Journal of Computer Science

Original Research Paper

Securing Mobile Devices from Malware: A Faceoff Between Federated Learning and Deep Learning Models for Android Malware Classification

[1]Narayan Subramanian, [2]Logesh Ravi, [1]Mithin Jain Shaan, [1]Malathi Devarajan, [3,4]Tanupriya Choudhury, [5]Ketan Kotecha and [6]Subramaniyaswamy Vairavasundaram

[1]School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, Tamil Nadu, India
[2]School of Electronics Engineering, Centre for Advanced Data Science, Vellore Institute of Technology, Chennai, Tamil Nadu, India
[3]Department of Computer Science and Engineering, Graphic Era Deemed to be University, Dehradun, 248002, Uttarakhand, India
[4]Department of Computer Science and Engineering, Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune, Maharashtra, India
[5]Symbiosis Centre for Applied Artificial Intelligence, Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune, India
[6]School of Computing, SASTRA Deemed University, Thanjavur, Tamilnadu, India

Article history
Received: 20-10-2023
Revised: 02-11-2023
Accepted: 30-11-2023

Corresponding Author:
Tanupriya Choudhury
Department of Computer Science and Engineering, Graphic Era Deemed to be University, Dehradun, 248002, Uttarakhand, India;
Department of Computer Science and Engineering, Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune, Maharashtra, India
Email: tanupriyachoudhury.cse@geu.ac.in

Abstract: Amidst the escalating threats of android malware, urgency mounts to detect issues while safeguarding user privacy. Traditional machine learning and deep learning methods, dealt with scalability challenges and privacy compromises, finding a potential remedy in federated learning. This study introduces a groundbreaking federated learning-based methodology and compares federated learning with traditional deep learning techniques for Android malware classification, employing renowned datasets, including Drebin, Malgenome, Tuandromd, and Kronodroid. Shifting gears, a federated learning-based approach for malware classification excels in accuracy, scalability, and privacy preservation. Acknowledging limitations and ethical considerations, the study underscores the need for robust privacy measures and dataset transparency. This study unveils federated learning's prowess in android malware classification, opening doors to privacy-driven applications in diverse domains.

Keywords: Android Malware, Machine Learning, Deep Learning, Federated Learning, Privacy Preservation, Scalability

Introduction

The proliferation of mobile devices and the widespread usage of smart phones have made them prime targets for cyber attackers. Android is the most generally used mobile phone operating system and as such, it is oftentimes targeted by malware attacks. Traditional approaches to detecting and mitigating these attacks rely on centralized data storage, which compromises user privacy and scalability. To overcome these limitations, Federated Learning (FL) techniques have been suggested by many researchers for Android malicious software classification, which allows training machine learning models on decentralized data while preserving user privacy.

This study aims to differentiate between the performances of federated learning and traditional deep learning techniques for android malware categorization for which publicly available datasets are used to train and assess the efficiencies of FL and DL models. The research also focuses on examining an FL-based classification system that preserves user privacy and achieves high classification accuracy.

FL is a new advancement to scalable ML that is gaining popularity due to its ability to provide privacy and security awareness among a group of devices or individuals in use. Traditionally, with centralized control of machine learning such as neural networks such as MLP, all data is sent to the central server for training and processing, which requires strong communication and computer skills. Since user data is transferred and stored centrally, this process raises questions about privacy and security.

This study uniquely contributes to the field of Android malware classification by introducing and rigorously evaluating federated learning as an innovative alternative to traditional deep learning techniques. Its standout feature lies in the meticulous preservation of user privacy through decentralized data training, a critical advancement in the era of heightened privacy concerns. The paper not only showcases the superior performance of federated learning in terms of accuracy and scalability but also provides a practical application of the methodology in Android malware detection, establishing its real-world viability. Furthermore, the study extends its relevance by suggesting the potential applicability of federated learning in other domains where privacy considerations are paramount, solidifying its position as a promising and versatile approach with broad implications beyond the immediate scope of Android malware classification.

Federated learning allows personal devices to train themselves using local data, this means that data never leaves the device and the user's privacy is protected. Machine learning model parameters (such as weights) are encrypted and sent to the central server for model aggregation and then redistributed to a user for further learning. Knowledge from all devices can be shared iteratively to achieve accurate learning results.

Compared to traditional ML techniques, federated learning has several advantages, including better privacy protection, reduced communication and computational costs, and the capability to expand to a vast number of devices. Therefore, it has gained attention as a promising solution for privacy-preserving machine learning applications, such as Android malware classification.

The following major advances are presented in this study to create an effective Android malware classification model using network traffic:

- A detailed comparative study of the performance of federated learning and classic deep learning algorithms for android malware classification
- Deep insights into the seclusion and scalability welfares of federated learning for android malware classification
- Additionally, more in-depth exploration is conducted to determine the robustness of FL in the presence of a distributed set of data

This research paper is organized into five sections: Introduction, literature survey, background study, methodology, conclusion, and future scope. The literature survey section summarizes prior research on federated learning and DL models such as SNN, FNN, Autoencoders, and GRU. The background study section provides a brief introduction to each of the models used in the study. The methodology section explains the experiments conducted using four different datasets and compares the accuracy, stability, and security of the models. The results show that federated learning consistently provided an average of 96% accuracy. The conclusion section summarizes the study’s findings and provides insights into the future scope of research in the field, including optimization techniques, model compression, and improving communication efficiency. There are also proposals for practical uses of federated learning in various business applications.

In the field of mobile malware detection and classification, several studies have used ML and DL techniques to develop efficacious and coherent malware detection systems. This includes the use of centralized and federated learning approaches to improve accuracy and privacy preservation.

Pektaş and Acarman proposed a novel method that analyzed the timing and order of API calls for detecting malware on Android using sequential interval features of API calls (Pektaş and Acarman, 2020). Their method extracted sequences of APIrequests from malware samples and constructed interval-based features to feed into a deep-learning model. Their results showed improved detection performance compared to traditional methods. Zhang *et al*. proposed a method for generating adversarial examples to evade Android malicious application detection using Generative Adversarial Networks (GANs) (Zhang *et al*., 2020). They trained GANs to generate samples that look similar to benign applications but are classified as malware by the target detection model. Their results showed the effectiveness of the proposed approach.

Lu *et al*. proposed an android malicious software detection method based on ensemble deep learning methods (Lu *et al*., 2020). Their approach combined multiple deep learning models, including CNN and LSTM networks, to improve detection performance. Their results showed the effectiveness of the recommended methodology in detecting new and unknown malware samples. Ullah *et al*. proposed a novel hybrid technique for Android malicious software detection using DL and control flow graphs (Ullah *et al*., 2022). Their approach extracted features from control flow graphs of malware samples and fed them into a deep-learning model. Their results showed improved detection performance compared to traditional methods.

Krizhevsky *et al*. (2012) presented the AlexNet model, which utilized deep convolutional neural networks for the classification of images that achieved a significant improvement in performance on the ImageNet dataset compared to previous methods (Krizhevsky *et al*., 2012). Their success demonstrated the power of deep learning for computer vision tasks. The concept of generative adversarial networks is introduced by Goodfellow *et al*. (2020) which uses a generator network to create new data realistic enough to fool a discriminator network (Goodfellow *et al*., 2020). The presented work showed that GANs can be used for image and music generation, unsupervised learning, and other applications.

Radford *et al*. introduced the DCGAN architecture, which uses a generator and discriminator network in a GAN framework to learn unsupervised representations of data, such as images (Radford *et al*., 2015). The research on DCGANs can be used for the generation of images, feature extraction, and other unsupervised learning tasks. Howard *et al*. introduced mobile nets, a family of lightweight CNNs intended for use in mobile and embedded vision applications (Howard *et al*., 2017). The work by Howard *et al*. (2017) demonstrated that mobile nets can outperform other popular models on numerous computer vision benchmarks while using much fewer parameters and processing resources.

Almahmoud *et al*. proposed a DL-based technique for Android malicious software detection using static analysis (Almahmoud *et al*., 2021). Their approach used a CNN technique to extract features from the opcode sequences of Android application binaries. Their results showed the effectiveness of the proposed approach in detecting both known and unknown malware samples. Similarly, LeCun *et al*. introduced the CNN network, which has become the standard architecture for many computer vision problems (LeCun *et al*., 1998). The effectiveness of CNN in recognizing handwritten digits and the architecture of a specific CNN is described and called LeNet-5.

Cortes and Vapnik presented the SVM, a popular approach for classification and regression problems with the mathematical principles behind SVMs to demonstrate their effectiveness on a variety of problems, including handwriting recognition and face detection (Cortes and Vapnik, 1995). The LSTM architecture, which is a solution to overcome the vanishing gradient problem in training RNNs, is proposed by Hochreiter and Schmidhuber (1997) to demonstrate the effectiveness of LSTM in sequence prediction challenges, such as handwriting recognition and speech recognition.

Wilson *et al*. compare the adaptive gradient methods, such as Adam and Adagrad, to stochastic gradient descent and show that the former may not always provide significant improvement in convergence rates or generalization performance (Wilson *et al*., 2017). Pei *et al*. propose a framework for mobile malware detection using federated learning with a deep learning model (Pei *et al*., 2022). The approach uses local data on users' devices and federated averaging to aggregate model weights while preserving privacy. The method was tested on a real-world dataset and found to achieve comparable performance to a centralized model.

Mahindru and Arora present a federated learning approach to detecting malware in IoT networks (Mahindru and Arora, 2022). They used a CNN to learn from local data on IoT devices, federated averaging to aggregate model weights, and dynamic clustering to group similar devices for more efficient learning. Their approach achieves high accuracy on a synthetic dataset while reducing communication overhead compared to a centralized approach. The federated learning approach for malware classification on edge devices using an SVM model is proposed by Rey *et al*. (2022). The method achieves high accuracy, reduces communication overhead, and is robust to malicious attacks on edge devices, as evaluated on a real-world dataset.

A DL-based structure for detecting and classifying Android malware is proposed by Jebin Bose and Kalaiselvi, (2023). The framework uses static analysis to extract features from APK files and a deep neural network for classification. In terms of accuracy, experimental results reveal that the proposed structure outperforms standard ML-based methods. Aurangzeb and Aleem proposed an ensemble voting mechanism based on deep learning for evaluating and classifying obfuscated Android malware (Aurangzeb and Aleem, 2023). The proposed method outperforms traditional machine learning-based methods in terms of accuracy and robustness against obfuscation techniques. The paper highlights the potential of using DL techniques for Android malware detection and classification.

A comparison of traditional ML and DL models for the detection and classification of Android malware traffic is presented by Bovenzi *et al*. (2022). The paper suggests that DL-based techniques can improve the effectiveness of Android malware detection and classification. Yadav *et al*. (2022) propose a two-stage DL structure for image-based Android malware detection and variant classification. The framework consists of a feature extraction stage using a CNN and a classification stage using a deep neural network. The paper highlights the potential of using DL techniques for image-based Android malware detection and classification.

A metaheuristic optimization technique combined with a DL model for cybersecurity and Android malware detection and classification is proposed by Albakri *et al*. (2023). This approach aims to leverage the power of metaheuristics and DL to enhance the accuracy and robustness of Android malware detection and classification. A visualization-based binary classification method for Android malware using VGG16 is proposed by Marwaha *et al*. (2023). This proposed method achieves high accuracy in detecting and classifying Android malware by utilizing visualization techniques. The paper highlights the potential of using deep learning techniques for the visualization-based classification of Android malware.

Nguyen *et al*. (2023) propose a static analysis-based method for classifying android malware into categories and families. The proposed method extracts static features from the APK file and uses traditional machine-learning algorithms for classification. The results of the study suggest that the proposed strategy performs well in classifying android malware into various categories and families. An optimized ensemble learning method based on genetic algorithms for Android malware classification is proposed by Taha and Barukab (2022). In terms of accuracy and efficiency, research results suggest that the proposed method surpasses traditional ensemble learning methods. The paper highlights the potential of using genetic algorithms for optimizing ensemble learning in Android malware classification.

Chen *et al*. (2022) propose a new classification approach based on feature fusion and NLP for Android malware. The proposed method combines multiple feature extraction techniques and natural language processing to enhance the accuracy and robustness of Android malware classification. The paper highlights the potential of using feature fusion and natural language processing for the classification of mobile malware.

Materials

Given the rapidly growing number of malicious applications on the market android malware classification is a difficult task. Deep learning has shown promising results in detecting and classifying malware, but training the model requires a large amount of data. Yet, due to privacy concerns, legal restrictions, and the distributed nature of mobile devices, obtaining a large dataset is difficult. Federated learning is a novel approach that allows a machine learningmodel to be trained on distributed data without sharing the data itself. This study aims to compare the performance of deep learning and federated learning in the context of android malware classification while keeping in mind the privacy and security concerns associated with collecting and sharing sensitive data.

Federated Learning

FL is a distributed machine learning approach that allows model training to take place across a network of devices while data remains on edge devices or in decentralized databases. Instead of being sent to a centralized server for processing, data with federated learning remains under the control of the device owner. This strategy provides several benefits, including enhanced privacy and security, cheaper communication costs, and faster model training.

Healthcare, finance, transportation, and other industries could all benefit from federated learning. It is especially useful when data privacy is a concern, or when data is dispersed among several devices or databases. federated learning provides a powerful new tool for machine learning practitioners by allowing model training to take place in a decentralized and distributed manner. It is shown in Fig. 1 for a clear understanding.

Autoencoders

Autoencoders are a kind of unsupervised learning neural network that may be used to compress data and extract features. They are made up of two major components, the encoder and the decoder. The encoder decreases the input data's dimensionality by compressing it into a lower-dimensional representation. The decoder then uses the compressed representation to recover the original data as precisely as feasible. The overall architecture of the autoencoder is depicted in Fig. 2 and its purpose is to reduce the disparity between the original and recreated data.

Image and video compression, anomaly detection, and feature learning are just a few of the applications for autoencoders. One of their advantages is their ability to learn meaningful representations of input data without supervision, which can further be utilized for subsequent tasks such as classification or clustering. They are also relatively easy to implement and can be trained on large datasets efficiently.

Variational Autoencoders (VAE) are autoencoders that are utilized in generative modeling. VAEs, like ordinary autoencoders, learn a probability distribution across compressed representations and then sample from it to produce fresh data samples. VAEs have been used to produce realistic visuals, text, and music, among other things. Overall, autoencoders have shown to be versatile and strong DL algorithms.

Feedforward Neural Networks

An FNN model is a kind of artificial neural network in which information may only travel in one way, from the input layer to the output layer. It is a simple and widely used neural network architecture that has shown promising outcomes in various fields, including natural language processing, image classification, and speech recognition.

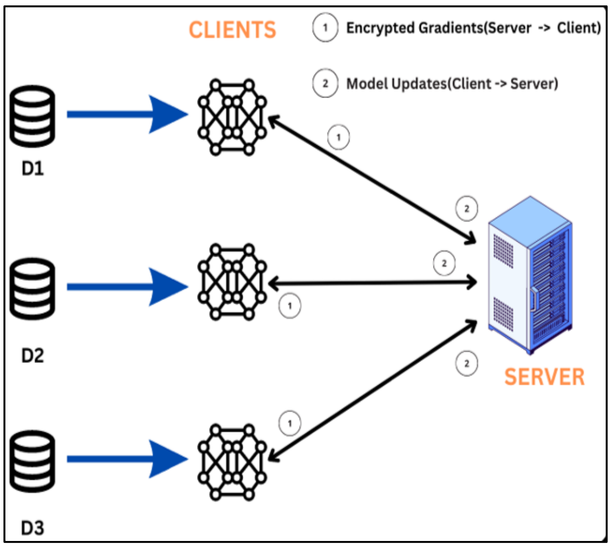
CLIENTS
1 Encrypted Gradients(Server -> Client)
2 Model Updates(Client -> Server)
D1
D2
D3
1
2
SERVER

Fig. 1: Federated learning architecture

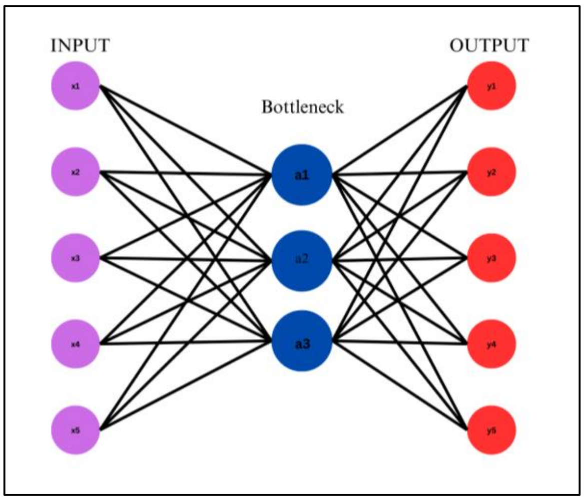

Fig. 2: Autoencoder architecture

An input layer with one or more hidden layers and an output layer comprises the FNN model. Every layer consists of a group of neurons that conduct a mathematical operation on their inputs to generate an output. In the input layer, the input features are fed to the network and the hidden layers perform a series of non-linear transformations on the input data. Depending on the objective, the output layer generates the network's final output, which might be a binary classification, multiclass classification, or regression problem.

The training of the FNN model is done by alternating the weights and biases of the neurons, thus reducing the difference between the actual and expected output. Backpropagation is commonly used for this method, which determines the gradient of the error for the weights and biases and updates them accordingly.

Overall, the FNN model is a powerful and versatile tool for solving various machine-learning problems. It is relatively easy to implement and train and with the proper choice of hyperparameters, it can achieve high accuracy on many tasks. The architecture of the feed-forward neural network is given in Fig. 3.

Gated Recurrent Unit

GRU is a neural network with the same structure as LSTM but with fewer features. GRU was created in 2014, as a solution to the vanishing gradient problem in classical recurrent neural networks. GRUs like LSTMs can detect long-term dependencies in datasets by selectively remembering or forgetting information over time.

The difference between GRU and LSTM is that GRU has a reset port and an update port whereas LSTM has a forget port, an in port, and an exit port. In the GRU, the update port controls how much of the entry is added to the current state, while the reset port means how much of the hidden state is forgotten. Figure 4 depicts the architecture of GRU.

GRU is frequently employed in NLP applications such as language modeling, voicerecognition, image processing, recommendation systems, and anomaly detection have all made use of them. GRU's design is simpler and contains fewer parameters than LSTM, making it faster to train and more computationally efficient.

Methods

In this section, differentiation is done between the performances of federated learning with the traditional deep learning methods such as simple neural networks, autoencoders, GRU, and FNN on four major datasets. While traditional DL methods train models on centralized data, FL enables training on decentralized data across multiple devices or data sources without compromising data privacy. By comparing the efficacy of these different methods, an observation can be made to better understand the advantages and limitations of federated learning in different scenarios.

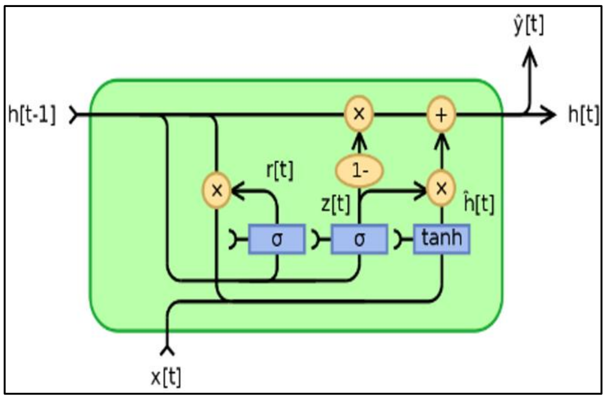

Fig. 3: Feedforward neural architecture

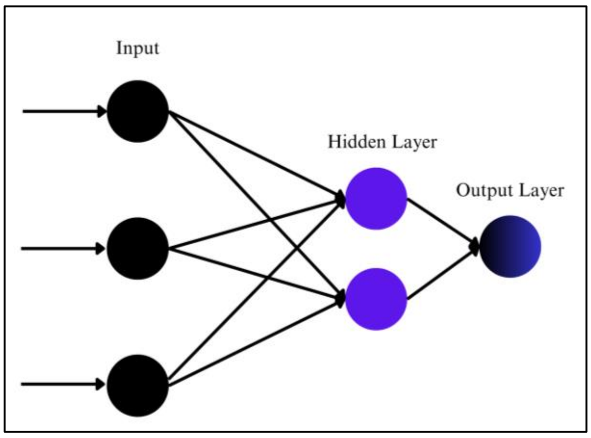

Fig. 4: GRU architecture

Dataset Description

The following datasets were used to compare the performance: Drebin, Malgenome, Tuandromd, and Kronodroid:

- Drebin is an Android application dataset that contains both malicious and benign samples. It has over 120,000 instances and is frequently used to evaluate Android malware detection systems
- Malgenome is another android application dataset that contains over 1,200 malware samples and over 6,000 benign applications. It is commonly used for evaluating malware detection systems and is especially useful for analyzing malware sample behavior
- Tuandromd is an Android malware dataset with over 17,000 samples from 24 different families. It includes both traditional malware and adware and it is commonly used to evaluate the effectiveness of ML algorithms for recognizing malware on android
- Kronodroid is an Android malware dataset that contains over 20,000 samples from 33 different families. It is intended to put machine learning algorithms to the test in detecting advanced malware, such as those that use obfuscation techniques to avoid detection

Environmental Setup

The tests were conducted on a Google Collaboratory (Colab) platform with an NVIDIA T4 Tensor Core GPU runtime environment. Colab is a cloud-based Jupyter Notebook environment provided by Google, which allows users to run their code on Google Cloud servers. The use of a GPU runtime environment in Colab enabled faster training and evaluation of the models, as compared to a CPU-only environment. The experimental analysis was performed using Python 3.8, with TensorFlow 2.6, TensorFlow-federated 0.19, and Keras 2.4 libraries for developing and evaluating the models. The datasets were loaded into the Colab environment using the panda's library.

Deep Learning Models

Simple neural network: The SNN model was implemented using the Keras library in Python. The dataset was loaded into the model using the Pandas library. The feature set of the dataset was extracted and preprocessed by standardizing the data using the standard scaler () function from the scikit-learn library. The labels were binarized using the label binarize () function and the scikit-lean'sfunction, train_test_split (), was used to divide the preprocessed set into testing and training sets.

The model architecture was defined with three dense layers. The first dense layer had 512 nodes with the input shape of the feature set and ReLU activation function. The second and third dense layers had 256 and 1 nodes, respectively, with the ReLU activation function. The dropout function was used to prevent overfitting in the model by randomly dropping out some nodes during training. The optimizer used was Adam, with accuracy as the chosen metric to measure the compiled model's performance. Using Binary Cross-Entropy (BCE) as the loss function, the model was trained. Table 1 presents the output of the SNN model:

$$y = \sigma\left(W_2\left(\sigma\left(W_1 x + b_1\right)\right) + b_2\right) \tag{1}$$

Feedforward neural network: In the FNN model for this study, two hidden layers are used with 512 and 256 neurons, respectively, using batch normalization and output to two hidden layers. The number of features in the dataset and the number of input operations are called neurons and the output operation has a single neuron with the ReLU activation function.

The binary cross entropy loss function is used and the Adam optimizer to specify the pattern and to avoid overfitting. The model was trained up to 10 times with 32 batch sizes. The architecture and hyperparameters of the FNN model remain the same for all datasets.

The interpretation of the FNN model was done using a variety of measures, including precision, recall, F1-score, accuracy, and AUC-ROC to verify its effectiveness in classification tasks and the results proved that the FNN model obtained a higher accuracy and performed well in differentiating between the classes in the datasets. The test loss and the test accuracy results for all the datasets used are presented in Table 2. The model's robustness was also tested by using different validation techniques, such as cross-validation and train-test splits and the results remained consistent:

$$y = f(w^L f(w^{L-1}(\cdots f\left(w^1 x + b^1\right)\cdots) + b^{L-1}) + b^L) \tag{2}$$

Autoencoders: An autoencoder is trained on the Drebin dataset. The input features are first scaled using the standard scaler from the pre-processing module of scikit-learn. Different layers are utilized in this study's model-an input layer and three distinct hidden layers with variable nodes. The enable function for all stealth operations is ReLU, whereas the line enable function is employed for output operations. The collection of samples is conducted through the MSE function and optimization is handled by ADAM.

Table 1: SNN model results

No.	Dataset	Test loss	Test accuracy (%)
1	Drebin	0.98	90.59
2	Malgenome	0.57	96.00
3	Tuandromd	2.81	81.52
4	Krondroid	0.35	96.99

Table 2: FNN model results

No.	Dataset	Test loss	Test accuracy (%)
1	Drebin	0.62	66.66
2	Malgenome	0.63	65.92
3	Tuandromd	0.86	82.30
4	Krondroid	1.68	53.26

During training, the model is assessed at irregular intervals to guarantee ongoing progress. If the data loss in validation fails to repeat itself for three successive occurrences, then the training procedure will cease beforehand.

After undergoing complete training, the model's performance undergoes assessment by its testing on various data sets. The experimental loss is computed with the aid of the MSE function in the same process.

The findings of our study indicate that autoencoders have an efficient ability to reconstruct input data with less experimentation. This observation proposes that this particular model has the potential to recognize vulnerabilities within malware identification. The results of the autoencoder model are given in Table 3.

The encoder function can be written as:

$$h = f_{enc}(x) \tag{3}$$

The decoder function can be written as:

$$x' = f_{dec}(h)) \tag{4}$$

Gated recurrent units: The architecture of the GRU model accepts the input of the shape (X_train. shape as it is a linear model and the input is passed through the 64-unit GRU layer, then through the 32-unit dense layer. Batch normalization and ReLU activation functions are used to output the dense layer. Another density layer of 16 units was added, followed by batch normalization and ReLU activation function.

Finally, the system with a sigmoid function is added. The model is compiled using Binary Cross Entropy (BCE) loss and Adam optimizer. An accuracy test is also included to evaluate the effectiveness of the model. An early recall limit is defined to monitor for false positives and stop training if the loss does not improve after three periods.

The training model is evaluated in the test system. The results are given in Table 4 and it shows that the GRU model achieves high accuracy scores and demonstrates its effectiveness in predicting binary values in ordinal data. The model can be further enhanced by modifying hyperparameters or adding layers for more complex data and can be used in many applications such as natural language processing, sentiment analysis, text generation, and language translation, thus enabling enhanced human-computer interaction:

$$h_t = GRU\left(x_t, h_{(t-1)}\right) \tag{5}$$

Table 3: Autoencoder model results

No.	Dataset	Test loss	Test accuracy (%)
1	Drebin	0.23	83.15
2	Malgenome	0.40	90.52
3	Tuandromd	0.05	88.47
4	Krondroid	0.13	82.69

Federated Learning

The Federated Learning model was implemented using the PySyft 3.6 library in Python. PySyft is a popular open-source library for building privacy-preserving machine learning applications. In our implementation, a federated dataset that consists of four separate datasets is employed, each owned by a different client. Then a deep neural network is trained on the federated dataset using the FedAvg algorithm. This FedAvg algorithm involves training the model on each client's local dataset and then aggregating the model weights across all clients. This approach ensures that no individual client has access to the global model parameters and the model accuracy is not affected by the data held by any single client. The framework of the unfolded architecture is illustrated in the Fig. 5.

The weighted FedAvg is performed at the global server based on the following Equation:

$$w_t^z = \frac{1}{N_k} \Sigma w_{t-1}^m \beta_i \tag{6}$$

The $w_t^z$is the worldwide demonstration made at time t and $N_k$is the number of neighborhood models gotten at the worldwide server? It also shows the overall number of clients that have taken an interest in the FL and $w_{t-1}^m$are the local models gotten from all the clients at a time (t-1)? The β is the dynamic weight related to each local model received.

The dynamic weight β is naturally balanced based on the execution of each local model on the client's side, where the worldwide server begins with considering each local model to demonstrate equal capability and thus equally essential. The worldwide server then produces a global priority index containing each local model's weights. The weights are further balanced based on the performance of the local models.

The Distributed Weighted FedAvg (DWFedAvg) technique runs for a specified set of iterations. In each round, the global model is updated by aggregating the local models' weights and each client trains a local model using their data. The entire range of samples in the dataset for each client affects the scaling factor of the weights in the local models. The new weights for our global model are then calculated as the weighted average of the local models' weights, with the weights for each client determined by the client's performance on the test set. The FL model's result for 10 rounds with 10 clients is presented in Table 5. Also, Fig. 6 depicts the global accuracy and loss analysis for all 10 rounds.

Table 4:GRU model results

No.	Dataset	Test loss	Test accuracy (%)
1	Drebin	1.33	90.29
2	Malgenome	0.12	98.15
3	Tuandromd	0.45	96.7
4	Krondroid	1.03	90.89

Table 5: FL model results in 10 clients-10 rounds

No.	Dataset	Global _Acc	Global _loss	Global _F1	Global _precision
1	Drebin	0.95	0.59	0.93	0.94
2	Malgenome	0.96	0.59	0.94	0.97
3	Tuandromd	0.99	0.39	0.99	0.99
4	Krondroid	0.92	0.55	0.92	0.93

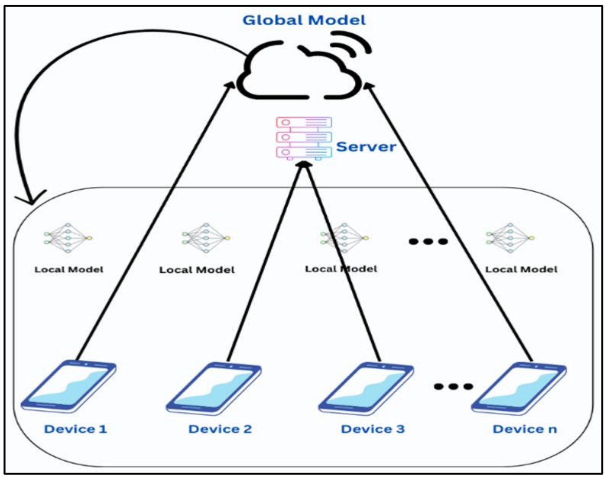

Fig. 5: Unfolded architecture framework

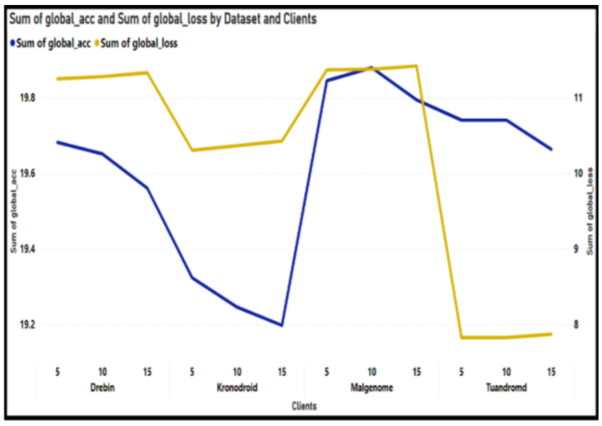

Fig. 6: Global accuracy and loss analysis (10 rounds)

Observations

Our research findings indicate that in classifying Android malware, deep learning models have been not as solid as federated learning algorithms. This study locates that education DL models call for a whole lot of records and might now and again over-fit training records, resulting in poor performance on new records. Federated learning, alternatively, showed more stability and potential for enhancing malware category accuracy and security on android devices.

However, some challenges were confronted for the duration of this research. Obtaining various and dependable records that accurately represented exclusive varieties of malware became one of the foremost barriers. Additionally, making sure that the models had been educated on balanced records, warding off biases, changed into another task.

Despite those challenges, our study indicates that FL can offer a robust and secure technique for the android malware class. By developing powerful and steady communique structures between gadgets, managing heterogeneous statistics resources, and dealing with model collections to make certain equity and accuracy, federated learning can help triumph over a number of the challenges that were faced during this research.

Results and Discussion

The results of our study showed that the accuracy of deep learning models varied depending on the dataset and model used. However, federated learning consistently provided an accuracy of 96% regardless of the dataset and model used. This suggests that federated learning can significantly improve the accuracy of deep learning models while maintaining the privacy and security of the data.

One of the key advantages of federated learning over traditional deep learning models is its ability to maintain stability in the face of changes in data distribution. federated learning utilizes multiple local models trained on individual devices, which reduces the impact of noisy or unrepresentative data on the overall model. In contrast, deep learning models can be prone to over-fitting training data, leading to reduced generalization performance on new data. Our findings suggest that integrated learning algorithms can provide a robust and secure method for Android malware classification.

Furthermore, our study highlights the advantages of federated learning over deep learning models in terms of security. By keeping the data on local devices and transmitting only model updates between devices, federated learning is less vulnerable to privacy breaches and cyber-attacks. This is a crucial advantage when dealing with sensitive or private data, as it allows for secure and accurate classification while maintaining data privacy.

There were several challenges during the study, such as obtaining diverse and reliable data that accurately represented different types of malware. Ensuring that the models were trained on balanced data and avoiding biases was another obstacle. Additionally, the limited processing power of mobile devices affected the training time and complexity of images, which was a challenge to overcome.

In conclusion, our study suggests that federated learning is a promising approach for training machine learning models on sensitive or private data. It provides high accuracy, stability, and security levels while maintaining data privacy, making it a suitable approach for applications that require secure and accurate classification of sensitive data. Future research should address the challenges that were faced in this study, such as obtaining diverse and reliable data and ensuring balanced data for training. Exploring other integrated learning algorithms and datasets could also further improve the accuracy and robustness of the classification models.

The evaluation of global accuracy and global loss across the three models, Federated Learning (FL), GRU, and SNN, reveals distinct trends and highlights the stability of the FL model. A bar chart (Figs. 7-8) depicting the performance of these models demonstrates that FL consistently maintains a high accuracy level across all the datasets, exhibiting minimal fluctuations. In contrast, both the GRU and SNN models exhibit significant fluctuations in accuracy and loss metrics, indicating less stability in their performance. This graphical representation underscores the superiority of the FL model in terms of stability, emphasizing its robustness in handling diverse datasets and maintaining consistent accuracy throughout the training process.

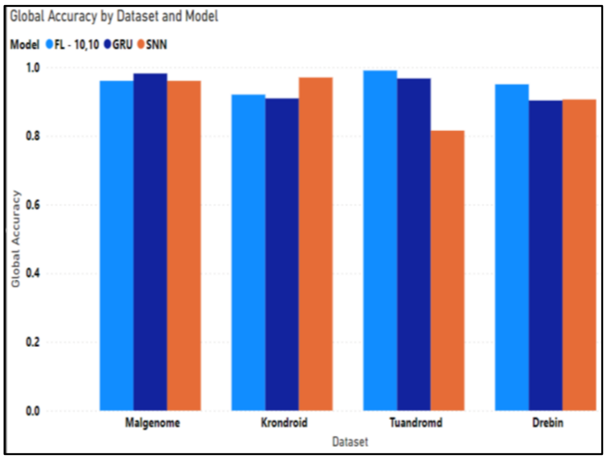

Fig. 7: Global accuracy between FL, GRU, SNN

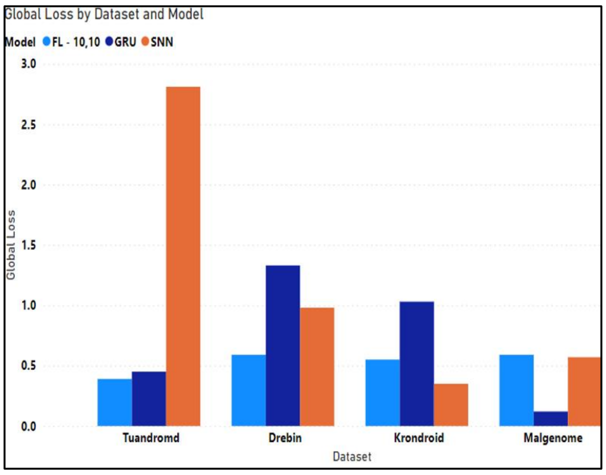

Fig. 8: Global loss between FL, GRU, SNN

Overall, the findings of our study highlight the potential of federated learning to improve the accuracy and security of deep learning models for Android malware classification. As mobile devices become increasingly ubiquitous, the need for secure and accurate classification of sensitive data on these devices will continue to grow. Federated learning is a promising solution to this problem, offering both accuracy and security while maintaining data privacy.

Conclusion

Our proposed approach of utilizing Federated Learning (FL) algorithms yielded outstanding results across four diverse datasets, including Drebin, Malgenome, Tuandromd, and Kronodroid. The experiments showcased a substantial improvement in accuracy, with FL algorithms achieving an impressive accuracy rate of 96% for all four datasets defeating all the other deep learning and machine learning algorithms. The presented research spearheaded the design and implementation of FL algorithms, conducted experiments, and analyzed results, while also outlining future research directions, emphasizing exploration of neural networks like CNNs and RNNs within FL algorithms for enhanced performance and this research played a pivotal role in conceptualizing and developing the experimental framework, contributing to the FL algorithms' success. Their specific contributions include an in-depth analysis of the inherent stability and security advantages of FL algorithms and outlining potential directions for future research, emphasizing advanced FL algorithms capable of handling heterogeneous data sources. Importantly, our study also revealed that FL algorithms possess inherent stability and security advantages over traditional models. By adopting a decentralized approach where data remains locally stored on individual devices and only model updates are exchanged, FL algorithms not only preserve the privacy and security of the data but also ensure the stability and robustness of the models themselves.

Our study highlights several potential areas for further research. For example, future research could investigate the use of other types of neural networks in FL algorithms, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). In addition, the research could focus on developing more advanced FL algorithms that can handle heterogeneous data sources and optimize models for performance and energy efficiency.

Overall, our findings suggest that federated learning has the potential to transform machine learning in a variety of disciplines. Our research study provides valuable insights that can guide the development of future FL applications and inspire researchers to explore the potential of this new approach hope that this research will contribute to the ongoing efforts to improve the accuracy, stability, and security of deep learning models.

Acknowledgment

Authors express their gratitude to Vellore Institute of Technology, Chennai, for providing the infrastructural facilities to carry out this research work. We would like to express our gratitude towards the unknown potential reviewers who have agreed to review this study and provided valuable suggestions to improve the quality of the paper.

Funding Information

The authors received no financial support for the research, authorship, and/or publication of this article.

Author's Contributions

Narayan Subramanian and Mithin Jain Shaan: Designed and implemented the software, performed the validation, and drafted the manuscript.

Logesh Ravi and Malathi Devarajan: Participated in the designed and execution of the study, and revised and edited the manuscript. Logesh Ravi also supervised and coordinated the project.

Tanupriya Choudhury: Conducted the data analysis, collected and curated the data, and created the visualizations.

Ketan Kotecha and Subramaniyaswamy Vairavasundaram: Oversaw the data analysis, provided the resources, and supervised and coordinated the project. They also revised and edited the manuscript. All authors read and approved the final manuscript.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and that no ethical issues are involved.

Competing Interests

The authors declare that they have no competing interests.

References

Albakri, A., Alhayan, F., Alturki, N., Ahamed, S., & Shamsudheen, S. (2023). Metaheuristics with deep learning model for cybersecurity and android malware detection and classification. *Applied Sciences*, *13*(4), 2172. https://doi.org/10.3390/app13042172

Almahmoud, M., Alzu'bi, D., & Yaseen, Q. (2021). ReDroidDet: android malware detection based on recurrent neural network. *Procedia Computer Science*, *184*, 841-846. https://doi.org/10.1016/j.procs.2021.03.105

Aurangzeb, S., & Aleem, M. (2023). Evaluation and classification of obfuscated Android malware through deep learning using ensemble voting mechanism. *Scientific Reports*, *13*(1), 3093. https://doi.org/10.1038/s41598-023-30028-w

Bovenzi, G., Cerasuolo, F., Montieri, A., Nascita, A., Persico, V., & Pescapé, A. (2022). A comparison of machine and deep learning models for detection and classification of android malware traffic. *In 2022 IEEE Symposium on Computers and Communications (ISCC)* 1-6. IEEE. https://doi.org/10.1109/ISCC55528.2022.9912986

Chen, J., Zhao, Z., Chen, X., Cai, S., Yin, S., & Song, L. (2022). A novel classification approach for android malware based on feature fusion and natural language processing. *In Proceedings of the 13th Asia-Pacific Symposium on Internet ware* 28-36. https://doi.org/10.1145/3545258.3545278

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273-297. https://link.springer.com/article/10.1007/BF00994018

Rey, V., Sánchez, P. M. S., Celdrán, A. H., & Bovet, G. (2022). Federated learning for malware detection in IoT devices. *Computer Networks*, 204, 108693.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139-144. https://doi.org/10.1145/3422622

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv Preprint ArXiv:1704.04861*. https://doi.org/10.48550/arXiv.1704.04861

Jebin Bose, S., & Kalaiselvi, R. (2023). An optimal detection of android malware using dynamic attention-based LSTM classifier. *Journal of Intelligent and Fuzzy Systems*, *44*(1), 1425-1438. https://doi.org/10.3233/JIFS-220828

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*. https://doi.org/10.1145/3065386

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. https://doi.org/10.1007/BF00994018

Lu, T., Du, Y., Ouyang, L., Chen, Q., & Wang, X. (2020). Android malware detection based on a hybrid deep learning model. *Security and Communication Networks*, 2020, 1-11. https://doi.org/10.1155/2020/8863617

Marwaha, A., Malik, R. Q., Beram, S. M., Rizwan, A., Kishore, K. H., Thakur, D., ... & Shabaz, M. (2023). Visualisation-based binary classification of android malware using vgg16. *IET Software*. https://doi.org/10.1049/sfw2.12094

Mahindru, A., & Arora, H. (2022, November). Dnndroid: Android malware detection framework based on federated learning and edge computing. In *International Conference on Advancements in Smart Computing and Information Security (pp. 96-107)*. Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-23095-0_7

Nguyen, C. D., Khoa, N. H., Doan, K. N. D., & Cam, N. T. (2023). Android malware category and family classification using static analysis. *In 2023 International Conference on Information Networking (ICOIN)* 162-167. IEEE. https://doi.org/10.1109/ICOIN56518.2023.10049039

Pektaş, A., & Acarman, T. (2020). Deep learning for effective Android malware detection using API call graph embeddings. *Soft Computing*, *24*, 1027-1043.

Pei, X., Deng, X., Tian, S., Zhang, L., & Xue, K. (2022). A knowledge transfer-based semi-supervised federated learning for IoT malware detection. *IEEE Transactions on Dependable and Secure Computing*. https://doi.org/10.1109/TDSC.2022.3173664

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *ArXiv Preprint ArXiv:1511.06434*. https://doi.org/10.48550/arXiv.1511.06434

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

Ullah, F., Srivastava, G., & Ullah, S. (2022). A malware detection system using a hybrid approach of multi-heads attention-based control flow traces and image visualization. *Journal of Cloud Computing*, *11*(1), 1-21 https://doi.org/10.1186/s13677-022-00349-8

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. *Advances in Neural Information Processing Systems*, *30*. shttps://proceedings.neurips.cc/paper_files/paper/2017/hash/81b3833e2504647f9d794f7d7b9bf341-Abstract.html

Yadav, P., Menon, N., Ravi, V., Vishvanathan, S., & Pham, T. D. (2022). A two-stage deep learning framework for image-based android malware detection and variant classification. *Computational Intelligence*, *38*(5), 1748-1771. https://doi.org/10.1111/coin.12532

Zhang, X., Zhou, Y., Pei, S., Zhuge, J., & Chen, J. (2020). Adversarial examples detection for XSS attacks based on generative adversarial networks. IEEE Access, 8, 10989-10996. https://doi.org/10.1109/ACCESS.2020.2965184