MLSTM: A GraphSAGE-based Android Malware Detection Method Integrating Mean Aggregation and LSTM

Liu Yi[1, 2], Md Gapar Md Johar[3*], Jacquline Tham[2]
[1]Jiujiang Polytechnic University of Science and Technology, Jiujiang 332020 , China
[2]School of Graduate Studies, Management and Science University, Shah Alam 40100, Malaysia.
[3]Software Engineering and Digital Innovation Centre, Management and Science University, Shah Alam 40100, Malaysia
E-mail: mdgapar@126.com
[*]Corresponding author

Keywords: Android malware detection, mean aggregator, LSTM, security, GraphSAGE

Received: August 15, 2025

With the popularity of smartphones and mobile applications, the threat of Android malware is increasingly serious. To realize the efficient and accurate detection of Android malware, a detection method combining mean aggregator and long-term short-term memory (MLSTM) has been proposed. This method is based on the graph sample and aggregate framework. Construct an isomorphic graph of an application based on permission requests and third-party library call features. MLSTM first aggregates the average features of adjacent nodes, and then uses long-term short-term memory (LSTM) to process sequence information, generating the final node embeddings for classification. The test results on AndroZoo and VirusShare datasets show that compared with baseline models including average aggregator, LSTM aggregator, max pool aggregator, graph convolution network, and gated recurrent unit (GRU), MLSTM has the smallest average absolute error and root mean square error, which are 3.84 and 6.26, respectively. Its detection performance is the best. In terms of permission features and third-party library features, the accuracy of MLSTM is (98.85 ± 0.12) % and (92.58 ± 0.2) %, respectively, significantly higher than GRU under the same permission features. In addition, under the projected gradient descent attack, the success rate of MLSTM attack is 35.28%. This model exhibits good robustness against adversarial attacks. The proposed method has good detection performance, enhanced robustness and stability. Applied to actual Android devices can improve the security and privacy protection level of user data. This method ensures the enhanced efficiency and stability, and provides a certain reference direction for Android malware detection.

Povzetek: Predlagana metoda MLSTM učinkovito in natančno zaznava Android zlonamerno programsko opremo z uporabo grafnih značilk in LSTM, pri čemer dosega visoko natančnost, robustnost proti napadom ter izboljšuje varnost uporabniških podatkov.

1 Introduction

With the continuous development of the Android platform and the expansion of the application field, the threat of malware will also increase day by day. Therefore, the research and application prospect of Android malware detection (AMD) is still very broad. Homogenous graph means that the relationships between all nodes in the graph are the same, that is, the connection mode and properties are the same between nodes. The study of homogeneous graphs focuses on similarities and commonness between nodes and is often used in social network analysis, bioinformatics, etc. [1]. Node aggregation is the merging of a set of nodes in a homogeneous graph into one hypernode to reduce the size and complexity of the graph. And GraphSAGE (Graph Sample and Aggregating) is a representation learning framework used for graph data. The current algorithm has some limitations in the node aggregation process and fails to make full use of the semantic relations and feature information between nodes. Therefore, further improvements of the algorithm to improve the expressive power of node aggregation to better meet the processing challenges of large-scale homogeneous graphs. Long and short-term memory (LSTM) is widely used in language models, machine translation, text classification, emotion analysis, named entity recognition and other tasks. Its memory unit and gating mechanism enable the LSTM model to effectively capture the long-term dependence of sentence and text sequences [2]. Malware detection systems often face high error rate and false alarm rate. The detection system may misidentify legitimate software as malware, or wrongly judge malware software as legitimate software. This can limit users' legal behavior or let potential malware from detecting [3]. Therefore, the paper studies the analysis and behavior modeling of Android malware features, and proposes a new AMD method combining mean with LSTM (MLSTM) aggregator. The core assumption of this study is to combine the stability and efficiency of mean aggregation with the powerful capture ability of LSTM for

sequence dependencies in the GraphSAGE framework. The MLSTM aggregator constructed can significantly improve the accuracy of node classification and generate models that are more robust than a single aggregator, thus more effectively addressing AMD tasks.

2 Related works

In recent years, malware is an increasing threat to mobile devices with the increasing number of users on Android platforms. Therefore, it is an urgent need to find the efficient and reliable detection methods, and many scholars have conducted relevant research. Yuan et al. developed a lightweight Android malware detector to address the privacy and communication overhead issues in cloud detection. This detector is designed to avoid communication overhead and privacy leakage, and improve detection accuracy and robustness. The results indicate that the detection accuracy of the detector exceeds 90%, approaching that of multilayer perceptron and convolutional neural network (CNN) [4]. In order to detect the malware of Android application, Mahindru and Sangal built the AMD framework through machine learning technology, so as to protect the privacy of users and the integrity of the system, and used the features in the feature selection method to complete the training. The results show that the detection rate of the framework is 98.8% [5]. Yang's team has improved their ability to detect new Android malware and proposed a novel AMD method with application program interface (API) semantics extraction (AMDASE). This method utilizes techniques such as API semantic clustering, call graph optimization, and abstraction of function calls. The results showed that its detection accuracy reached over 90%, and the aging rate was slower [6]. In order to reduce the dependence of feature learning on prior knowledge, Zhu and other researchers built a deep learning malicious software detection framework combined with the denoising autoencoder to learn rich features and improve the detection performance. The results show that the highest accuracy of this framework is 94.46% [7]. For AMD, Mahindru and Sangal conceived a framework combined with unsupervised machine learning to improve detection efficiency, and used different feature sorting methods to select feature measurement performance parameters and implement self-organization mapping algorithm. The results show that the detection rate of this framework reached 98.7% [8]. In order to avoid malware threats to device security, professionals such as Zhu et al., implement principal component analysis of feature subsets through the stack integration framework of bootstrapping sample technology and learn the implied supplementary information, which show that the average accuracy of this method is 94.92% [9].

Many researchers have studied LSTM in depth and proposed various improvement methods to further improve their prediction performance and applicability to function in more fields. In order to predict the individual bidding of competitors in the power market, Dhanasekaran's team developed a lightweight deep learning model that combines CNN and LSTM to enhance the detection and classification capabilities of malicious software in 5G network environments. The model was validated on the Malimg dataset, achieving an accuracy of 99.8% and improving classification performance by over 12% compared to existing models [10]. Priyatno and other scholars have solved the problem of detecting malicious software in encrypted traffic by proposing a deep learning model that combines one-dimensional CNN and LSTM to analyze API call sequences. The model achieved a detection accuracy of 99.2% on a benchmark dataset, outperforming existing advanced models [11]. To improve the accuracy of malware analysis, Thakur et al. proposed a hybrid model combining CNN and LSTM (converting malware binary files into grayscale images and using Principal Component Analysis for feature dimensionality reduction and voting classification), which was validated on public datasets to effectively reduce manual analysis resource consumption and outperform existing methods with high-precision metrics [12]. Aslam and other professionals use LSTM to detect Android malware (i.e. AMalLSTM framework) and analyze static features such as API calls and permissions to prevent network attacks. The results showed that its detection accuracy reached 98.4%, precision rate was 98.5%, recall rate was 97.2%, and F1 value was 97.8% [13]. Alsaedi et al. improved the accuracy of malware classification by combining AlexNet model to extract image features and LSTM network for classification. They achieved excellent performance with training accuracy of 99.80% and testing accuracy of 99.49% on the MaliMG dataset [14]. Lingayya and other scholars have improved the accuracy and efficiency of AMD by proposing a novel hybrid dual path bidirectional long short-term memory Kepler dynamic graph convolutional network model, combined with adaptive aphid ant optimization algorithm for feature selection. The detection accuracy on four benchmark datasets reached 99.2%, 99.1%, 99.8%, and 99.8%, respectively, and significantly reduced computation time [15].

In conclusion, the current AMD methods and LSTM have both made some progress in improving the detection efficiency and accuracy. However, the information loss in the GraphSAGE framework has not been improved. Therefore, in order to realize the efficient and accurate detection of Android malware, the analysis and behavior modeling of Android malware features are studied, and an AMD method combining mean aggregator and LSTM is proposed. The comparison of different methods is shown in Table 1.

Table 1: Comparison of different methods

Method	Feature Types	Dataset Used	Performance	Identified Limitations
Yuan et al. [4]	Lightweight features (not specified)	A benchmark dataset	Accuracy > 90%	Weak reproducibility
Mahindru et al. [5]	Features selected via feature selection methods	A benchmark dataset	Detection Rate: 98.8%	Dependent feature selection
Yang et al. [6]	API semantics, Optimized call graphs, Functional call pairs	A benchmark dataset	Accuracy > 90%	The scalability and efficiency of the call graph optimization process for large-scale applications may require further validation.
Zhu et al. [7]	Features learned by denoising autoencoder	A benchmark dataset	Highest Accuracy: 94.46%	Unable to effectively capture temporal dependencies of behavioral characteristics
Mahindru et al. [8]	Unsupervised feature ranking	A benchmark dataset	Detection Rate: 98.7%	Dependent feature selection
Zhu et al. [9]	Principal Component Analysis (PCA) on feature subsets	A benchmark dataset	Average Accuracy: 94.92%	High computational cost and insufficient modeling of complex relationships
Dhanasekaran et al. [10]	Image features (from Malimg dataset)	Malimg	Accuracy: 99.8%	Not specifically targeting behavior graph structure analysis
Priyatno et al. [11]	API call sequences	A benchmark dataset	Accuracy: 99.2%	Ignore the topological relationships between nodes
Thakur et al. [12]	Grayscale image features	Public datasets	High-precision metrics	The semantic information of the original code is lost during the image conversion process
Aslam et al. [13]	Static features (API calls, permissions)	Not specified	Acc: 98.4%, Pre: 98.5%, Rec: 97.2%, F1: 97.8%	Unresolved problem of data information loss in graph structures
Alsaedi et al. [14]	Image features (extracted by AlexNet)	Malimg	Test Accuracy: 99.49%	Weak semantic representation of graph structure in Android application code
Lingayya et al. [15]	Features selected by Adaptive Aphid Ant Optimization	Four benchmark datasets	Up to 99.8%	Additional computational overhead without considering information loss issues

3 AMD model based on MLSTM

The pre-processing and feature extraction of Android malware data are studied, and the AMD model of MLSTM is built by second-order neighborhood aggregation is used.

3.1 Data pre-processing and feature extraction of Android malware

Android malware refers to malicious programs or applications aimed at the Android operating system, aiming to cause damage or illegally gain benefits to users' devices, data and privacy. It is generally for viruses, Trojan horses, adware, spyware and other forms. Android malware typically requests excessive permissions to get sensitive information about users or to perform malicious actions. These permissions may include accessing contacts, making calls, obtaining location information, and other [16]. But Android malware can cost a lot of system resources, such as CPU, memory and battery, and abnormal behavior can be found by monitoring resource usage. Therefore, the data characteristics of Android malware samples involve permission request, behavior mode, code characteristics, network communication and resource occupation, etc. Studying these characteristics can provide a basis for the construction of effective malware detection methods. The common feature extraction methods in malware detection include both static and dynamic [17]. To complete data processing outside the image, GraphSAGE is used to gradually propagate and integrate the feature information of the nodes to obtain a richer and higher-level representation. GraphSAGE Model schematically, as shown in Figure 1.

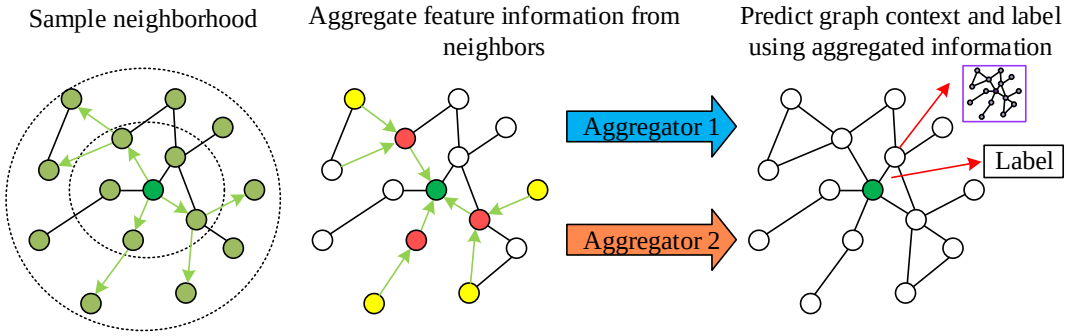


Figure 1: GraphSAGE model diagram

In Figure 1, GraphSAGE learns a low-dimensional representation of nodes by sampling and aggregating features of neighbor nodes to perform tasks on the graph, such as node classification, link prediction, etc. The core idea is to enrich the representation of the target nodes by sampling the features of the neighbor nodes and aggregating these features onto the target node. It aggregates based on the features of the neighbors' nodes and uses these aggregated features to update the representation of the target nodes. In $t+1$ times, the neighbors of the target node i features converge $m_i^{(t+1)}$, as shown in Eq. (1).

$$m_i^{(t+1)} = \sum_{j \in N(i)} M_k \left(h_i^{(k-1)}, h_j^{(k-1)}, e_{ij} \right) \quad (1)$$

In Eq. (1), $j \in N(i)$ is the neighbor of the target node i. In $t+1$ times, the neighbor feature update process $h_i^{(t+1)}$ of the target node i is shown in Eq. (2).

$$h_i^{(t+1)} = U_t \left(h_i^{(t)}, m_i^{(t+1)} \right) \quad (2)$$

The model of node aggregation algorithm can capture the higher level of semantic information through the neighborhood information between the nodes, so as to improve the model robustness. However, in the process of node aggregation, the current algorithm cannot make full use of the semantic relations and feature information between nodes. Therefore, to improve the expression ability of node aggregation, homogeneous maps of third-party library calling features and authority features are constructed. The construction elements of homogeneous graphs in this study are defined as follows: nodes correspond to a single Android application package. If two nodes share at least one specific third-party library, or if there is at least one common request in a sensitive permission set, an undirected edge is established between the two to characterize the similarity of the application at the functional/behavioral level. Based on the graph convolutional neural network, each Android software node is learned combined with the mean aggregator of LSTM, and then classifies the feature vectors in the classifier. Given the high dimensionality of the original API features, this study used chi square test to reduce dimensionality and screen the features most relevant to the category of malicious software. Firstly, extract all APIs from the classs.dex file of the sample to construct an initial high-dimensional set. Then, calculate the chi square statistics of each API and the sample label, and sort all APIs in descending order of chi square values. Finally, select the top 5000 APIs with the highest chi square values as the most discriminative features [18]. The flow of feature extraction is schematic, as shown in Figure 2.

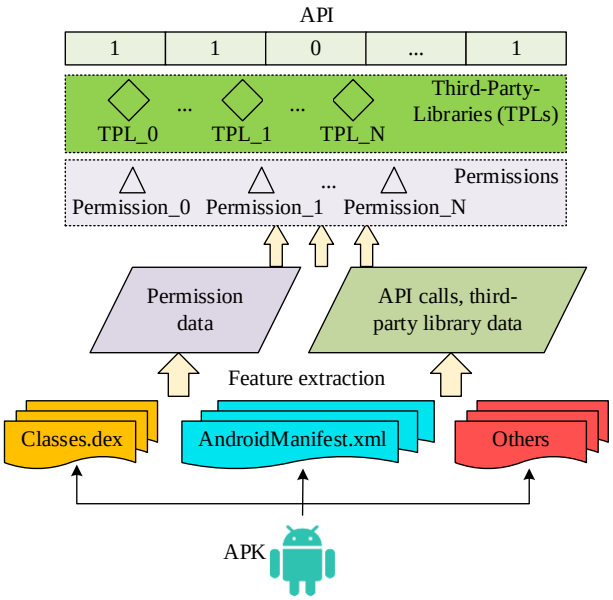


Figure 2: Schematic diagram of feature extraction process

In Figure 2, the Android software internal file Classes. The dex is able to complete API calls and third-party library data, AndroidManifest. Complete xml for permission feature extraction. The original embedding of software nodes in the homogeneous graph is API characteristics, and the establishment of connections between software nodes is completed by third-party library characteristics and permission characteristics.

3.2 Malware detection model based on MLSTM

In GraphSAGE, the goal of the aggregator is to aggregate the features of the neighbor nodes of the target node and generate a new node representation. Mean aggregators are simple and efficient, with small information loss. Where the vector x_v^k expression of mean aggregation updates, as shown in Eq. (3).

$$x_v^k \leftarrow \sigma\left(W \cdot MEAN\left(\left\{ x_v^{k-1} \right\} \cup \left\{ x_u^{k-1}, \forall u \in S(v) \right\} \right) \right) \quad (3)$$

In Eq. (3), the input sample is x, σ is the Sigmoid activation function, x_v^{k-1} is for the current vector, x_u^{k-1} is the sampled neighbor node information, W is the weight vector. However, the mean aggregation cannot handle the long-distance dependence very well. LSTM aggregation can capture the timing information of the nodes, and the working principle is to use the LSTM model to model the feature sequence of the neighbor nodes to obtain a new representation of the target nodes. Therefore, to improve the performance of GraphSAGE on tasks such as node classification and link prediction, the study of a MLSTM aggregator was designed. Due to the unordered nature of the node set obtained from neighborhood sampling, and the need for standard LSTM to process sequential data, a sorting strategy is introduced: before sending the feature matrix of neighboring nodes into LSTM, the L2 norm of their feature vectors is ranked from large to small (assuming that the larger the norm, the richer and more important the information is). Based on this, the unordered node set is mapped into an ordered sequence, allowing LSTM to effectively capture potential dependencies between neighboring nodes. The neighbor feature matrix completes the averaging processing, and after the hidden layer embedding vector changes, it enters the LSTM layer, and finally output the $k-1$ layer vector. The LSTM structure is schematized, as shown in Figure 3.

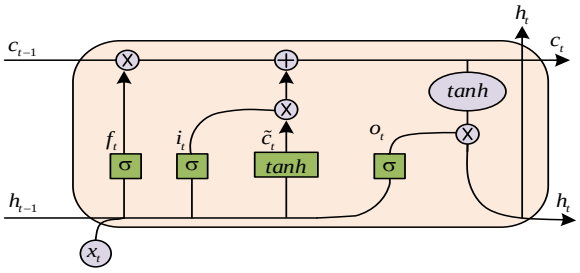


Figure 3: LSTM structural diagram

In Figure 3, LSTM has three gating units, namely input gate, forgetting gate and output gate. The input gate determines which information in the current input needs to be added to the memory state. It takes the hidden state h_{t-1} at time $t-1$ and the input x_t at time t as inputs. After a Sigmoid function and a Tanh function, respectively, we output a value between 0 and 1 and a value between-1 and 1, which represent the proportion and information to be added, respectively. The input gate is i_t, and its calculation formula is shown in Eq. (4).

$$i_t = \sigma\left(W_i\left[h_{t-1}, x_t\right] + b_i\right) \tag{4}$$

In Eq. (4), b_i is the bias of the input gate and the weight matrix of the input gate is W_i. The Tanh function calculates the new candidate vector $\tilde{c}_t$, as shown in Eq. (5).

$$\tilde{c}_t = \mathrm{Tanh}\left(W_c\left[h_{t-1}, x_t\right] + b_c\right) \tag{5}$$

In Eq. (5), b_c is $\tilde{c}_t$ update. The bias, for W_c update. Tanh represents the Tanh function. The weight of the matrix. The forgetting gate determines which information needs to be forgotten in the memory state of the previous moment, and its input h_{t-1} and x_t, after a Sigmoid function, outputs a value between 0 and 1, which represents the proportion of information that needs to be retained or forgotten. The formula f_t is as shown in Eq. (6).

$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right) \tag{6}$$

In Eq. (6), b_f is the bias of the forgetting gate and the weight matrix of the forgetting gate is W_f. The memory element to achieve the state update, update after the result c_t, As shown in Eq. (7).

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{7}$$

In Eq. (7), c_{t-1} is the result at time $t-1$. The value o_t of the output gate in the memory element state is shown in Eq. (8).

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{8}$$

In Eq. (8), W_o and b_o are the weights and biases of the output gate, respectively. The final output h_t, as shown in Eq. (9).

$$h_t = o_t \cdot \mathrm{Tanh}\left(c_t\right) \tag{9}$$

In graph convolutional neural networks, second-order neighborhood aggregation can capture richer context information, stronger expression ability and context perception, which can improve the model performance and robustness [19]. Figure 4 shows the complete workflow of the MLSTM aggregator for implementing second-order neighborhood aggregation. This process is divided into two stages, with the core being that each layer sequentially performs neighborhood sampling, feature aggregation, and node feature updates.

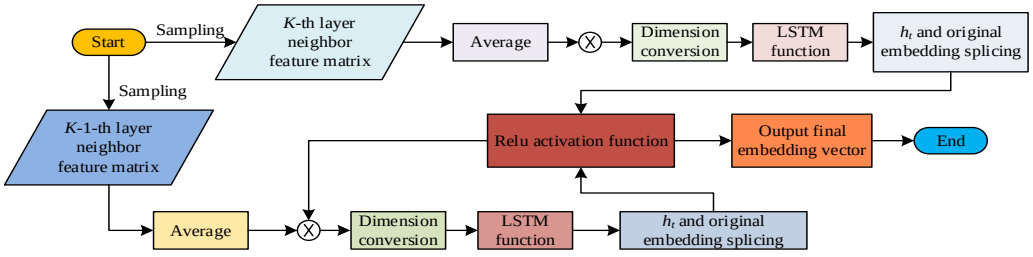


Figure 4: Schematic diagram of MLSTM implementing second-order neighborhood aggregation process

In Figure 4, the first order aggregation of neighborhood sampling and feature matrix construction refers to randomly sampling S_1 nodes from the first order neighbors of the target node υ, stacking the feature vectors of these neighbors in the initial layer to form the first order neighborhood feature matrix $\mathbf{X}^1_{N(\upsilon)}$. MLSTM aggregation is first mean aggregation, followed by LSTM aggregation, and finally feature fusion of the two. Mean aggregation is the process of calculating the average value of matrix $\mathbf{X}^1_{N(\upsilon)}$ along the node dimension to obtain a first-order mean aggregation vector $\mathbf{h}^1_{N(\upsilon),\text{mean}}$. During LSTM aggregation, matrix $\mathbf{X}^1_{N(\upsilon)}$ is sorted in descending order according to the L2 norm of the node eigenvectors and input into the LSTM network. The hidden state of the final time step of LSTM is taken as the first-order sequence aggregation vector $\mathbf{h}^1_{N(\upsilon),\text{LSTM}}$. The vectors $\mathbf{h}^1_{N(\upsilon),\text{mean}}$ and $\mathbf{h}^1_{N(\upsilon),\text{LSTM}}$ are concatenated to form the first-order aggregated neighbor total feature $\mathbf{h}^1_{N(\upsilon)}$, as shown in equation (10).

$$\mathbf{h}^1_{N(\upsilon)} = \text{Concat}\left(\mathbf{h}^1_{N(\upsilon),\text{mean}}, \mathbf{h}^1_{N(\upsilon),\text{LSTM}}\right) \quad (10)$$

Next, update the target node. Specifically, it refers to concatenating the feature $\mathbf{h}^0_\upsilon$ of the target node itself at layer 0 with the first-order aggregated neighbor representation $\mathbf{h}^1_{N(\upsilon)}$. Then, through a fully connected layer (with Relu activation function attached), the new feature representation $\mathbf{h}^1_\upsilon$ of the target node in the first layer is obtained, as shown in equation (11).

$$\mathbf{h}^1_\upsilon = \text{Relu}\left(W^1 \cdot Concat\left(\mathbf{h}^0_\upsilon, \mathbf{h}^1_{N(\upsilon)}\right)\right) \quad (11)$$

In Eq. (11), W^1 is the trainable weight. Relu represents the Relu function. The neighborhood sampling and feature matrix construction of second-order aggregation refers to randomly sampling S_2 nodes (i.e. the second-order neighbors of the target node υ) from each of the S_1 first-order neighbors obtained based on first-order sampling. Average the feature vectors of these second-order neighbors and generate a summarized second-order neighborhood vector for each first-order neighbor. Stack the second-order summary vectors corresponding to all first-order neighbors to form a second-order neighborhood feature matrix $\mathbf{X}^2_{N(\upsilon)}$. Repeat the steps in first-order aggregation to process the matrix $\mathbf{X}^2_{N(\upsilon)}$ and obtain the total representation $\mathbf{h}^2_{N(\upsilon)}$ of the neighbors after second-order aggregation. Concatenate the updated feature $\mathbf{h}^1_\upsilon$ of the target node in the first layer with the second-order aggregated neighbor representation $\mathbf{h}^2_{N(\upsilon)}$, and use a fully connected layer and Relu activation function to obtain the final embedding table $\mathbf{h}^2_\upsilon$ of the target node υ for downstream classification tasks. The MLSTM malware detection model for the study design, is shown in Figure 5.

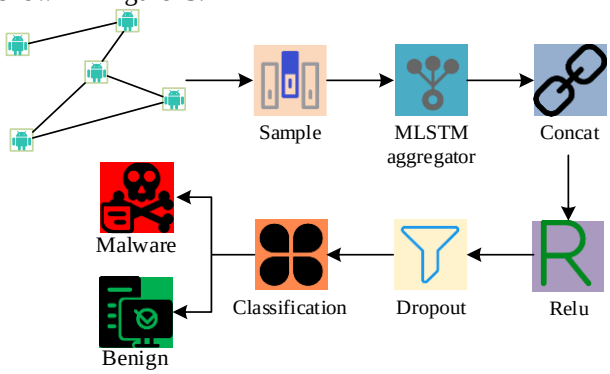


Figure 5: MLSTM malicious software detection model

In Figure 5, the original feature selection software of Android software selects the single feature connection between the software, so as to build the homogeneous graph. Under the GraphSAGE framework, the neighbor node information of the target node is extracted, and the information aggregation is realized after utilizing the improved aggregator. For the output result splicing, the processing is completed in the Relu activation function and the Dropout layer, and then the feature vectors are classified in the classifier, including malware and benign software. For the malware detection model, Fast Gradient Sign Method (FGSM) is an attack method used to generate counter samples. The principle is to use the gradient information to find the direction of the model prediction result in the input sample, and perturb in this direction, so that the model is misjudged. The FGSM formula is expressed, as shown in Eq. (12).

$$\hat{x} = x + \varepsilon sign\left(\nabla_x J\left(f\left(x\right), y\right)\right) \quad (12)$$

In Eq. (12), the size of the disturbance is ε. $J\left(f\left(x\right), y\right)$ is the loss function. ∇_x is calculate the gradient operation and find the maximum gradient d, $d=10$, $d=20$. In the model detection evaluation index, the mean absolute error (MAE) is shown in Eq. (13).

$$MAE = \sum |P - Q| / n \quad (13)$$

In Eq. (13), P is the predicted value, Q is the actual value, and the number of samples is n. Root mean squared error (RMSE), as shown in Eq. (14).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2} \quad (14)$$

In Eq. (14), y_i and $\hat{y}_i$, The predicted and actual values, respectively.

4 Performance analysis of the malware detection method based on MLSTM

The influence of different parameters on the performance of the MLSTM model was analyzed, and the detection accuracy with different aggregators was

compared to verify the robustness of the detection results of the improved model.

4.1 MLSTM model performance analysis

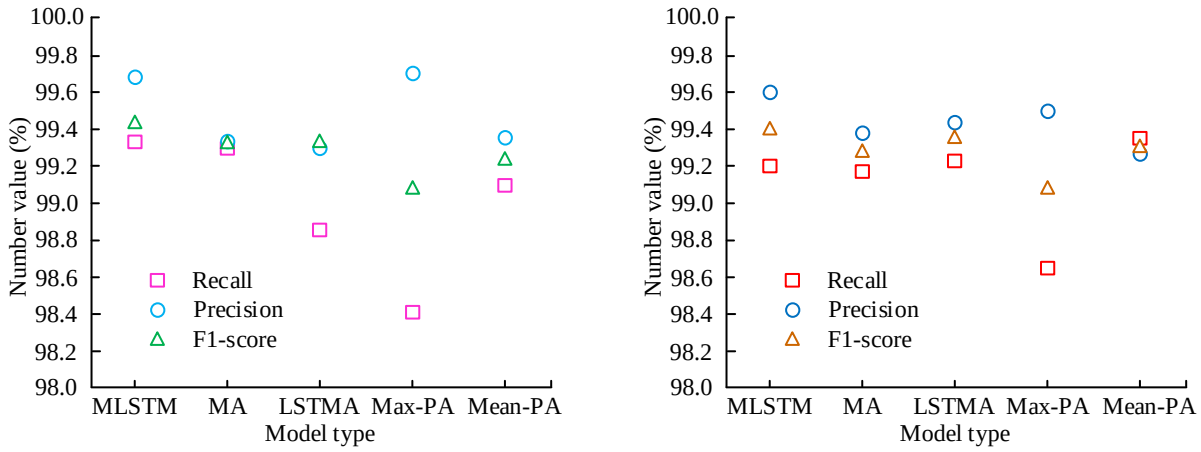
The APK files used in this study were sourced from AndroZoo and VirusShare, covering a sample of 158803 independent Android applications from 2020 to 2023. Among them, 96994 were benign and 61809 were malicious. The top five malicious families are Dropper, Banker, Spyware, Ransomware, and Trojan. According to VirusTotal's report, data labeling is considered benign if there are no engine alarms, and malicious if there are two or more engine alarms. And use stratified sampling to randomly divide the dataset into training and testing sets in an 8:2 ratio. All precision, recall, and F1 scores in this study were calculated using macro averaging. The experimental running platform selects Windows Server 2016 Datacenter, the version is Python 3.7, the code editor uses PyCharm 2019.3.5, the open-source Python machine learning library PyTorch building model, and the optimizer selects Adam. Experimental environment configuration information, as shown in Table 2.

Table 2: Experimental environment configuration information

Hardware and software	Illustrate
CPU	Intel (R) Xeon (R) CPU E5-2678 v3 @ 2.50 GHz
Running Platform	Windows Server 2016 Datacenter
Hard Disk	100 G
Memory	8 G
Deep Learning Framework	PyTorch
Programming Language	Python 3.7, PyCharm 2019.3.5

To ensure the reliability and statistical significance of the experimental results, this study adopted a randomized repeated experiment: each model was trained and tested with 5 different random seeds to eliminate the influence of random factors, and the performance indicators were taken as the mean ± standard deviation of the 5 results; Simultaneously use paired t-test (p<0.05) to verify the statistical significance of the performance differences between MLSTM and other baseline models for key comparisons.

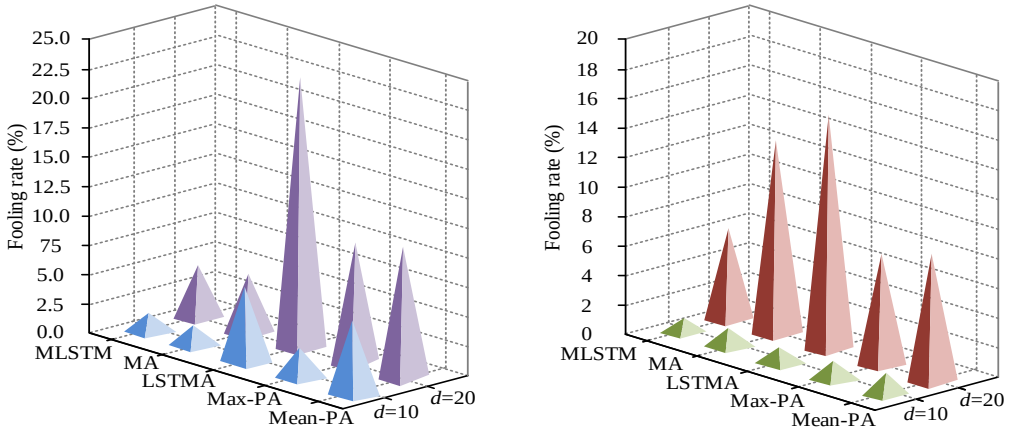
To evaluate the contribution of each component of MLSTM aggregator, this study compared it with four models: Mean Aggregator (MA), LSTM Aggregator (LSTM), Maximum Pooling Aggregator (Max-PA), and Mean Pooling Aggregator (Mean-PA) under the same experimental settings. The MA used in ablation research is an MLSTM architecture without LSTM, while LSTM is an architecture that replaces average aggregation with pure sequential modeling, with all other parameters being the same. The result is shown in Figure 6.



(a) Permission characteristics (b) Third party library features

Figure 6: Performance results of different models under different features

In Figure 6(a), under the permission feature, the recall rate, accuracy rate, and F1-score of MLSTM are (99.18 ± 0.12)%, (99.61 ± 0.08)%, and (99.32 ± 0.09)%, respectively. F1 has a statistically significant improvement compared to MA and LSTMA (p<0.01), indicating that both components are non redundant and MLSTM has the best overall performance. In Figure 6(b), the recall, precision, and F1 score of the MLSTM model are (99.05±0.04)%, (99.69±0.06)%, and (99.31±0.11)%, respectively. The F1-score of MLSTM is (99.31 ± 0.11)%, which still maintains robust performance, while ablation models such as MA and LSTMA are difficult to achieve balance. In summary, ablation studies have confirmed that MLSTM hybrid design combines the advantages of mean and LSTM aggregation, resulting in superior and versatile performance across different features. To evaluate the robustness of the model, the comparison of MLSTM model and MA, LSTMA, Max-PA, Mean-PA models in unknown samples. When $d = 10$ and $d = 20$, the results of different models under permission characteristics and third-party library features are shown in Figure 7.



(a) Permission characteristics (b) Third party library features

Figure 7: The fooling rate results of different models with different d values under permission features and third-party library features

In Figure 7(a), under the permission characteristics, the LSTMA model at $d=10$ and $d=20$ has the largest fool rate, respectively 6.84% and 24.78%. At $d=10$ and $d=20$, the MLSTM model has the smallest fool rate compared to the other four models, respectively, 1.78% and 5.06%. At $d=10$ and $d=20$, the fooling rate of the LSTMA model is approximately 3.85 times and 4.90 times that of the MLSTM model, respectively. In Figure 7(b), under the characteristics of the third-party library, the MLSTM model is 0.97% and 6.40% respectively. When $d=20$, the LSTMA model has the highest fool rate, which is 15.90%.

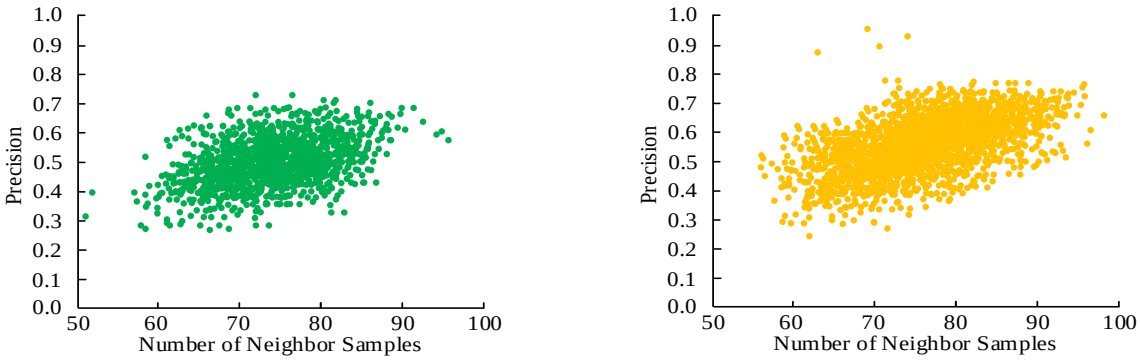
To comprehensively evaluate the adversarial robustness of MLSTM, Projected Gradient Descent (PGD) and Carlini Wagner (CW) attacks (superior to FGSM) were used in white box settings. FGSM is tested with a perturbation size of ε=0.1, PGD is a 10 step iterative attack (step size ε/4, perturbation upper limit ε), and CW is measured using L2 distance (confidence level 0, maximum 1000 iterations). The attack results are shown in Table 3.

Table 3: Attack results of different models

Model	Attack method	Attack success rate (%)	Average L2 perturbation size	Fooling rate (%)	Model accuracy after disturbance (%)
MLSTM	FGSM	19.43 ± 1.05	0.092 ± 0.003	20.11 ± 1.10	79.89 ± 1.10
	PGD	35.28 ± 1.87	0.098 ± 0.004	36.01 ± 1.92	63.99 ± 1.92
	CW	28.95 ± 1.65	1.24 ± 0.15	29.50 ± 1.70	70.50 ± 1.70
LSTMA	FGSM	45.67 ± 1.58	0.095 ± 0.003	47.82 ± 1.63	52.18 ± 1.63
	PGD	68.90 ± 1.45	0.099 ± 0.003	72.15 ± 1.50	27.85 ± 1.50
	CW	60.33 ± 1.72	1.31 ± 0.14	62.88 ± 1.78	37.12 ± 1.78
Max-PA	FGSM	32.10 ± 1.32	0.093 ± 0.003	33.50 ± 1.38	66.50 ± 1.38
	PGD	52.46 ± 1.63	0.097 ± 0.004	54.20 ± 1.68	45.80 ± 1.68
	CW	45.71 ± 1.55	1.28 ± 0.13	47.62 ± 1.60	52.38 ± 1.60
GCN	FGSM	65.88 ± 1.52	0.096 ± 0.003	69.05 ± 1.58	30.95 ± 1.58
	PGD	85.24 ± 1.21	0.100 ± 0.003	89.01 ± 1.15	10.99 ± 1.15
	CW	78.66 ± 1.38	1.35 ± 0.16	82.14 ± 1.32	17.86 ± 1.32

In Table 3, under all attack settings, the MLSTM model showed the lowest attack success rate and fooling rate, and maintained the highest accuracy after perturbation. Under PGD attack, the success rate and spoofing rate of MLSTM attack are 35.28% and 36.01%, respectively, significantly lower than LSTM (68.90%, 72.15%), Max-PA (52.46%, 54.20%), and Graph Convolution Network (GCN) (85.24%, 89.01%). The MLSTM aggregator can enhance its ability to resist adversarial perturbations by fusing neighborhood information. Under PGD and CW attacks, the fooling rate increase of MLSTM is much smaller than that of the comparison model, indicating stronger robustness. Among them, GCN has the weakest robustness, while Max-PA is better than LSTM but not as good as MLSTM, indicating that relying solely on LSTM or Max-PA to deal

with complex adversarial attacks has limitations, and the hybrid design of MLSTM is better. To verify the effect of the number of neighbor samples on the detection accuracy of the MLSTM model, the number of neighbor samples was compared from 50 to 100. The Max-PA model and Detection accuracy of the MLSTM model, is shown in Figure 8.



(a) Max-PA (b) MLSTM

Figure 8: Comparison of detection accuracy between Max-PA model and MLSTM model

In Figure 8(a), Max-PA model MAE is 1.82 and RMSE is 2.26, and when the sampling number is 75, the average accuracy is only 0.47. The model accuracy improved as the number of neighbor samples increased. In Figure 8(b), the MAE of the MLSTM model is 1.64 and the RMSE is 2.08, which is smaller than the Max-PA model, indicating the smaller error and better detection accuracy. The more sample data, the better the model fit and the higher the accuracy. Considering that the composition similarity and the number of sampling layers are one of the influencing factors of the model detection results, the study designed the Euclidean distance similarity, Max-PA model and cosine similarity. The MLSTM model was presented under the permission characteristics and the third-party library characteristics. The prediction results of, as shown in Table 4.

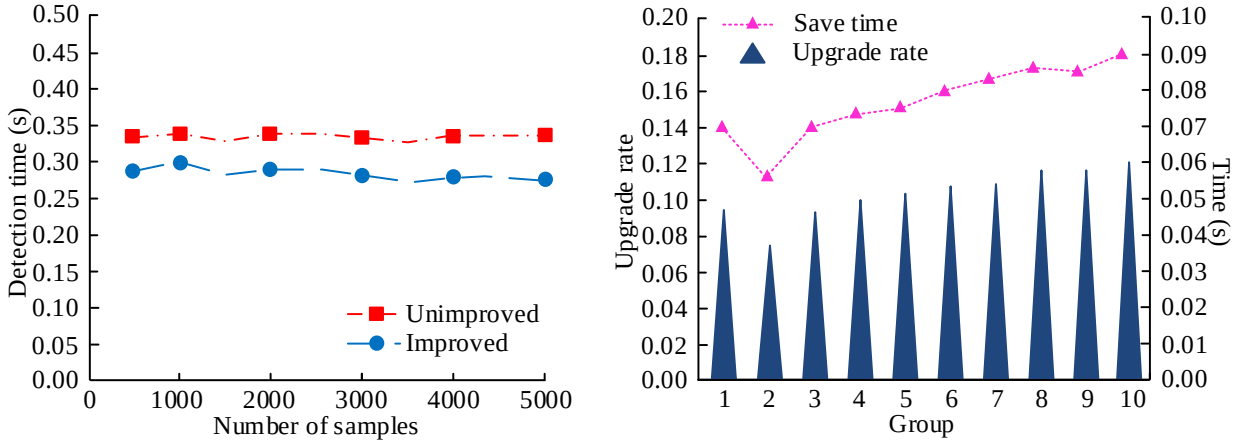
Table 4: Prediction results of different models with different features

Characteristic	Model	Composition similarity		Number of sampling layers		
		Euclidean distance similarity	Cosine similarity	1	2	3
Permission characteristics	Max-PA	0.52	0.50	0.33	0.52	0.41
	MLSTM	0.56	0.53	0.30	0.56	0.35
Third party library features	Max-PA	0.55	0.53	0.35	0.55	0.44
	MLSTM	0.59	0.56	0.36	0.59	0.42

In Table 4, in Authority features and third-party library features under the Max-PA model and of the MLSTM model Euclidean distance similarity detection were higher. The reason is that the Euclidean distance similarity can effectively measure the similarity between features. Before the Euclidean distance, the features are standardized to eliminate the dimensional difference between feature values. The number of sampled layers determines the information about how many layers of neighbor nodes need to be considered during the aggregation process of each node. Max-PA model sum under the third-party library features. If there are too few layers (1 layer), the receptive field is limited and the information is insufficient. If there are too many layers (3 layers), it may introduce too much noise and increase computational complexity, resulting in a slight decrease in performance. When the sampling level is 2, the detection results of Max-PA model and MLSTM model under third-party library features are the highest, at 0.55 and 0.59, respectively.

4.2 Malware detection results of MLSTM

In order to determine the application performance of the designed MLSTM system, the detection time for the multiple quantity of the original system and the improved system, as well as the time saving and improvement rate of the improved system, as shown in Figure 9.

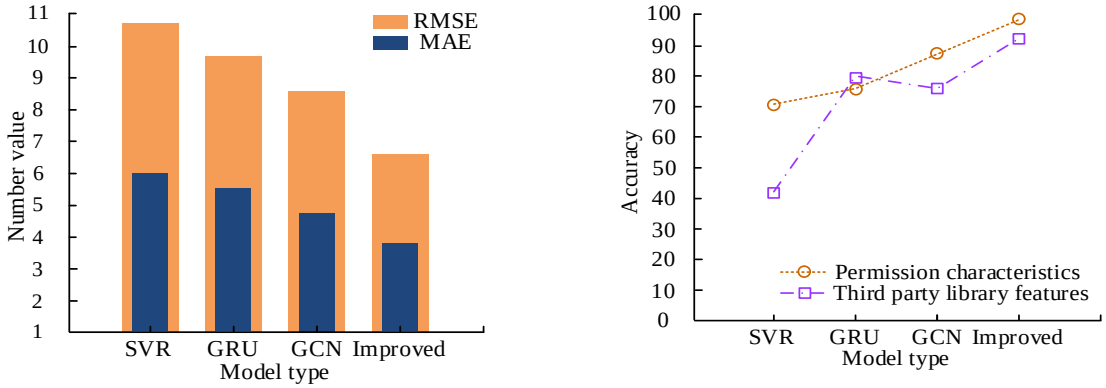


(a) Changes in detection time for different sample sizes

(b) The time saved and spgrade rate of the improved system

Figure 9: Comparison of running time and upgrade rate between improved and unimproved systems

In Figure 9(a), with the increase of the number of samples, the detection time of the system is always in a relatively stable state, and the average detection time of the improved and unimproved systems is respectively 0.274 s, 0.336 s. In Figure 9(b), the average time saving of the improvement system is about 0.05 s regardless of the number of samples. Overall, the improvement rate of malware detection by the improved system increases with the number of samples, with a rate of up to 18.2%. Group 2 has a low improvement rate, which may be because the input data takes a long time in preprocessing. In order to compare the detection performance of the improved model, it was compared with support vector regression (Support Vector Regression, SVR), gating cycle unit (Gated Recurrent Unit, GRU), graph convolutional network (Graph convolution Network, GCN), MAE and RMSE results of different models, as well as the accuracy under permission characteristics and third-party library features, as shown in Figure 10.



(a) Comparison of RMSE and MAE results of different models

(b) Comparison of accuracy results of different models

Figure 10: MAE and RMSE results and accuracy of different models

In Figure 10(a), the MAE and RMSE of the improved model are 3.84 and 6.26, respectively. In contrast, SVR has the highest MAE and RMSE, indicating that it has the worst detection performance and the best detection performance of the improved model. In Figure 10(b), under the permission feature and third-party library feature, the accuracy of the improved model is higher compared to other models, at (98.85 ± 0.12)% and (92.58 ± 0.25)%, respectively, significantly higher than other compared models (all p-values for comparison are<0.05). The feature fusion strategy can effectively improve the performance of the model, indicating that the two types of features are complementary. The accuracy rates under GRU permission features and third-party library features are (75.98 ± 0.10)% and (79.32 ± 0.13)%, respectively. The reason may be that the data quality of the permission feature is poor, including noise or incomplete, which may cause the model to be unable to accurately learn the effective mode, thus reducing the accuracy.

5 Discussion and summary

To improve AMD efficiency, a MLSTM detection method was proposed and the model robustness was enhanced through second-order neighborhood aggregation. The results showed that the recall rate,

accuracy rate, and F1 score under the MLSTM model permission feature were (99.18 ± 0.12)%, (99.61 ± 0.08)%, and (99.32 ± 0.09)%, respectively. The recall rate, accuracy rate, and F1 score under the third-party library feature of MLSTM model are (99.05 ± 0.04)%, (99.69 ± 0.06)%, and (99.31 ± 0.11)%, respectively. Compared with the MLDroid model that relies on manual feature engineering in reference [5], MLSTM automatically captures high-dimensional correlations between features through end-to-end graph learning. MLSTM exhibits good generalization under both permission features (low dimensionality, clear definition, sparse distribution) and third-party library features (semantic richness, contextual relevance). The reason is that it can adaptively learn heterogeneous feature semantics. The mean aggregator can aggregate common information of neighboring nodes and efficiently identify typical combination patterns of permissions. The LSTM component can capture the potential order/dependency relationships of nodes, helping to understand the complex co-occurrence and invocation relationships of third-party libraries. It is not a simple fusion of features, but relies on a hybrid aggregation mechanism to optimize the extraction and fusion of different features, thus exhibiting excellent generalization performance on both types of features mentioned above.

Under PGD attack, the MLSTM fooling rate is 36.01%, significantly lower than LSTM (72.15%). MLSTM is significantly superior to LSTM only aggregators (LSTMA) in terms of fooling rate, due to their different structural capabilities in resisting noise and adversarial perturbations. LSTM only relies on the serialization modeling of neighboring nodes. In adversarial scenarios, small perturbations in node features can be amplified due to LSTM's sensitivity to input order, resulting in hidden state shifts. MLSTM first smooths the input features through mean aggregation (with denoising effect), and then sends the smoothed sequence to LSTM to suppress input fluctuations and make LSTM more stable in capturing long-term dependencies. This two-stage design of smoothing first and then sequence makes it more stable under attacks such as FGSM and PGD compared to a single LSTM. Compared with the ensemble learning based SEDMDynamic model in reference [9], the MLSTM architecture has more efficiency advantages. Compared with the method of converting binary files into images and using the CNN-LSTM hybrid model in reference [12], MLSTM directly processes structured features, avoiding the loss of information in the format conversion process, and better understanding the semantic associations between applications through explicit modeling of the graph structure. The current model is based on static features (permissions, APIs, etc.) for detection and cannot capture the dynamic behavior of malicious software during runtime. In the future, dynamic analysis features can be introduced to construct a multimodal detection model that combines dynamic and static detection, in order to enhance the comprehensiveness of detection.

In summary, the improvement in MLSTM performance is mainly due to its unique hybrid aggregator design, which achieves a better balance between the robustness of information aggregation and semantic representation capability.

References

[1] Dhalaria M, & Gandotra E. Android malware detection techniques: A literature review. Recent Patents on Engineering, 2021, 15(2): 225-245. DOI: 10.2174/1872212114999200710143847.

[2] Kouliaridis V, Barmpatsalou K, Kambourakis G, & Chen S. A survey on mobile malware detection techniques. IEICE Transactions on Information and Systems, 2020, 103(2): 204-211. DOI: 10.1587/transinf.2019ini0003.

[3] Mohammad H A, Husien I M. A deep transfer learning framework for robust IoT attack detection: A review. Informatica, 2024, 48(12): 55-64. DOI: 10.31449/inf.v48i12.5955.

[4] Yuan W, Jiang Y, Li H, & Cai M. A lightweight on-device detection method for android malware. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2019, 51(9): 5600-5611. DOI: 10.1109/TSMC.2019.2958382.

[5] Mahindru A, & Sangal A L. MLDroid - framework for Android malware detection using machine learning techniques. Neural Computing and Applications, 2021, 33(10): 5183-5240. DOI: 10.1007/s00521-020-05309-4.

[6] Yang H , Wang Y , Hu C Z .A novel Android malware detection method with API semantics extraction. Computers & Security, 2024, 137(Feb.):760-771.DOI:10.1016/j.cose.2023.103651.

[7] Zhu H J, Wang L M, Zhong S, Li Y, & Sheng V S. A hybrid deep network framework for android malware detection. IEEE Transactions on Knowledge and Data Engineering, 2021, 34(12): 5558-5570. DOI: 10.1109/TKDE.2021.3067658.

[8] Mahindru A, & Sangal A L. SOMDROID: Android malware detection by artificial neural network trained using unsupervised learning. Evolutionary Intelligence, 2022, 15(1): 407-437. DOI: 10.1007/s12065-020-00518-1.

[9] Zhu H, Li Y, Li R, Li J, You Z, & Song H. SEDMDroid: An enhanced stacking ensemble framework for Android malware detection. IEEE Transactions on Network Science and Engineering, 2020, 8(2): 984-994. DOI: 10.1109/TNSE.2020.2996379.

[10] Dhanasekaran S, Thamaraimanalan T, Vivek Karthick P, & Silambarasan, D. A lightweight CNN with LSTM malware detection architecture for 5G and IoT networks. IETE Journal of Research, 2024, 70(9): 7100-7111. DOI: 10.1080/03772063.2024.2352151.

[11] Priyatno A M, Ningsih Y, Vandika A Y, & Muhammadong, M. A robust hybrid approach for malware detection: Leveraging CNN and LSTM for encrypted traffic analysis. Journal of Engineering and

Science Application, 2024, 1(2): 17-25. DOI: 10.69693/jesa.v1i2.10.

[12] Thakur P, Kansal V, Rishiwal V. Hybrid deep learning approach based on lSTM and CNN for malware detection. Wireless Personal Communications, 2024, 136(3): 1879-1901. DOI: 10.1007/s11277-024-11366-y.

[13] Aslam M, Naeem A, Sarfraz A, Abid, M. K., Aziz, Y., Aslam, N., & Fuzail, M. AMalLSTM: Android malware detection using LSTM. Kashf Journal of Multidisciplinary Research, 2025, 2(03): 61-73. DOI: 10.71146/kjmr333.

[14] Alsaedi B R H. Advanced malware detection: integrating convolutional neural networks with LSTM RNNs for enhanced security. Wasit Journal of Computer and Mathematics Science, 2024, 3(4): 15-31. DOI: 10.31185/wjcms.288.

[15] Lingayya S, Kulkarni P, Salins R D, Uppoor, S., & Gurudas, V. R. Detection and analysis of android malwares using hybrid dual Path bi-LSTM Kepler dynamic graph convolutional network. International Journal of Machine Learning and Cybernetics, 2025, 16(2): 835-853. DOI: 10.1007/s13042-024-02303-3.

[16] AlSobeh A M R, Gaber K, Hammad M M, Nuser, M., & Shatnawi, A. Android malware detection using time-aware machine learning approach. Cluster Computing, 2024, 27(9): 12627-12648. DOI: 10.1007/s10586-024-04484-6.

[17] Dong S, Shu L, Nie S. Android malware detection method based on CNN and DNN bybrid mechanism. IEEE Transactions on Industrial Informatics, 2024, 20(5): 7744-7753. DOI: 10.1109/TII.2024.3363016.

[18] Zhang Z, Li Y, Dong H, Gao H, Jin Y, & Wang W. Spectral-based directed graph network for malware detection. IEEE Transactions on Network Science and Engineering, 2020, 8(2): 957-970. DOI: 10.1109/TNSE.2020.3024557.

[19] Habeeb M A, Khaleel Y L. Enhanced android malware detection through artificial neural networks technique. Mesopotamian Journal of Cyber Security, 2025, 5(1): 62-77. DOI: 10.58496/MJCS/2025/005.