 5, no. 3, pp. 282-292, 1 Sept. 2019
[10] M. Samara and E. -S. M. El-Alfy, "Benchmarking Open-Source Android Malware Detection Tools," 2019 2nd IEEE Middle East and North Africa COMMunications Conference (MENACOMM), 2019, pp. 1-6,
[11] Y. Xue et al., "Auditing Anti-Malware Tools by Evolving Android Malware and Dynamic Loading Technique," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 7, pp. 1529-1544, July 2017
[12] Y. Zhang et al., "Familial Clustering for Weakly-Labeled Android Malware Using Hybrid Representation Learning," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 3401-3414, 2020
[13] R. B. Hadiprakoso, I. K. S. Buana and Y. R. Pramadi, "Android Malware Detection Using Hybrid-Based Analysis & Deep Neural Network," 2020 3rd International Conference on Information and Communications Technology (ICOIACT), 2020, pp. 252-256
[14] S. Liang and X. Du, "Permission-combination-based scheme for Android mobile malware detection," 2014 IEEE International Conference on Communications (ICC), 2014, pp. 2301-2306
[15] E. C. Bayazit, O. Koray Sahingoz and B. Dogan, "Malware Detection in Android Systems with Traditional Machine Learning Models: A Survey," 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), 2020, pp. 1-8
[16] J. Wu and A. Kanai, "Utilizing obfuscation information in deep learning-based Android malware detection," 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), 2021, pp. 1321-1326
[17] Xie Nannan, Mu Linyang, Wang Yangfan, and M. Yubo, “Andro-BCFL: Blockchain and federated learning-based Android malware detection,” Computers & Electrical Engineering, vol. 122, pp. 109948–109948, Dec. 2025, doi: https://doi.org/10.1016/j.compeleceng.2024.109948.
[18] Z. Liu et al., “LDCDroid: Learning data drift characteristics for handling the model aging problem in Android malware detection,” Computers & Security, vol. 150, p. 104294, Mar. 2025, doi: https://doi.org/10.1016/j.cose.2024.104294.
[19] P. Mishra et al., “CloudIntellMal: An advanced cloud based intelligent malware detection framework to analyze android applications,” Computers & Electrical Engineering, vol. 119, pp. 109483–109483, Oct. 2024, doi: https://doi.org/10.1016/j.compeleceng.2024.109483.
[20] X. Lu, J. Zhao, S. Zhu, and P. Lio, “SNDGCN: Robust Android malware detection based on

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 19 No. 01 (2025)

subgraph network and denoising GCN network," Expert Systems with Applications, vol. 250, pp. 123922–123922, Apr. 2024, doi: https://doi.org/10.1016/j.eswa.2024.123922.
[21] L. Huang, J. Xue, Y. Wang, Z. Liu, J. Chen, and Z. Kong, "WHGDroid: Effective android malware detection based on weighted heterogeneous graph," Journal of Information Security and Applications, vol. 77, pp. 103556–103556, Jul. 2023, doi: https://doi.org/10.1016/j.jisa.2023.103556.
[22] P. Tarwireyi, A. Terzoli, and M. O. Adigun, "Using Multi-Audio Feature Fusion for Android Malware Detection," Computers & Security, p. 103282, May 2023, doi: https://doi.org/10.1016/j.cose.2023.103282.
[23] S. K. Smmarwar, G. P. Gupta, S. Kumar, and P. Kumar, "An optimized and efficient android malware detection framework for future sustainable computing," Sustainable Energy Technologies and Assessments, vol. 54, p. 102852, Dec. 2022, doi: https://doi.org/10.1016/j.seta.2022.102852.
[24] T. Frenklach, D. Cohen, A. Shabtai, and R. Puzis, "Android malware detection via an app similarity graph," Computers & Security, vol. 109, p. 102386, Oct. 2021, doi: https://doi.org/10.1016/j.cose.2021.102386.
[25] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based android malware detection using real devices," Computers & Security, vol. 89, p. 101663, Feb. 2020, doi: https://doi.org/10.1016/j.cose.2019.101663.