INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

Deep Learning Based Maldroid Stacked Propagate Network for Android Malware Prediction for Security Enhancement

M Anusha[1], M Karthika[2]*

1 Assistant Professor, PG & Research Department of Computer Science, National College (Autonomous), Affiliated to Bharathidasan University, Tamil Nadu, India
2 Research Scholar, PG & Research Department of Computer Science, National College (Autonomous), Affiliated to Bharathidasan University, Tamil Nadu, India

OPEN ACCESS

Received: 27-09-2024

Accepted: 07-11-2024

Published: 10-12-2024

Citation: Anusha M, Karthika M (2024) Deep Learning Based Maldroid Stacked Propagate Network for Android Malware Prediction for Security Enhancement. Indian Journal of Science and Technology 17(45): 4743-4755. https://doi.org/10.17485/IJST/v17i45.3099

* **Corresponding author**.

karthika56498@gmail.com

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: This study aims to address the growing challenge of Android malware obfuscation, which undermines traditional detection methods. We propose the MalDroid Stacked Propagate Network (MDSPN), a deep learning-based system designed to enhance the accuracy and effectiveness of Android malware detection, with a focus on overcoming obfuscation techniques and improving classification performance. **Methods:** The MDSPN framework incorporates advanced data preprocessing techniques, including regressor data imputation to handle missing values and Adam Divisive Clustering to effectively organize and categorize malware features. **Findings:** MDSPN achieved an outstanding 99.5% detection accuracy, surpassing traditional state-of-the-art malware detection techniques by 7.3% in classification accuracy. Additionally, the model demonstrated a 92% precision rate and reduced false positives by 15%, highlighting its superior performance in identifying and classifying malicious Android applications. **Novelty:** This study introduces an innovative approach by combining data imputation, divisive clustering, and deep learning within the MDSPN framework. The integration of these techniques provides a more accurate, computationally efficient alternative to conventional detection methods, advancing the state-of-the-art in Android malware detection and offering a robust solution to combat sophisticated malware obfuscation strategies.

Keywords: Android Malware Detection (AMD); MalDroid stacked propagate network; Androzoo and Drebin Dataset; Malware Classification; Deep Learning

1 Introduction

By 2021, Android will have captured close to 84% of the global smartphone operating system market, making it a dominant player with an extensive user base. However, its open-source nature has rendered it a prime target for sophisticated malware. Recent trends show an increase in automated message abuse and Banking Trojans, illustrating

the growing threat of persistent malware on Android platforms. The need for accurate detection systems has become urgent to safeguard user resources effectively.

Machine learning (ML) and deep learning (DL) methods have shown promise for detecting malicious apps. Techniques such as CNNs, RNNs, and GNNs have been applied to classify Android malware based on various graph structures (e.g., control- flow and data-flow graphs). Despite these advancements, current detection systems face limitations. They struggle with complex malware tactics, such as cooperative data flows and inter-component collusion, which can evade traditional detection. Additionally, most models neglect vital vulnerability data, such as critical system calls associated with published CVEs, which could enhance malware identification accuracy.

Research Gap: While several approaches use structural and temporal features to identify malicious patterns, they often lack an integrated approach that leverages diverse flow types, such as ICCs, and comprehensive vulnerability data. These omissions limit the effectiveness of current models, particularly against malware families that exhibit complex behavioral flows. Therefore, there is a need for a detection system that combines heterogeneous edge types and vulnerability-informed data to capture nuanced malware behavior more accurately.

This study aims to address these gaps by introducing a novel MalDroid Stacked Propagate Network (MDSPN). Our method leverages advanced data imputation, clustering, and a robust graph-based model that integrates heterogeneous flows and critical call sequences, significantly enhancing Android malware detection accuracy and adaptability.

The results of the investigation are summarized below.

(1) We suggest using deep learning to identify malicious apps on Android. The method is based on the current vulnerability reports, namely the call traces that reach crucial APIs. To accomplish fine-grained feature extraction, the optimization-based model considers temporal features leading to the identification of crucial events related to features.

(2) We assess our method's efficacy by contrasting it with well-known deep learning-based detection strategies. The outcomes using datasets prove the efficacy of our technique.

The remaining segments of the research are laid out as follows: Our methodology's outline is presented in Section 2. The results and the discussion part of the datasets are provided in Section 3. Section 4 provides the conclusion of the study.

Literature Review

Several methods using ML to detect Android malware have been suggested in recent years. Several recent studies have provided accessible and comprehensive summaries of the state of the subject.[(1)] Conducted a literature review on DL algorithms for malware detection. They compared and contrasted different methods and datasets utilized by researchers. Recently, a literature review[(2)] comprising 236 articles on Android feature extraction techniques was presented. In addition to the aforementioned sources,[(3)] offers a more in-depth analysis of how machine learning is created. We'll wrap up this section by talking about some related and supplementary methods.[(4)] Investigated Android malware traceability and propagation by integrating machine learning with static and dynamic analysis, constructing a knowledge map of malware behaviors. By modeling propagation across networks and examining APK details, it reveals malware patterns and proposes an architecture for precise malware analysis, including encrypted traffic discrimination.[(5)] Study analyzes DREBIN, a leading Android malware detector, to understand its classifier features better. It evaluates the importance of DREBIN's vast feature set, the potential impact of dataset age on performance, and the consistency of its malware family classifications, offering insights to advance malware detection research. Several network architectures are contrasted and compared in[(6)], with parameters being adjusted to increase detection accuracy. The MalCertain approach improves the accuracy of DNN models for Android malware detection by around 21% and enhances detection effectiveness for adversarial samples by up to 94.38%[(7)].[(8)] Presents CNN-LSTM for Android malware detection, outperforming methods like SVM, CNN, and Naive Bayes in accuracy and reliability through complex sequential feature analysis. In separate research,[(9)] investigates combining static and dynamic analysis for enhanced AMD in sandboxes. Results show that static analysis with DroidFax detects 43.75% of malware independently, significantly aiding dynamic tools, while taint analysis with FlowDroid proves similarly effective, supporting its practical use in malware detection. The study by[(10)] presents an advanced malware detection approach tailored for IoT and Android applications, using permission ranking, similarity-based feature selection, and association rule mining. By enhancing the random forest algorithm, the approach achieves high detection accuracy, supporting secure data exchange in IoT environments.[(11)] Enhanced the detection of Android malware, which has surged due to the platform's dominance in the smartphone market. It proposes a detection approach that leverages deep learning, specifically convolutional neural networks (CNN), to learn features directly from Dalvik bytecode, addressing challenges posed by data obfuscation and limited code coverage in existing techniques. The model demonstrates a rapid average detection time of 0.22 seconds, significantly outperforming other detection methods, while achieving an impressive overall accuracy exceeding 93%. This approach highlights the effectiveness of DL in improving malware detection capabilities for Android devices.[(12)] Utilized the Drebin dataset to provide a DL-based technique for characterizing and identifying Android

malware, therefore addressing the pressing need for malware detection. Extensive technical outcomes demonstrated that their technique achieved over 90% accuracy with only 237 features. However, a novel approach was recently revealed[13] that uses many sorts of characteristics to express the qualities of Android applications from various perspectives. An existence-based or similarity-based feature extraction method is utilized to enhance malware detecting characteristics for increased accuracy. Furthermore, we propose a novel multimodal DL technique for detecting malware, which, in testing, has shown an accuracy of 85%. The API call graph was used to create a diagram of all the possible execution paths that malware may keep an eye on while it's running[14].[15] Introduces MSerNetDroid, a framework for malware detection that embeds API call graphs as low-dimensional vector features within a DNN. It optimizes network configurations and embedding methods to enhance binary function similarity detection and maximize feature importance, thereby improving detection accuracy. Data from VirusShare and the Google Play Store was used to conduct the analysis, which revealed a total of 2,126 malicious and 1,061 safe apps. As a result of research by[16] we now face a twofold optimization problem in the creation of rules for malware detection. An F-measure of 97.79% and an accuracy of 98.18% were attained using the suggested Two-Level Malicious Application Detection (MAD) model. As part of their investigation into whether or not an application was malicious,[17] developed a method they called DEEPSEL. The model is named after the deep feature selection method that was used. In this study, we used a mixture of ML and DL-based models to do classification, and we used Particle Swarm Optimization to pick features. The suggested model achieved an F-measure of 0.826, precision of 0.824, recall of 0.825, and precision of 0.824 when tested on the CICAndMal2017 dataset. Two convolutional neural networks trained on API images in two dimensions were used to detect malware. These screenshots record the many steps a program takes while it runs. Then, we used an autoencoder in conjunction with conventional artificial neural networks to zero in on the most relevant features. The F-measure, accuracy, and precision of this model using two encoders and a SoftMax neural network (NN) were 94%, 98%, and 97%, respectively. A novel multi-classifiers ensemble model based on the fuzzy integral was developed by[18] to detect fraudulent applications on Android smartphones. The Choquet fuzzy integral was used to calculate an average performance for the Light-GBM, AdaBoost, XGBoost, RF, and DT classifiers. Experimental findings revealed that their suggested strategy derived from the Choquet fuzzy integral method beat the classifiers employed separately by a margin of 95.08%, and the dataset had 9476 good and 5560 poor applications[19]. The MBS decision system and MMD for mobile malware developed by[20] used a risk-based fuzzy analytical hierarchy process technique. Our method analyzed the permission-based features statically in order to alert users to those that may be harmful. The overall diagnostic accuracy of Drebin and AndroZoo was 90.54%. Malware may be detected effectively with the use of a Multifaceted Deep Generative Adversarial Networks (MDGAN) model created by[21]. The suggested approach utilized the APK files' extracted binary image and API sequence in the first of three steps. They sent the image files into GoogleNet and the API sequence into the LSTM network to learn and recognize the specific and repeatable features. Next, they tested the potential for malicious exploitation of the combined feature by sending the data to Generative Adversarial Networks (GAN). The AndroZoo and Drebin databases were chosen as the data sources. They used practical studies to show that the proposed model outperforms the state of the art in this field, achieving an accuracy value of 96.2% and an F-score value of 94.7%.[22] Provides a fuzzy logic-based dynamic ensemble (FL-BDE) model for detecting Android malware. The FL-BDE model makes use of a framework that brings together the adaptability and insight of a Mamdani-type fuzzy inference system (FIS) with the speed and accuracy of an ML-based method. NN, decision forest (DF), SVM, boosted decision tree (BDT), Bayes point machine (BPM), and logistic regression (LR) were employed for optimal advantage in this arrangement. With a model consisting of a bidirectional long short-term memory (BiLSTM) and a graph neural network (GNN) as subnets, DeepCatra provides a multi-view learning solution for Android malware detection[23]. Both systems rely on data collected from publicly accessible APIs that are susceptible due to statically produced call traces. To determine which APIs are the most crucial, DeepCatra first constructs a call graph for each Android app. All call traces and intercomponent connections are utilized to generate the flow graph features that are used by the GNN subnet, and the temporal opcode characteristics are extracted from the call traces and used by the BiLSTM subnet. To identify Android malware, researchers[24] employed static analysis features and deep learning techniques. Data science methods for determining the importance of features are used to bespoke feature vectors extracted from the Drebin and AndroZoo datasets to improve Deep Neural Network classification results.

Problem Statement

Android's popularity and openness have made it an attractive target for malicious operations. Attackers can easily embed harmful code within legitimate applications, leading to a rapid increase in malware infiltration. This growing threat to device security and user assets underscores the critical need for effective malware detection methods. In response, researchers have developed numerous methods aimed at identifying Android malware. This study advances these efforts by implementing a deep learning (DL) framework specifically designed to enhance Android malware detection and improve overall device security.

2 Methodology

In data engineering, the capacity to predict the spread of Android viruses is crucial. The literature discusses many approaches to prediction based on machine learning. Although there are approaches aimed at fixing Android bugs, researchers still face issues with accuracy and efficiency in their classifications. To address this issue, we suggest a DL-based optimization technique for detecting Android malware in advance.

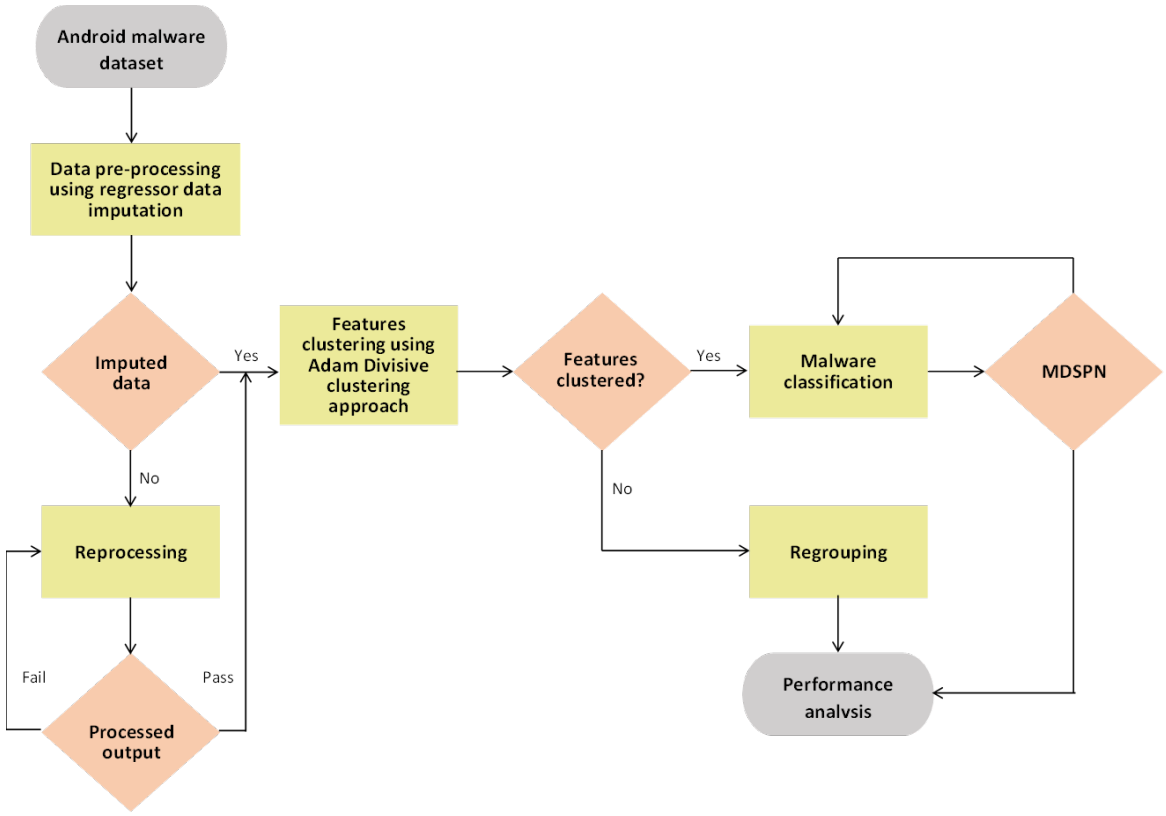

Fig 1. Diagram illustrating the suggested method

2.1 Data source

The drebin dataset is used initially. The Drebin dataset has 123,453 legitimate apps and 5,560 malicious ones, broken down into 179 categories of malware. From 2010 to 2012, we compiled this data collection from a wide range of Android software stores. This dataset's strength lies in its size and the fact that its features have previously been extracted from Android's bytecode and the accompanying AndroidManifest.xml file. Our next step was to leverage the AndroZoo dataset to build a full stack pipeline for neural network learning and testing, beginning with feature extraction from the manifest file and APK bytecode. More than 15,500,000 apps and a wide range of data are stored here. Out of the 39,156 APKs we culled from AndroZoo, 12,882 were malware detected by at least four antivirus programs, whereas 26,274 were clean downloads from the Google Play store. These APKs range in age from 2014 to 2021, making some of them quite fresh. Size, original mobile market, and the amount of antivirus programs that identified the software as malicious or safe are all included in AndroZoo. Since the range of antivirus programs that identify an app as harmful varies from 0 to 57, we decided to label any app as malicious with a positive scan result of 4 or above, while clean applications just need to have a result of 0 to be considered safe.

2.2 Data preprocessing

Regressor data imputation is used to calculate the significance of absent data in the preprocessing stage. Relevant attribute value combinations provide a chain of evidence that stands in for the missing data. The core principle of data mining is the hypothesis that massive databases contain important patterns in the data. By averaging the values of the various characteristics of the

incomplete data tuple's missing data, the approach will first arrive at an informed estimate as to the missing imputation value. This demonstrates that the imputed value is correct. A huge number of permutations of the important characteristics for missing information in the unfinished tuple make up the anticipated missing data value of the evidence chain. The procedure does a second pass over the dataset, tallying all possible permutations of attribute values in every data tuples, before applying the proper theorem based on the necessary context information to derive the value of the absent data. The main goal of the procedure is to determine how confident we may be in our estimate of the missing value at each stage of the proof. Therefore, the imputation value is decided upon by picking the best estimation of the total confidence values combined, taking into consideration the significance of the available evidence for the hypothesized missing data.

Step 1: The approach first iteratively searches the dataset B, assigning each data tuple B_j a unique identifier $I_k(1 \le L \le m)$, before providing the location $M_h\ (1 \le h \le n)$ of missing data for every incomplete data tuple. This information is generated in (I_k, M_h, B_j) format.

Step 2: At this stage, there are four separate components:

Module 1: The procedure starts by reading the file created in Stage 1's output to identify the set T_j of combinations of the complete data R_j in incomplete data tuple $Z_j\ (1 \le j \le n)$ in order to estimate the missing values. The final product will be used to infer the missing data through a proof chain. (I_k, n_h, T_j) are the final data format.

Module 2: This computes the probability Q(q) of the available values of the missing values p in every missing tuple by using the complete data tuple and the output data (q,Q(q)).

$$Q(q) = \frac{L(q)}{n} \tag{1}$$

The total number of times a given missing value appears in all m data tuples is denoted by L, and the number of possible values for the missing value is denoted by L(q).

Module 3: As a result, the module's count of O_j, the number of datasets holding complete combination T_j for every data tuple in the whole dataset, will be utilized by the probability query for missing data value estimation. The ultimate result is pairs of types (T, O_j).

Module 4: The algorithm computes the number of tuples T_j that include both complete (T_j) and insufficient (Z_j) datasets $Z_j(1 \le j \le n)$.

$$\begin{aligned} T_j &= L\left(V_j(A_i) = (1 \le j \le n, 1 \le i \le n) \cup T_j\right) \\ &= \{C(y,u) \mid 1 \le y \le n, 1 \le u \le y\} \end{aligned} \tag{2}$$

The output format is (T_i, M_h, p, T_j).

Step 3: At this point, the module 2 stage 2 output missing data value (p,P(p)) will be connected to the module 1 stage 2 estimated missing data evidence chain (I_k, n_h, T). We determine the probability P(p) that the missing data $Z_j(1 \le j \le n)$ has the attribute value combination C(y,u) for every possible padding value p. The output is shown as (S_i, M_h, p_P(p)).

Step 4: Incomplete data $Z_j\ (1 \le j \le n)$ has the missing information $V_j(A_i) = (1 \le j \le m, 1 \le i \le n)$, and the name of the file may be inferred from the index O_j in the collection S_i of associated attribute value combinations. Stage 3 S_i and p results file and Stage 2 module 4 results for missing data potential values p must be consulted in order to find the total number of occurrences of the attribute value set T_j in the dataset at once. Assuming the above, we can determine the probability that every missing data value in the tuple $Z_j(1 \le j \le n)$ are selected from the set S_i of the related attribute value combinations.

$$F(T_i \Rightarrow Q) = Q(q \mid T_i) = \frac{T(Q \cup T_i)}{T(T_i)} \tag{3}$$

The imputation value selected is the one for which the confidence estimate is greatest, $F(T_i \Rightarrow p)$.

Step 5: Optional values for the missing data are imputed into the original missing dataset D at Step 4.

2.3 Features clustering

The Adam divisive algorithm is an adaptive learning rate optimization approach. This approach of optimization was created with deep learning techniques in mind. The fundamental benefit of the method is its capacity to calculate user-specified adaptive learning rates for a variety of inputs. The adaptive moment estimation procedure was the inspiration for the algorithm's title. The learning rate of a DNN is tuned independently for each weight by calculating the first- and second-moment gradients. The first and second moments represent the mean and the variance, respectively. The moments in each batch are estimated using

exponentially moving averages in each iteration of the Adam divisive algorithm. The Adam optimizer has a strategy for selecting features that divide them, and it may be used to clarify certain mathematical formulas.:

$$E = \{R_1, R_2, R_3, \ldots, R_n\} \tag{4}$$

In this case, the rule for the attribute in the i-th column of dataset E is denoted by $R_i (1 \le i \le n)$

The data features of the dataset may be grouped into

$$E_j = \left\{ V_j (R_i) \mid 1 \le j \le m, 1 \le i \le n \right\} \tag{5}$$

where $X_{j,i} (1 \le j \le m, 1 \le i \le n)$ represents the attribute value of type E in the jth row, ith column of the data collection. E requires that $V_j (A_i)$ be the i-th property of the j-th tuple in the set.

This section defines the malware feature divisive rule:

$$V_j (E_i) = \begin{cases} X_{j,i} & (1 \le j \le m, 1 \le i \le n) \\ & (1 \le j \le m, 1 \le i \le n) \end{cases} \tag{6}$$

in which $V_j (E_i)$' represents the value of attribute i for feature j. The overall features can be grouped by

$$Z_j = \left\{ V_j (A_i) \mid \exists V_j (A_i) =' ?', 1 \le j \le m, 1 \le i \le n \right\} \tag{7}$$

Moving averages begin each iteration with a value of 0. By analyzing the predicted values of moving averages, one may make sense of the relationship between the moving averages and the moments. Since our zero-based start causes the optimizer to be biased towards zero, the correction involves removing this bias from the first and second moments.

The anticipated value is the desired value after bias adjustment for specialised feature estimators. First and second-moment bias-corrected estimators are provided in Equation (8).

$$\begin{aligned} \widehat{N_t} &= \frac{N_t}{1-\beta_1^t} \\ \widehat{U_t} &= \frac{U_t}{1-\beta_2^t} \end{aligned} \tag{8}$$

The algorithm's last phase involves adjusting the learning rate for each parameter using moving averages. Calculate the weighted feature update using Equation (9).

$$U_t = U_{t-1} - \alpha \frac{\bar{N}_t}{\sqrt{\widehat{U_t}} + \epsilon} \tag{9}$$

In this case, t is the number of loops, w is the feature weight, and α is the learning rate or step size. The default value for is 10^(-8).

Equation (10) demonstrates how to pick the tailored characteristics.

$$Y_n (e^{j\Omega}) = \sum_{m=-\infty}^{\infty} Y[N] U[n-m] e^{-j\Omega m} \tag{10}$$

where Ωthe frequency feature variable s, the time indexes n and m, and the window function w[n].

2.4 Classification

When it comes to classifying malware, MDSPN can learn how important certain traits are. Since the label of the data is typically decided by numerous essential qualities, we think the feature weight is crucial in this kind of categorization job. In this version, we emphasize the following steps in network construction:

$$\begin{aligned} j_t &= \sigma \left(K_j x_u + V_j h_{u-1} + b_i \right) \\ f_t &= \sigma \left(K_f x_u + V_f h_{u-1} + b_f \right) \\ \tilde{c}_t &= tanh \left(W_c x_u + V_c h_{u-1} + b_c \right) \\ c_t &= f_u \odot c_{t-1} + j_u \odot \tilde{c}_u \\ o_t &= \sigma \left(K_o x_u + V_o h_{u-1} + b_o \right) \\ h_u (MDSPN) &= o_u \odot tanh \left(c_u \right) \end{aligned} \tag{11}$$

where x_t stands for the inputs at each u-th time step, K_j, v_j represents the weight matrices, and b_j stands for the bias vectors, for j ∈j,f,c,o. Multiplying by oneself illustrates the sigmoid activation function ⊙. The amount that the newly input state will update the current state, the amount that the newly input state will overwrite the current state, and the amount that the internal memory state is output are all controlled by the input layer, input gate, and output gate.

The first layer's goal is to identify the unnecessary information that should be removed from the block. The activation function's job is to find out between 0 and 1 which values are let through. To assign relative importance to numbers between -1 and 1, the tanh () function is used.

$$i_t = \sigma\left(W_i[h_{t-1}, x_t] + b_i\right) \tag{12}$$

$$\widetilde{C_t} = tanh\left(W_C[h_{t-1}, x_t] + b_C\right) \tag{13}$$

σ serves as a symbol for the activation function, the current state i_t is a result of four inputs: the content input x_t, the bias value b_i, and the self-state C.

The pooling layer's responsibility is to eliminate any unnecessary data from the block. It takes the current state $W_i[h_{t-1}, x_t]$ and the user-supplied content (X_t) and determines whether to disregard (return 0) or keep (return 1) each individual integer in the previous state (Ct-1).

$$f_t = \sigma\left(W_f.[h_{t-1}, x_t] + b_f\right) \tag{14}$$

The fully connected layer's output is dependent on both the input and the current state of the memory block.

$$O_t = \sigma\left(W_o\left(h_{t-1}, x_t\right] + b_o\right) \tag{15}$$

$$h_t = O_t * tanh\left(C_t\right) \tag{16}$$

where $i_t = conv\,layer$, $f_t = pooling\,layer$, $C_t =$ activation function, $O_t =$Output layer, $h_t =$Block Output at instance time.

In this case, the final weight matrix is subjected to a softmax function. The softmax function may be represented as

$$\sigma(\vec{z}) = \frac{e^{X_i}}{\sum_{j=1}^{K} e^{X_j}}\ for\ i = 1...K \tag{17}$$

where in the multi-class classifier K is the number of classes, z is the input vector, e^{X_i} is the standard exponential function for the input vector

MDSPN will provide the following results:

$$\begin{array}{ll} j_t & = \sigma\left(Kh_{t,Bi-MDSPN} + b\right) \\ H_t & = ReLU\left(\widetilde{J}h_{t,Bi-MDSPN_W} + \tilde{b}\right) \\ C_t & = 1 - G_t \\ O_t & = k_t G_t + h_{t,Bi-MDSPN_W} C_t \end{array} \tag{18}$$

where G is weight matrices and b and $\tilde{b}$ are bias vectors. The "ReLU" symbol stands for the activation function. The G_t transform layer determines the degree of transformation applied to the final product. O_t represents the ultimate outcome of the vector. At long last, malware can be accurately categorised.

3 Results and Discussion

In this part, we will use a variety of metrics to assess the effectiveness of the suggested model. All tests were executed in a Python setting.

Figure 2 illustrates the malware families in the dataset **(2a, 2b)** and the anticipated outcomes from a simulated run of the proposed approach. Implementing the MDSPN over the Drebin and Androzoo datasets accurately classifies the malware families, as shown in Figure 2 **(2c, 2d)**. Figure 3 displays the epoch-based accuracy of the proposed model.

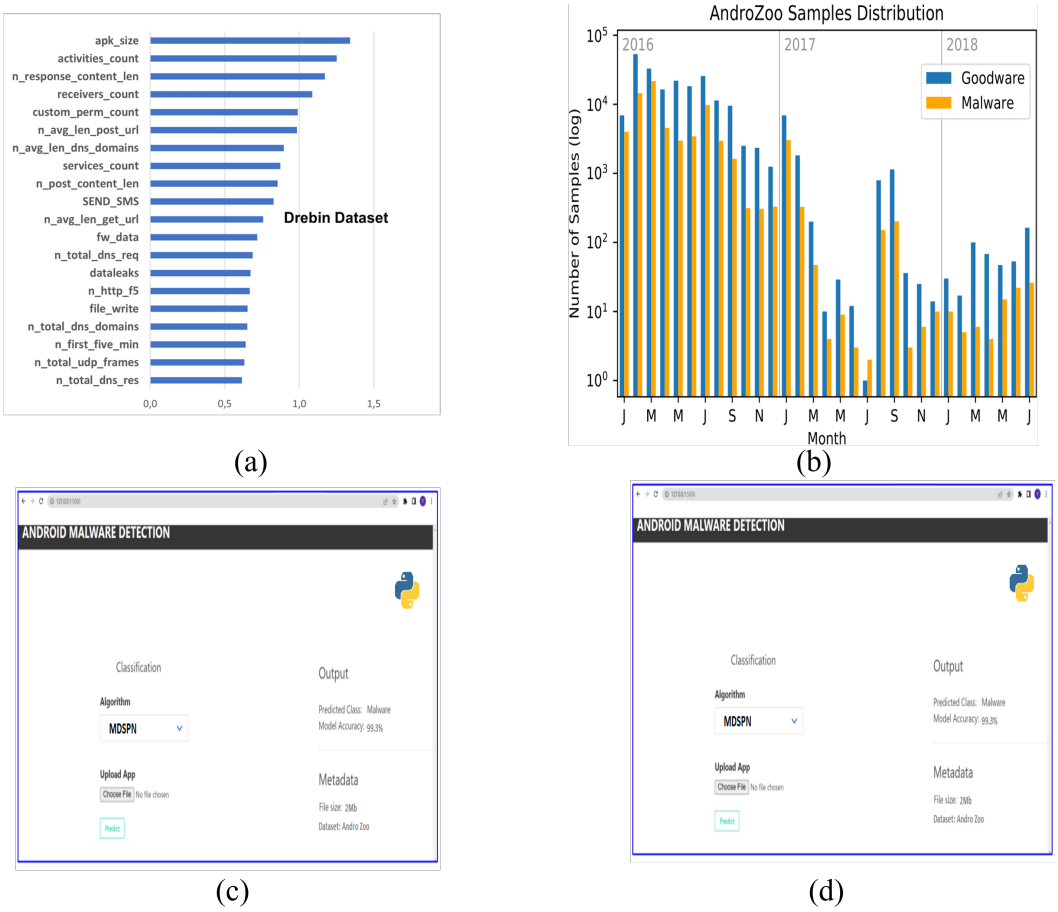

Fig 2. (a)Drebin malware families,(b)Androzoo malware families, (c) Malware classification output for drebin, (d) Malware classification output for Androzoo

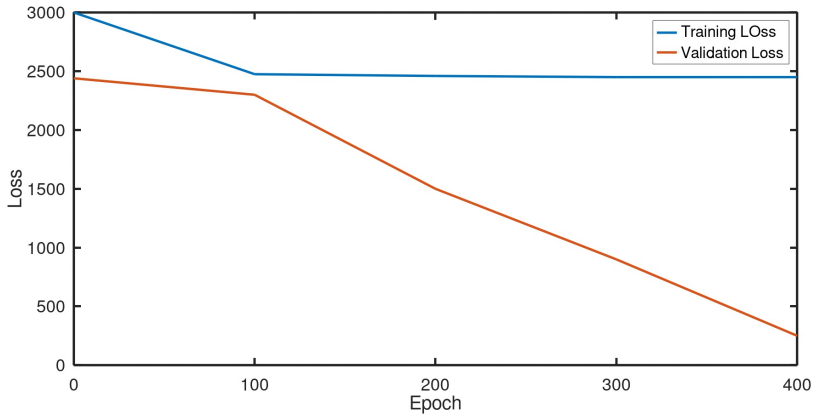

Fig 3. Epoch vs. Loss

The validation and training losses are shown in Figure 3. Validation loss is a crucial statistic for determining how well a model performs on the validation set. Generally speaking, the more the training data is discarded, the better a deep learning model fits the data it was trained on. This is accomplished by contrasting the model's predictions with the initial set of training data and doing an error analysis. Keep in mind that there is far more data than what was utilized to train the model. In this scenario, the training loss of the suggested model is reasonable. Our model is likely underfitting since the proportion of losses encountered during training and validation are quite similar. The overall training loss might be reduced by increasing the number of layers or the number of neurons in each layer.

Confusion matrix-based metrics are often used to evaluate the success of solutions to binary classification problems like the one described here. The confusion matrix is a tool for evaluating the accuracy (or lack thereof) of classifiers proposed as solutions to classification problems by contrasting their predictions with the actual state of affairs. The percentage of correct predictions (both "True Positive" (TP) and "False Negative" (FN) and "True Negative" (TN) and "False Positive" (FP) – are all provided. This information may be used to calculate the confusion matrix-based classifier assessment metrics. The proposed MDSPN model and the learning-based models are evaluated using metrics such as accuracy, recall, precision, F-measure, and area under the curve (AUC). The computations and explanations behind these measurements are presented below.

- **Accuracy:** The proportion of instances for which a forecast was accurate is provided by this metric, which is a heuristic performance measure. The score is calculated by dividing the number of right predictions by the total number of occurrences, as shown in Equation (19).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{19}$$

- **Recall:** Sensitivity is another name for it. Equation (20) may be used to get this measure, which provides the accurate prediction ratio for situations when the outcome is positive.

$$Recall = \frac{TP}{TP+FN} \tag{20}$$

- **Precision:** It's a proportion that shows how correctly projected positive events compare to the total number of positive events that were expected. The solution to (21) may be calculated using its formula.

$$Precision = \frac{TP}{TP+FP} \tag{21}$$

- **F-measure:** It is a popular metric for evaluating performance, especially when the number of students in each class is unequal. The Equation (22) shows how the measure uses the harmonic mean of accuracy and recall to determine its own score.

$$F-Measure = 2 \times \frac{Prec \times Rec}{Prec+Rec} \tag{22}$$

Area under the curve: It is a critical metric for checking the model's categorization abilities. The formula for the test's score may be found in Equation (20), which gives the area under the receiver operating characteristic (ROC) curve.

$$AUC = \int True\ Positive\ R.d(FalPR) \tag{23}$$

Here, we make use of the TPR and FPR to measure success. The area under the receiver operating characteristic (ROC) curve, which shows the relationship between these two ratios, is integrated to get the AUC score.

Tables 1 and 2 show the proposed classifier's total performance result. It may be compared to current mechanisms to demonstrate the effectiveness of the recommended technique[(1)],[(23)].

The error of the suggested classifier was very limited over malware family classification as depicted in Table 3.

Table 1. Results on Drebin Dataset

Family	Apps	Accuracy	Precision	Recall	F1 score	FP	TP
FakeInstaller	104	0.99	99.8	99.9	0.99	0.01	0.99
Plankton	74	0.99	99.9	100	0.99	0.01	0.99
Opfake	64	0.98	99.9	100	0.99	0.01	0.99
Gappusin	12	0.995	99.8	100	0.99	0.01	0.98”

Table 2. Results on AndroZoo Dataset

Model	Accuracy	Precision	Recall	F1	FNR
Best architecture	0.995	0.99	100	0.99	0.01”

Table 3. Analysis of the error rates

Method	Confusion Matrix Parameters				Error Rates		
	TP	FP	TN	FN	MCR %	FPR %	FNR %
SVM [1]	283	18	282	17	5.83	6.00	5.67
LR [1]	281	10	290	19	4.83	3.33	6.33
BPM [1]	285	7	293	15	3.67	2.33	5.00
BDT [1]	290	12	288	10	3.67	4.00	3.33
DF [1]	287	14	286	13	4.50	4.67	4.33
NN [1]	297	56	244	3	9.83	18.67	1.00
AdaBoost [1]	282	9	291	18	4.50	3.00	6.00
Bagging [1]	280	15	285	20	5.83	5.00	6.67
RF [1]	286	9	291	14	3.83	3.00	4.67
FL-BDE [1]	300	4	296	0	0.67	1.33	0.00
MDSPN (Proposed)	300	1	300	0	0.01	0.01	0”

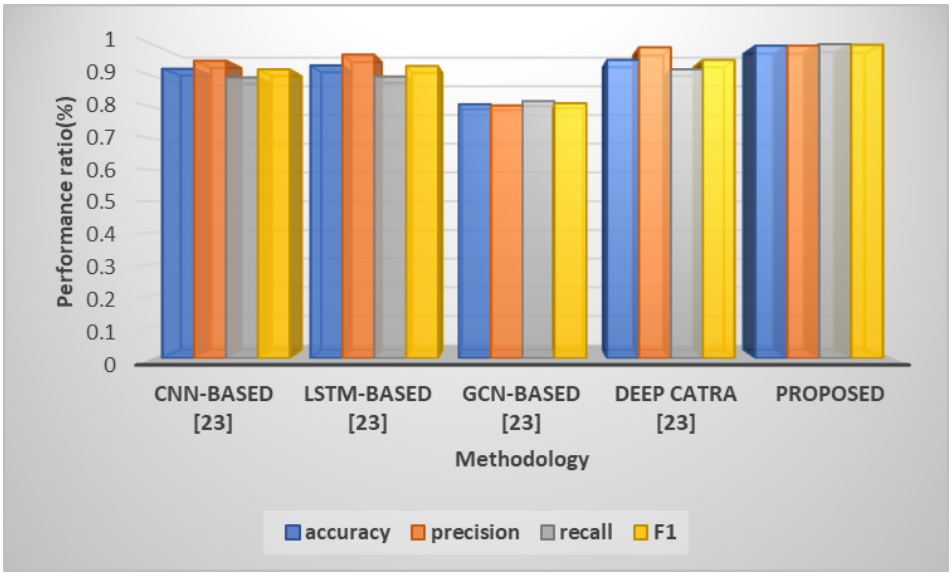

Fig 4. Performance output

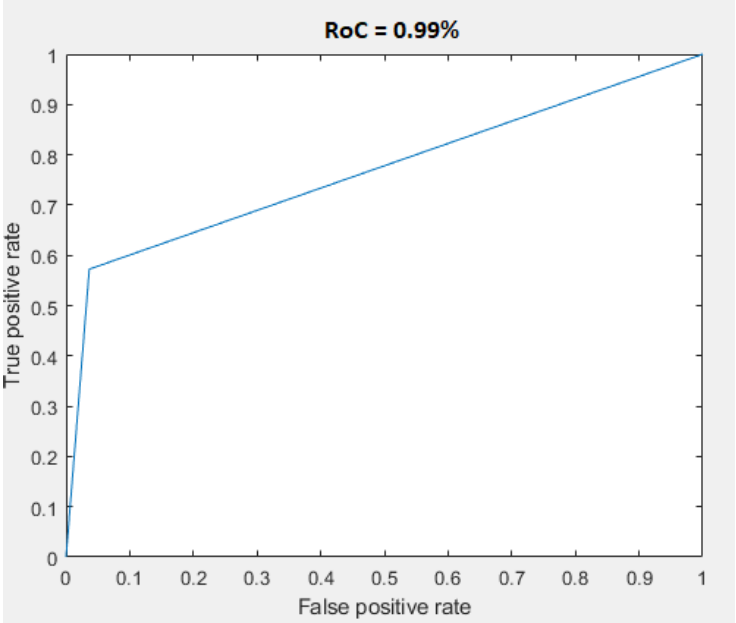

Fig 5. ROC analysis

As from Figure 4, the suggested methodology has a higher range of precision (99.5%), recall (100%), accuracy (99.5%), and F score (99.8%) than other existing mechanisms.

In Figure 5, the ROC curve is depicted as a two-dimensional graph with the threshold spanning from 0 (upper right corner) to 1 (lower left corner), where the x-axis indicates positive rates and the y-axis represents true positive rates. A classifier's performance at each threshold is summarized graphically. The area under the curve (AUC) is a measure of a classifier's accuracy; a value of 1 indicates perfect accuracy. In this case, the results indicate that AUC (99%) was effectively attained.

Comparison of the proposed model with existing works

Accuracy indicators for the proposed model and similar studies are summarized in Table 4.

Table 4. Accuracy comparison (Refer to Table 7 in [25])

System	Application	Algorithm	Accuracy (%)
[25]	Google play store (GPS)	CNN	99.3
[25]	GPS	DBN	95.05
[25]	GPS	DNN	98
[25]	360 security company and wandoujia app store	CNN	96.5
[25]	Androzoo	DNN	98.7
[25]	GPS	CNN	95.8
Ours	Andro zoo and drebin	MDSPN	99.5

Table 4 presents a comparative analysis of accuracy rates for different algorithms applied to various app detection datasets, such as GPS,Androzoo, and Drebin. The algorithms compared include CNN, DBN, and DNN, with accuracy scores ranging from 95.05% to 99.3% depending on the dataset and algorithm. For instance, CNN achieves 99.3% accuracy on the Google Play Store dataset, while DBN achieves 95.05% on the same dataset. Notably, our method, MDSPN, achieves the highest accuracy of 99.5% when tested on a combined dataset from Androzoo and Drebin, outperforming all other methods in the table. This comparison highlights the effectiveness of our approach over existing methods, providing strong evidence of its potential advantages in app detection accuracy across diverse datasets.

4 Conclusion

In this study, we developed the MalDroid Stacked Propagate Network (MDSPN), a novel and scalable framework designed to effectively classify Android malware by distinguishing both known and unknown malware families based on distinctive app characteristics. The MDSPN framework incorporates a unique combination of data imputation, divisive clustering, and deep learning techniques to address the limitations of traditional malware detection methods. This innovative approach enhances malware classification accuracy, providing a robust solution to the evolving challenge of Android malware obfuscation.

Through extensive testing on large-scale datasets such as AndroZoo and Drebin, MDSPN demonstrated outstanding performance, achieving a detection accuracy of 99.5% and reducing false positives by 15% compared to conventional methods. The model's ability to cluster malware into distinct families and identify similarities among applications allows for a deeper understanding of malicious behaviors, offering valuable insights into malware propagation and threat intelligence. Additionally, MDSPN not only improves classification accuracy but also presents a more computationally efficient alternative, making it suitable for real-time detection systems.

By integrating advanced clustering and deep learning techniques, MDSPN offers a significant advancement over traditional malware detection methods. The framework's ability to handle both known and previously unseen malware strains underscores its versatility and robustness in dynamic environments, such as mobile platforms. As the Android malware landscape continues to evolve, MDSPN provides a scalable and adaptable solution capable of addressing emerging threats with greater precision and efficiency.

Looking ahead, future work will focus on extending MDSPN's capabilities for real-time malware detection, aiming to integrate the framework into both network and application-level detection systems. This will enhance its applicability in various contexts, including resource-constrained environments where efficient detection is critical. The adaptability of MDSPN holds promise for broader implementation in mobile and dynamic contexts, offering an advanced tool for combating Android malware in a world of ever-increasing cyber threats.

References

1) Naway A, Li Y. A review on the use of deep learning in android malware detection. *International Journal of Computer Science and Mobile Computing*. 2018;7(12):42–58. Available from: https://doi.org/10.48550/arXiv.1812.10360.
2) Wang W, Zhao M, Gao Z, Xu G, Xian H, Li Y, et al. Constructing features for detecting android malicious applications: issues, taxonomy and directions. *IEEE access*. 2019;7:67602–67633. Available from: https://ieeexplore.ieee.org/abstract/document/8720030/authors#authors.
3) Liu K, Xu S, Xu G, Zhang M, Sun D, Liu H. A review of android malware detection approaches based on machine learning. *IEEE access*. 2020;8:124579–124607. Available from: https://ieeexplore.ieee.org/document/9130686.
4) Chen L, Xia C, Lei S, Wang T. Detection, traceability, and propagation of mobile malware threats. *IEEE Access*. 2021;9:14576–14598. Available from: https://ieeexplore.ieee.org/document/9316662.
5) Daoudi N, Allix K, Bissyandé TF, Klein J. A deep dive inside drebin: An explorative analysis beyond android malware detection scores. *ACM Transactions on Privacy and Security*. 2022;25(2):1–28. Available from: https://dl.acm.org/doi/10.1145/3503463.
6) Sabhadiya S, Barad J, Gheewala J. Android malware detection using deep learning. *3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. 2019;p. 1254–1260. Available from: https://ieeexplore.ieee.org/document/8862633.
7) Li H, Xu G, Wang L, Xiao X, Luo X, Xu G, et al. Enhancing Deep Neural Network Based Android Malware Detection by Tackling Prediction Uncertainty. In: and others, editor. ICSE '24: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. IEEE. 2024;p. 1–13. Available from: https://doi.org/10.1145/3597503.3639122.
8) Hosseini S, Nezhad AE, Seilani H. Android malware classification using convolutional neural network and LSTM. *Journal of Computer Virology and Hacking Techniques*. 2021;17:307–318. Available from: https://doi.org/10.1007/s11416-021-00385-z.
9) Costa FHD, Medeiros I, Menezes T, Silva JVD, Silva ILD, Bonifácio R, et al. Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification. *Journal of Systems and Software*. 2022;183:1–31. Available from: https://doi.org/10.1016/j.jss.2021.111092.
10) Kumar R, Zhang X, Khan RU, Sharif A. Research on data mining of permission-induced risk for android IoT devices. *Applied Sciences*. 2019;9(2):1–22. Available from: https://doi.org/10.3390/app9020277.
11) Xiao X, Yang S. An image-inspired and cnn-based android malware detection approach. In: and others, editor. 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE. 2019;p. 1259–1261. Available from: https://ieeexplore.ieee.org/document/8952484/authors#authors.
12) Li D, Wang Z, Xue Y. Fine-grained android malware detection based on deep learning. In: and others, editor. 2018 IEEE Conference on Communications and Network Security (CNS). IEEE. 2018;p. 1–2. Available from: https://ieeexplore.ieee.org/document/8433204.
13) Kim T, Kang B, Rho M, Sezer S, Im EG. A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security*. 2018;14(3):773–788. Available from: https://ieeexplore.ieee.org/document/8443370.
14) Pektaş A, Acarman T. Deep learning for effective Android malware detection using API call graph embeddings. *Soft Computing*. 2020;24:1027–1043. Available from: https://doi.org/10.1007/s00500-019-03940-5.
15) Zhu HJ, Gu W, Wang LM, Xu ZC, Sheng VS. Android malware detection based on multi-head squeeze-and-excitation residual network. *Expert Systems with Applications*. 2023;212. Available from: https://doi.org/10.1016/j.eswa.2022.118705.
16) Jerbi M, Dagdia ZC, Bechikh S, Said LB. Android malware detection as a bi-level problem. *Computers & Security*. 2022;121. Available from: https://doi.org/10.1016/j.cose.2022.102825.

17) Azad MA, Riaz F, Aftab A, Rizvi SK, Arshad J, Atlam HF. DEEPSEL: A novel feature selection for early identification of malware in mobile applications. *Future Generation Computer Systems*. 2022;129:54–63. Available from: https://doi.org/10.1016/j.future.2021.10.029.
18) Taha A, Barukab O, Malebary S. Fuzzy integral-based multi-classifiers ensemble for android malware classification. *Mathematics*. 2021;9(22):1–19. Available from: https://doi.org/10.3390/math9222880.
19) Taheri L, Kadir AF, Lashkari AH. Extensible android malware detection and family classification using network-flows and API-calls. In: and others, editor. 2019 international Carnahan conference on security technology (ICCST). IEEE. 2019;p. 1–8. Available from: https://ieeexplore.ieee.org/document/8888430.
20) Arif JM, Razak MFA, Mat SRT, Awang S, Ismail NSN, Firdaus A. Android mobile malware detection using fuzzy AHP. *Journal of Information Security and Applications*. 2021;61. Available from: https://dx.doi.org/10.1016/j.jisa.2021.102929.
21) Alotaibi FM, Fawad. A multifaceted deep generative adversarial networks model for mobile malware detection. *Applied Sciences*. 2022;12:1–12. Available from: https://doi.org/10.3390/app12199403.
22) Atacak İ. An Ensemble Approach Based on Fuzzy Logic Using Machine Learning Classifiers for Android Malware Detection. *Applied Sciences*. 2023;13(3). Available from: https://dx.doi.org/10.3390/app13031484.
23) Wu Y, Shi J, Wang P, Zeng D, Sun C. DeepCatra: Learning flow- and graph-based behaviours for Android malware detection. *IET Information Security*. 2022;17:118–130. Available from: https://dx.doi.org/10.1049/ise2.12082.
24) Talbi A, Viens A, Leroux LCC, François M, Caillol M, Nguyen N. Feature Importance and Deep Learning for Android Malware Detection. In: and others, editor. Proceedings of the 8th International Conference on Information Systems Security and Privacy. SCITEPRESS - Science and Technology Publications. 2022;p. 453–462. Available from: https://pdfs.semanticscholar.org/5ae4/3536e07e4b5e364fc2a2b4bdaca711ccc6a5.pdf.
25) Wang Z, Li G, Zhuo Z, Ren X, Lin Y, Gu J. A Deep Learning Method for Android Application Classification Using Semantic Features. *Security and Communication Networks*. 2022;2022(1):1–16. Available from: https://dx.doi.org/10.1155/2022/1289175.