Arabian Journal for Science and Engineering
https://doi.org/10.1007/s13369-025-10043-x

RESEARCH ARTICLE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

FEDetect: A Federated Learning-Based Malware Detection and Classification Using Deep Neural Network Algorithms

Zeki Çıplak[1] · Kazım Yıldız[2] · Şahsene Altınkaya[3]

Received: 5 October 2024 / Accepted: 10 February 2025

Abstract
The growing importance of data security in modern information systems extends beyond the preventing malicious software and includes the critical topic of data privacy. Centralized data processing in traditional machine learning methods presents significant challenges, including greater risk of data breaches and attacks on centralized systems. This study addresses the critical issue of maintaining data privacy while obtaining effective malware detection and classification. The motivation stems from the growing requirement for robust and privacy-preserving machine learning methodologies in response to rising threats to centralized data systems. Federated learning offers a novel solution that eliminates the requirement for centralized data collecting while preserving privacy. In this paper, we investigate the performance of federated learning-based models and compare them classic non-federated approaches. Using the CIC-MalMem-2022 dataset, we built 22 models with feedforward neural networks and long short-term memory methods, including four non-federated models. The results show that federated learning performed outstanding performance with an accuracy of 0.999 in binary classification and 0.845 in multiclass classification, despite different numbers of users. This study contributes significantly to understanding the practical implementation and impact of federated learning. By examining the impact of various factors on classification performance, we highlight the potential of federated learning as a privacy-preserving alternative to centralized machine learning methods, filling a major gap in the field of secure data processing.

Keywords Federated learning · Malware detection · Malware classification · Feedforward neural network · Long short-term memory · Data privacy

1 Introduction

Cyber attackers, with the aid of software known as malicious software or simply malware [11], can gain access to users' computers or mobile devices and steal many personal confidential information or render user devices unusable. It is conceivable to see a comparable rise in cyberattacks as internet usage rises in the computing world [99]. As a result, malware detection is a critical issue in cybersecurity that has received significant attention, and research continues in this field for an effective solution.

Traditional malware detection approaches are based on a dataset stored on a central server. This strategy offers a substantial risk to data privacy because users' information is transferred to a central server. For decades, several research have been conducted in many nations throughout the world to determine what can be done to improve data privacy and standardize it. It has been emphasized that this risk has global importance and needs to be addressed seriously [2, 29, 33,

Kazım Yıldız and Şahsene Altınkaya have contributed equally to this work.

✉ Zeki Çıplak
zeki.ciplak@gedik.edu.tr

Kazım Yıldız
kazim.yildiz@marmara.edu.tr

Şahsene Altınkaya
sahsene.altinkaya@utu.fi

1 Department of Computer Technologies, Gedik Vocational School, Istanbul Gedik University, 34909 Pendik, Istanbul, Türkiye

2 Department of Computer Engineering, Faculty of Technology, Marmara University, 34854 Maltepe, Istanbul, Türkiye

3 Department of Mathematics and Statistics, University of Turku, 20014 Turku, Finland

Published online: 17 March 2025

77, 83]. On the other hand, collecting and analyzing data on a central server takes a long time and consumes a lot of server capacity. This is because users must send all of their data to the central server in order for the model to be developed. In the federated learning (FL) approach, users do not need to send all their data to a central server. This strategy offers a good solution for both increased security and data protection. Also known as cooperative learning, FL was first proposed by Google in 2016 [61, 106]. In recent years, numerous studies have shown that federated learning is becoming a popular machine learning strategy [16, 21, 52, 100, 105, 108]. FL operates on the principle that data continues to be hosted on local devices that produce that data, and only the parameters of the models produced on the devices are sent to a central server (FL server). In this regard, while FL protects data privacy, it also enables machine learning models to be generated more correctly and quickly on the go. Devices in the FL-based system share a major task among themselves, as opposed to the conventional approach. That is, each local device in the FL system develops a specific element of a bigger model, which are then integrated at the FL server. Thus, the FL strategy eliminates the time and resource loss that happens when all work is done at a single location.

Another important point is that centralized data processing methods include limitations such as the requirement for high bandwidth, time loss, and risks to data privacy. However, the originality of this research comes in actually proving how the federated learning strategy addresses these shortcomings. For example, the scalability of FL in response to an increase in the number of users has been analyzed along with its data privacy advantages. This study aims to fill gaps in the literature by focusing on the technical details of local FL simulations that are lacking.

From this point of view, it is predicted that the use of federated learning method in malware detection will provide great benefits to users in many aspects, including data security. With the malware detection application using the FL approach, users can detect threats on their own devices and contribute to the development of a larger model. Users can create their own models and send the model parameters to a central FL server. The FL server can then process local models containing information about threats from different user devices and develop a global model that can predict and block most of the threats that may occur in this area. The parameters of the global model can be sent back to all local devices in the FL system. Thus, each local device user gains the experience (without seeing the data) of all other users.

This study focuses on the development of the FL system (FEDetect) and users' capacity to detect malware on their own data while maintaining data privacy. For this purpose, a simulation representing all local users in the FEDetect and the FL server, operating on a basic device, has been designed. The simulation first creates a local model for each local device and then merges all created local models (replacing the FL server). The local user devices and the FL server are created virtually. There is no communication between them. All operations have been performed on a main computer and symbolically.

1.1 Contributions and organization

This work is important for testing a FL system in a simulation environment before adapting it to a real-world scenario. It provides a basis for understanding the possible performance and challenges of the systems. Local computation without inter-station communication under simulation conditions can be considered as an innovative way to study different aspects of federated learning.

In terms of testing architectures for deep learning models to be built in the FL system and, if necessary, optimizing the number of hidden layers or neurons in these architectures, it can provide better preparation before moving to a real-world scenario.

When the technical details of the system used (processor, RAM, etc.) and the results obtained are evaluated, it can be considered that better results (especially in terms of time) can be obtained with a technically more powerful system. For this purpose, it is a study that can be used as a basis for comparison. The study analyzes the effect of different numbers of virtual computers (8, 16, 32, 64 and 128) on federative learning processes. This analysis provides a new perspective on the computational efficiency and scalability of federative learning.

In this study, 'reaching a target accuracy value' was used as the criteria for stopping the federative system. By determining the best accuracy value before the target accuracy value, models that reach a value close to the target accuracy in less time are generated and their effects are analyzed (early round stopping).

Since it includes a pseudo code solution for local simulations of the FL system, it also provides a technical perspective on how federated learning can be applied in local simulation. It is a new contribution to the limited literature on local simulations of federated learning systems. By filling the gaps in this area, it can serve as a basis for future research.

In the study, both malware detection (binary classification—BC) and malware type detection (multiclass classification—MC) have been performed. The second section of the article includes a literature review of studies in the same field. The third section contains information about the CIC-MalMem-2022 dataset used in the study [17]. Additionally, details about the preprocessing performed on this dataset are provided. The fourth section provides information about the proposed methodology and the established simulation. In addition, classification algorithms used in the study, model architectures, parameters of the models, pseudo codes of the basic part of the program written for the study, used libraries, and other technological details are included. The fifth section discusses the results of the study and compares them with previous studies. The final section discusses the challenges of the study and provides forecasts and recommendations on what different things could be done in addition to this study in the potential future and how the study could be advanced.

2 Related Works

In this research paper, previous studies are evaluated in two groups. The first group includes studies that use the CIC-MalMem-2022 dataset without the FL approach and detect malware. The most important studies worth comparing in terms of similarity are in this group. In the second one, there are studies that detect malware with the FL approach without CIC-MalMem-2022 dataset.

2.1 CIC-MalMem-2022 Dataset without Federated Learning

Louk and Tama [55] have compared various tree-based community learning methods in their malware detection study using many datasets such as BODMAS, Kaggle, and CIC-MalMem-2022. Several tree-based machine learning algorithms have been tested for each dataset. For the CIC-MalMem-2022 dataset, many algorithms such as CatBoost (CB), XGBoost (XGB), LightGBM (LGBM), random forest (RF), and gradient boosting model (GBM) have been tested, with the best result being obtained by the RF algorithm. Accuracy, precision, and F1 metrics achieved 100%, and recall achieved 99.99%.

Talukder et al. [96] have proposed a hybrid model by combining machine learning and deep learning techniques in their study on network attack detection systems. Data balancing with the synthetic minority oversampling technique (SMOTE) and dominant feature extraction with XGB have been achieved. In the study using the KDDCUP'99 [97] and CIC-MalMem-2022 datasets, they achieved success rates of 99.99% and 100%, respectively, without encountering problems like overfitting, type−1, and type−2.

Smith et al. [91] have conducted malware detection using supervised and unsupervised learning algorithms. Two separate datasets, Malware-Exploratory [12] and CIC-MalMem-2022, have been used for this purpose. They used clustering algorithms like K-means, density-based spatial clustering of applications with noise (DBSCAN), and Gaussian mixture model (GMM), and a total of seven classification algorithms including decision tree (DT), KNeighbors (kNN), RF, stochastic gradient descent (SGD), Ada Boost (AB), Extra Trees (ET), Gaussian Naive Bayes (GNB). The Ada Boost algorithm gave the best results for both datasets, achieving accuracy rates of 100% for Malware-Exploratory and 99.99% for CIC-MalMem-2022.

Mezina et al. [64] have conducted a study to detect malware hidden in memory. In the study using the CIC-MalMem-2022 dataset, traditional machine learning methods were first tested, and then an extended convolutional neural network model was proposed. The accuracy rate of all methods in malware detection was found to be approximately 0.99. The best of these methods was found to be RF. The best method for malware type detection was determined to be the proposed neural network architecture, with an accuracy rate of 0.83 for the neural network used for malware classification.

Dener et al. [24] have conducted a malware detection study based on the use of memory data. In the study where various machine learning and deep learning approaches were tried, big data platforms were also utilized. The CIC-MalMem-2022 dataset was used on the Google Colab platform with the PySpark big data management tool. Binary classification was performed using algorithms such as RF, DT, Gradient Boosted Tree (GBT), logistic regression (LR), Naive Bayes (NB), linear vector support machine (LVSM), multilayer perceptron (MLP), deep feed forward neural network (DFFNN), and long short-term memory (LSTM). Accuracy rates of 99.97 for LR, 99.94 for GBT, and 98.41 for NB were achieved in malware detection.

Ghazi and Raghava [32] have conducted a study to detect malicious software hidden in cloud environments. In the study where the CIC-MalMem-2022 dataset was used, the primary aim was to overcome the difficulties in feature selection. For this purpose, algorithms such as binary bat algorithm (BBA), cuckoo search algorithm (CSA), mayfly algorithm (MA), and particle swarm optimization (PSO) have been used. After feature selection, benign–malignant records were classified with machine learning algorithms kNN, RF, and support vector machine (SVM). The calculated metrics included accuracy, precision, and recall. In experiments using the RF algorithm, the best result was obtained with the MA feature selection algorithm. A success of approximately 0.99 was achieved in all calculated metrics. In experiments using kNN and SVM machine learning algorithms, the best results were obtained with the PSO feature selection algorithm. A success rate of 0.99 was again reached in all calculated metrics for both algorithms.

Naeem et al. [67] have developed a visual-based system in their study using many datasets. The study proposes a completely platform-independent scheme. This scheme involves extracting a file from the dumps of the operation memory and converting it to an image. A community model has been created with records taken in sufficient quantities from Dumpware10, CIC-MalMem-2022, and real-world dataset of Android applications. The created model combined the prediction outputs over the convolutional neural network (CNN) and presented them as a learning input to MLP. The outputs obtained from MLP were used as inputs in many machine learning algorithms. According to the findings obtained at the end of the study, in malware detection on the Windows operating system, the proposed scheme was found to achieve an accuracy rate of 99.1. This rate increases to 99.8 for hidden malware analysis on the same operating system. An accuracy rate of 94.3 was detected for Android.

2.2 Federated Learning Without CIC-MalMem-2022 Dataset

Lin and Huang [53] have conducted a malware classification study with a dataset provided by VirusTotal, containing 10,907 records. In the study where SVM and LSTM machine learning models were configured with and without federated learning, a total of four models have been tested. The highest accuracy rate achieved is 0.9167.

Rey et al. [78], in their IoT malware detection study for supervised and unsupervised learning models, have demonstrated that the federated learning approach maintains data privacy without compromising the model's performance. The N-BaIoT dataset [62], which models network traffic, has been used in the study. A success of over 97% has been achieved in all calculated metrics (Accuracy, TPN, TPR) in the study where many different models were tested.

Hsu et al. [42], have conducted malware detection using the SVM algorithm and FL approach with 87,182 APK files collected from the Opera Mobile Store. Applications in a total of 12 categories, such as Entertainment, Health, Games, Travel, and E-book, have been examined. In the study where many different scenarios were tested, federated learning approach with SVM involving five users and the classic approach where data are centralized were compared. The accuracy rate of the federated approach was lower at 93.45%, compared to the classic approach at 94.05%. Despite being lower in terms of accuracy rates, the federated learning approach has shown higher values in precision, recall, fpr, and f-measure metrics compared to the classic approach.

Galvez et al. [30] have shown that they achieved more efficient results with the federated learning approach compared to classic classification using the less is more (LiM) malicious software classification framework. In the study, 200 users with 25K clean applications and 25K malicious applications participated in the test. The MaMaDroid dataset [58] was used in the study, and the F1 score was calculated as 95%.

Jiang et al. [44] have proposed a new classification scheme called FedHGCDroid in their study, classifying malicious software on Android devices based on federated learning. Initially for malicious software classification, they used HGCDroid, which extracts the behavior features of malware software, using CNN and graphic neural network (GNN). In tests with the Androzoo dataset [6], it was found that FedHGCDroid achieved more adaptability and higher accuracy compared to other methods. Examining the success metrics, they performed malware detection with an accuracy rate of 91.3% and an F1 score of 91.25%.

Taheri et al. [95] have proposed an Android architecture called Fed-IIoT for malware detection, consisting of two components, participant and server side, with a federated learning approach. In the study where three different datasets were used, the best result was obtained with the Genome dataset [109], and the accuracy rates of the used FedGAN models (FedGAN-BM and FedGAN-BK) were measured as 89.51% and 93.24%.

Pei et al. [71] have proposed a framework called FedMalDE based on federated learning for semi-supervised detection of malicious software in IoT devices. Additionally, in this study, a specially developed subgraph aggregated capsule network (SACN) model was used to healthily perceive malicious behaviors. The performance of the SACN model was compared with five different classifiers: Feedforward neural network (FNN), CNN, gated recurrent unit (GRU), LSTM, and capsule network (CAP). DREBIN [8], AndroZoo, VirusShare [102] and MalDroid [56] datasets were used in the study. The best results in the study were obtained with the DREBIN dataset, with a TPR metric of 97.5 and an *F*-score of 97.64.

A comprehensive overview of recent studies in both non-federated and federated learning techniques as applied to malware detection and analysis is presented in Table 1. It summarizes key studies, highlighting the datasets used, the techniques and algorithms employed, and the main findings, thereby providing a clear comparison of the effectiveness and innovation in the field.

3 Material

In this section, detailed information about the data set is given, and the preprocessing done on the data set is also mentioned.

3.1 Dataset

The dataset used in this study is CIC-MalMem-2022 prepared by the Canadian Cyber Security Institute [17]. This dataset is one of the most up-to-date sources that can be used to detect malware.

The CIC-MalMem-2022 dataset is an important resource that reflects as close to reality as possible the effects on memory of the major malware commonly used in the real world. It represents an accurate representation of what a user would run on a computer environment during a malware attack. In the dataset of 58,296 records, there are 29,298 records that are balanced between benign and malicious (malware). In

Table 1 Comparison of recent flow studies on non-federated and federated learning techniques

Ref	Dataset	Technique	Algorithm	Summary
Louk and Tama [55]	CIC-MalMem-2022, BODMAS, Kaggle	Not federated	CatBoost, XGBoost, LightGBM, random forest, gradient boosting model	Tree-based machine learning methods tested; RF achieved the best result with near-perfect metrics
Talukder et al. [96]	CIC-MalMem-2022, KDDCUP-99	Not federated	SMOTE, XGBoost, Hybrid ML-DL model	Hybrid model combining ML and DL techniques; achieved 100% accuracy for CIC-MalMem-2022
Smith et al. [91]	CIC-MalMem-2022, Malware-Exploratory	Not federated	K-Means, DBSCAN, Gaussian mixture model, AdaBoost, decision tree, etc.	AdaBoost algorithm performed best, achieving 99.99% accuracy for CIC-MalMem-2022
Mezina et al. [64]	CIC-MalMem-2022	Not Federated	Extended CNN, random forest	Extended CNN proposed for malware type detection; RF excelled in overall malware detection
Dener et al. [24]	CIC-MalMem-2022	Not Federated	RF, DT, Gradient Boosted Tree, logistic regression, LSTM, etc.	Big data platforms used; achieved accuracy of 99.97% with logistic regression
Ghazi and Raghava [32]	CIC-MalMem-2022	Not federated	Binary bat algorithm, PSO, RF, kNN	PSO achieved the best results across all metrics
Naeem et al. [67]	Dumpware10, CIC-MalMem-2022, Android Apps	Not Federated	CNN, MLP	Platform-independent visual-based system proposed; achieved 99.8% accuracy for hidden malware analysis

Table 1 continued

Ref	Dataset	Technique	Algorithm	Summary
Lin and Huang [53]	VirusTotal	Federated	SVM, LSTM	Four models tested; federated learning achieved 91.67% accuracy
Rey et al. [78]	N-BaIoT	Federated	Supervised and unsupervised models	Federated approach maintained privacy and achieved over 97% accuracy
Hsu et al. [42]	Opera mobile store	Federated	SVM	Compared federated and centralized approaches; federated showed superior precision and recall
Galvez et al. [30]	MaMaDroid	Federated	Less is more framework	Federated approach achieved 95% F1 score, outperforming traditional methods
Jiang et al. [44]	AndroZoo	Federated	FedHGCDroid (CNN, GNN)	Proposed FedHGCDroid; achieved 91.3% accuracy and superior adaptability
Taheri et al. [95]	Genome	Federated	FedGAN-BM, FedGAN-BK	Proposed Fed-IIoT architecture; achieved up to 93.24% accuracy
Pei et al. [71]	DREBIN, AndroZoo, VirusShare, MalDroid	Federated	Subgraph aggregated capsule network	SACN model achieved 97.64% F1 score with federated learning
Shafi et al. [84]	CIC-MalMem-2022	Federated	SVM	High performance with 99% accuracy in malware detection for smart systems

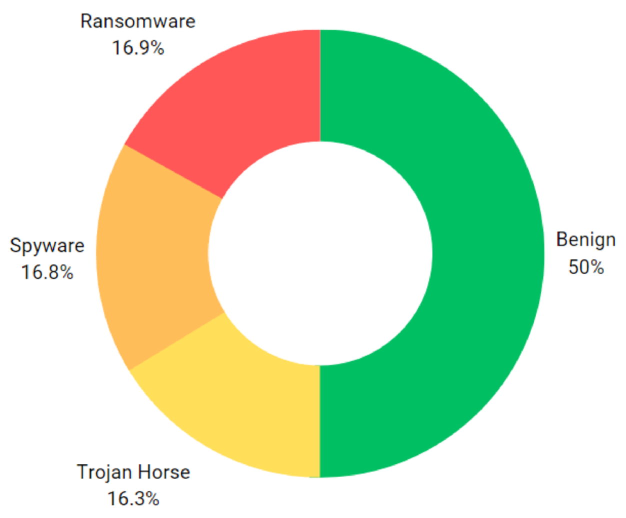

Fig. 1 Malware types of CIC-MalMem-2022 dataset

addition, the data marked as malicious were categorized into three different malware categories (Trojan Horse [46], Spyware [93], ransomware [69, 79]). The proportions of the three categories in the whole dataset are given in Fig. 1.

There are 57 attributes in total in the CIC-MalMem-2022 dataset. These attributes generally include information about the processes running in the operating system, the current memory status on the device, sessions, and the structures (handlers) created and called by the operating system to perform a certain function [27, 50]. Some of the models created in this study use this information to detect malware, while others try to detect the type of malware.

3.2 Data Preprocessing

The CIC-MalMem-2022 dataset has undergone some preprocessing before being used in the study. The aim of these preprocessing was to obtain two datasets that could be used separately for binary and multiclassification. First of all, it was considered to remove the columns with all the same values from the original dataset. Because having all the same values of an attribute does not provide any benefit to the machine learning model to be created. For this reason, the attributes removed from the dataset are with ID 5, 11 and 52. With this process, the total number of attributes in the dataset decreased to 54. Some basic data analysis checks were then performed, such as whether there were any data with an infinite value in the dataset and whether there was a missing value row.

Then, the unique numbers and other malware-related details in the Category column values were removed, ensuring that only the names of malware families are visible in the relevant column. For example, a value in the Category column, such as "Ransomware-Shade-fa03be3078d1b9840f06745f160eb..." has been rearranged to be just "ransomware". The "benign" values in the Class attribute have been converted to 0 (zero), and the Malware values have been converted to 1 (one). At this point, the original dataset has been modified in many ways. However, this latest, slightly modified version of the original dataset has been used in subsequent processes. A copy of the slightly modified original dataset has been created to form the dataset to be used for binary classification. The dataset to be used for binary classification is briefly referred to as the binary dataset (BD). In binary classification, the Class attribute has been used as the target variable, and the Category attribute has been removed from BD. The reason for this process can be explained as follows. The category attribute, with its values, is the attribute that most affects the target variable. The Category variable contains values like "benign", "ransomware", "Trojan Horse", and "Spyware", and a to-be-created malware detection model could decide whether the data in the relevant row are "benign" or "Malware" just by evaluating the Category variable. Such a situation would cause the loss of importance of all other attributes. For this reason, to allow other attributes to also affect the model and to create a healthier malware detection model, the Category variable has been removed from BD.

Table 2 Benign and malware data amounts in binary dataset

Class	Data amount
Benign (0)	29,298
Malware (1)	29,298

As a result, the total number of variables in BD has been reduced to 53. Then, excluding the target variable, all rows related to the remaining 52 independent variables have been subjected to Z-Score Normalization (Standardization). It is a known fact that standardization processes performed on datasets generally have an enhancing effect on model performances [4, 15, 86]. In this way, BD has been prepared. The amounts of data labeled as "benign" and "Malware" in the Class column within BD are shown in Table 2.

In order to create the dataset to be used for multiclass classification, a copy of the original dataset, slightly modified above, was created. Recall that this dataset contains 54 attributes. The dataset to be used in multiclass classification is called multiclass dataset (MD). Just as the Category column was removed from the data set when creating the BD, the Class column was removed from the data set when creating the MD. The Class variable was removed from the MD as it contains direct information about whether the data row is benign or malicious (malware). In real world conditions, in the natural state of data that is considered a malware attack, there is no labeling of whether the data are malware or not. Finally, as in BD, Standardization was applied to all variables

Table 3 Labeled data amounts in multiclass dataset

Category	Data amount
Benign (0)	29,298
Ransomware (1)	9791
Spyware (2)	10,020
Trojan Horse (3)	9487

except the target variable. In this way, the MD was ready for use.

In the final case, both the BD and MD contain 52 independent variables. BD was used in binary classification models for estimating the value of the class variable. MD was used in multiclass classification models for estimating the value of the category variable. The data amounts of the tags in the Category column in the MD are given in Table 3.

Data preprocessing and all subsequent steps are summarized in Fig. 2

4 Methodology

In this section, model architectures, information about simulation, tools, libraries, system features and algorithms are detailed.

4.1 Deep Learning Algorithms

In this study, a total of 22 basic models in 4 different architectures are proposed using FNN and LSTM for binary and multiclass classification. Apart from the basic models, variants of some models were also created for experimental purposes and their results were also used in the study.

4.1.1 Feedforward Neural Network (FNN)

It is a type of artificial neural network often used in machine learning (ML) and artificial intelligence (AI) applications. It is often used in clustering and classification applications. FNN is an artificial neural network with hidden layers in addition to input and output layers [39, 43, 74, 81, 82]. FNNs make it possible to solve complex relationships between variables in a dataset.

4.1.2 Long Short-Term Memory (LSTM)

LSTM is a more advanced version of RNN (recurrent neural network), a type of artificial neural network family and has been popularized in many studies in recent years [3, 19, 20, 36, 45, 75, 90]. Both structures are used in machine learning models where sequential data are preferred, such as time series and natural language processing. LSTM is a deep learning algorithm that can also be used on non-sequential data, as in this study. The fact that LSTM is a powerful and flexible tool for different types of problems can be seen as one of the reasons for its popularization [10, 31, 35, 88]. RNN has the ability to better learn the relationships between items that are close to each other in sequential data. LSTM, on the other hand, can also handle relationships between more distant items. In LSTM, there are three special gates: forget, input and output gates. With these gates, LSTM is able to learn relations with long dependencies. LSTM has a more complex architecture than RNN. There are more parameters in LSTM, which requires more computational power and training time [57].

4.2 Proposed Architectures

Two of the four types of architectures used in the study are based on FNN [49], and the other two are based on LSTM [41]. Before addressing the model architectures specifically, some of their general features can be mentioned. For example, it is important which activation function to choose in an architecture. The good performance of any activation function often depends on the application and the data. In particular, the model performance of a neural network is improved by choosing the right activation function. There is no general rule that a particular activation function is always more effective. Instead, there is a trial-and-error process where various activation functions are tested to determine which one works best for a given problem or dataset [63, 87, 94].

FNN allows for the rapid processing of data due to its flat structure, while LSTM is known for its superior performance particularly with time-dependent data. For complex and time-dependent issues such as malware detection, LSTM is an effective choice due to its sequence-based modeling capacity. Additionally, the successful use of these two models in a federated learning environment is also supported by previous studies [16, 52, 105].

In this study, after a series of tests for both binary classification and multiclass, the activation functions that were found to give the best results were chosen. Rectified linear unit (ReLU) activation function is used in all architectures used for binary classification and hyperbolic tangent (Tanh) is used in all architectures used for multiclass classification. ReLU is used to calculate the output of each neuron in its layer. ReLU returns a value based on the input value and returns 0 (zero) if the input value is below zero. Otherwise, it returns the input value itself [37]. Tanh is one of the hyperbolic counterparts of trigonometric functions. The tanh function takes values between -1 and $+1$, which is similar to the sine and cosine functions taking values between -1 and $+1$. This limitation between axes has made the tanh function popular to be used as an activation function, especially in

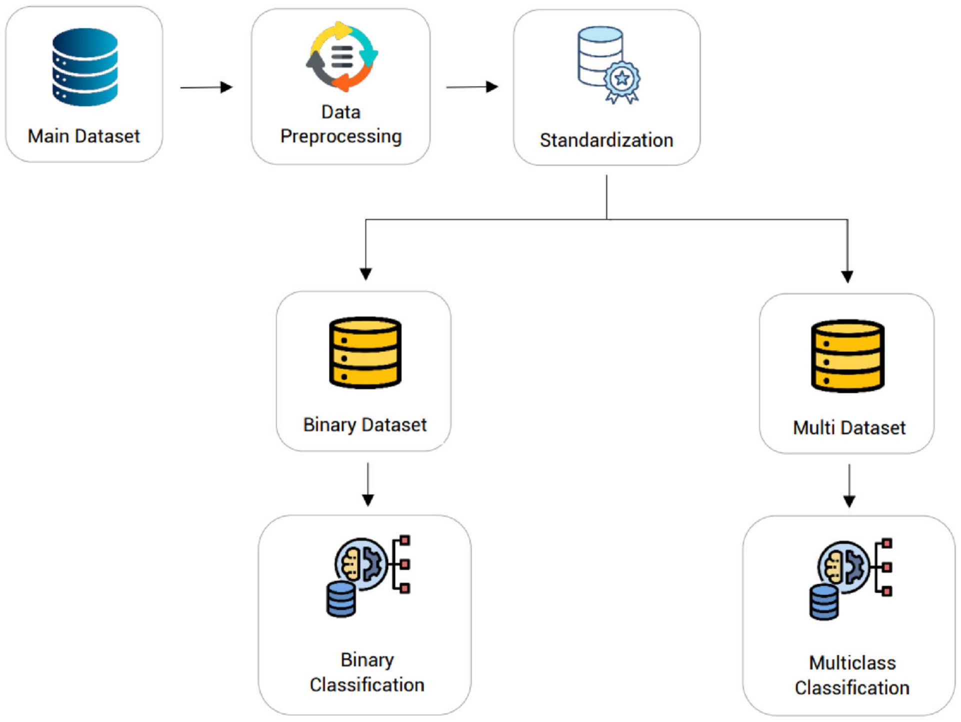

Fig. 2 Data preprocessing, standardization process and classification according to the created datasets

neural networks and machine learning. By normalizing the data, Tanh allows the model to be trained faster and more effectively, enabling more complex and more accurate predictions to be made [22, 25, 76]. In addition to activation functions, determining the number of neurons and layers is also an important parameter that should be taken into account when creating a neural network architecture. In the deep learning literature, there are many studies conducted in the past on determining the optimum number of neurons and layers [7, 103, 104]. In this study, a series of tests were carried out to determine the number of neurons and layers. In the architectures constructed after these tests, care was taken to use as few layers and neurons as possible so as not to affect the accuracy values. Another reason for wanting to minimize the number of neurons and layers as much as possible is that the processor capacities of internet-connected devices in the federated learning system may not be as high as those of personal computers. The training process of the FNN or LSTM architecture to be installed should be completed as soon as possible. This will only be possible with a non-complex neural network. Thus a local device in the federated system with lower computing power will be able to perform its duty within the system properly. In the subsections presented below, preliminary information is given about the architectures in the study and the models created from these architectures. The findings obtained after these models were trained were evaluated in the results and discussion section.

4.2.1 Feedforward Neural Network for Binary Classification (FNN-BC)

It is a feedforward neural network architecture used to solve the binary classification problem. FNN-BC has two hidden layers after the input layer and one output layer. The input layer of FNN-BC receives 52 input features and generates 10 output neurons. In the input layer and two hidden layers, ReLU is used as the activation function. The ReLU activation function is used to calculate the output of each neuron in this layer. The subsequent hidden layers also have 10 inputs and 10 outputs. The last layer in the architecture takes the 10 neurons from the last hidden layer as input and produces 1 output neuron at the end. The D3.js library was used to draw the figure of this architecture and the figures of the other three architectures described below [13]. Figure 3 shows the FNN-BC architecture.

Behind each hidden layer in the FNN-BC architecture, there is a dropout layer. If dropout layers are included, it can be said that there are six layers in total in this architecture. The purpose of the use of the dropout layer is to prevent overfitting and set the activations of randomly selected neurons to zero with a specified percentage (0.5 in this architecture) [9, 92]. Since FNN-BC will be used for binary classification, there is 1 neuron in the output layer. Sigmoid function is used as the activation function in the output layer. This function takes any real number as input and ensures that the result is 0 or 1. It is used in cases where the output needs to be interpreted probabilistically. In this way, binary classi-

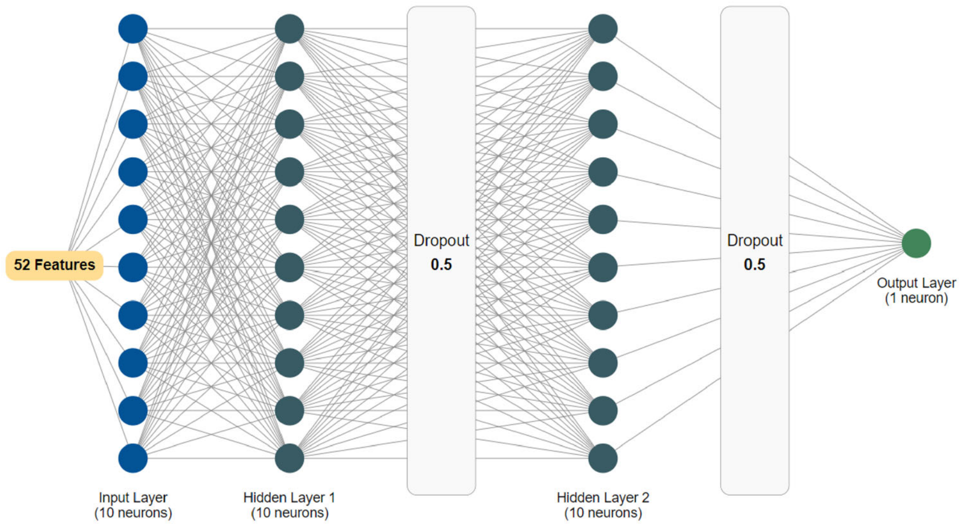

Fig. 3 Layers and Neurons of FNN-BC Architecture

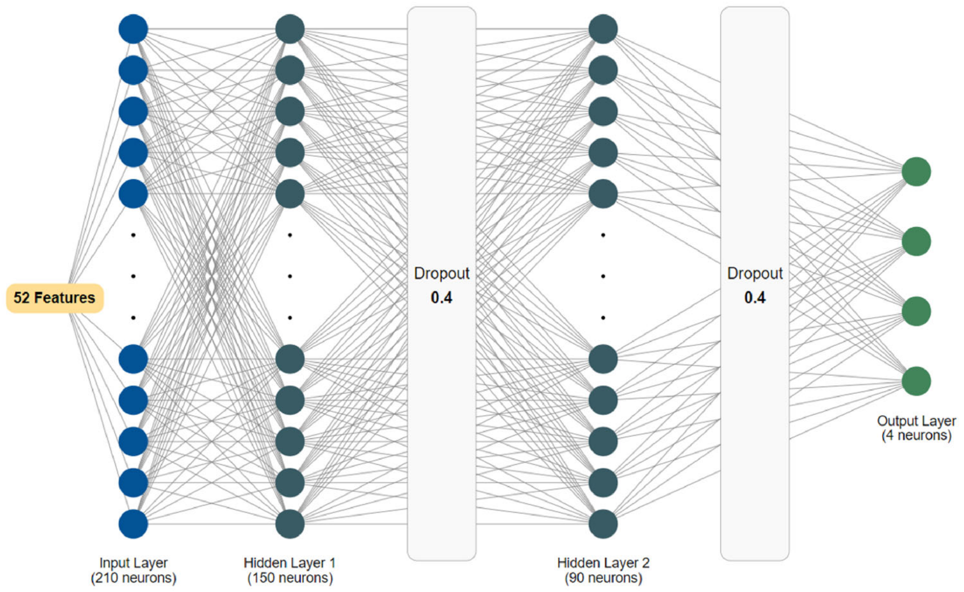
52 Features
Input Layer (210 neurons)
Hidden Layer 1 (150 neurons)
Dropout 0.4
Hidden Layer 2 (90 neurons)
Dropout 0.4
Output Layer (4 neurons)

Fig. 4 Layers and Neurons of the FNN-MC Architecture

fication prediction can be made. Sigmoid reduces the effect of extreme values and makes the models more stable. Since sigmoid is nonlinear, it allows a neural network to learn in more detail [101].

4.2.2 Feedforward Neural Network for Multiclass Classification (FNN-MC)

It is a feedforward neural network architecture used to detect malware types with multiclass classification. It has one input layer, two hidden layers and one output layer. As in FNN-BC, there are dropout layers after the hidden layers. The percentage of dropout layers in this architecture is 0.4. The first layer of the architecture takes 52 input features and produces 210 output neurons. These neurons are used as input to the first hidden layer. The first hidden layer has 150 neurons and the second hidden layer has 90 neurons. The Tanh activation function is used in the input layer and the two hidden layers. Tanh is used to calculate the output of each neuron in its layer. The last layer in this architecture takes the 90 neurons from the last hidden layer as input and finally produces four output neurons. Figure 4 shows the FNN-MC architecture.

Since FNN-BC will be used for multiclass classification, there are four neurons in the output layer. Softmax function is used as the activation function in the output layer. It acts as a mechanism to normalize the output vectors in a given layer of a neural network into a probability distribution. This function is particularly useful in multiclass classification problems. Because probability values are obtained for each class and the sum of these values is equal to 1 [89].

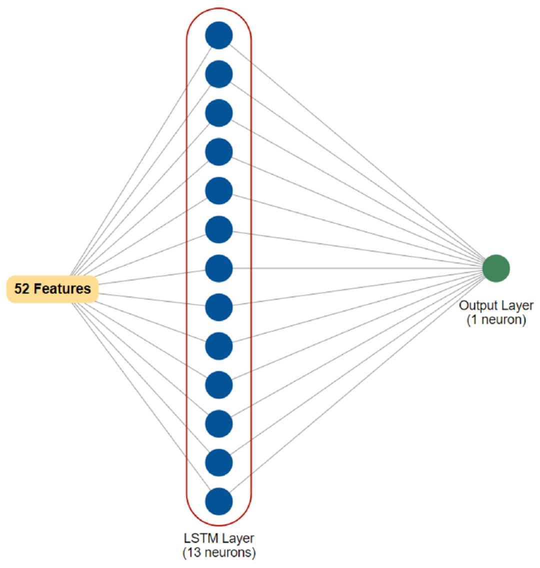

Fig. 5 Layers and neurons of the LSTM-BC architecture

4.2.3 Long Short-Term Memory for Binary Classification (LSTM-BC)

It is an LSTM architecture used as an alternative to the FNN-BC architecture for binary classification. This architecture basically consists of two layers. The first layer is the LSTM layer and the other layer is the output layer. The LSTM layer is a recursive layer used for the model to learn the sequential data structure. This layer takes 52 input features and contains a total of 13 neurons. So the output of this layer is a 13-dimensional vector. This vector is presented as input to the output layer. ReLU is used as the activation function in the LSTM layer. The output layer is a fully connected (Dense) layer and produces the final prediction of the model. Since this architecture is used for binary classification, there is only 1 neuron in the output layer. Sigmoid function is used as activation function in this layer.

LSTM-BC is a simple LSTM architecture. Therefore, the dropout layer, is not needed in this architecture. LSTM layers have a different structure than fully connected layers. In the figure showing the LSTM-BC and LSTM-MC architectures, a red line is drawn around it to emphasize this difference. Figure 5 shows all neurons and layers of the LSTM-BC architecture.

4.2.4 Long Short-Term Memory for Multiclass Classification (LSTM-MC)

The LSTM architecture was created to implement multiclass classification and to be an alternative to the FNN-MC architecture. LSTM-MC is a more complex architecture than the previously described LSTM architecture. The first layer of this architecture is an LSTM layer consisting of 200 neurons. This layer receives 52 input features. Tanh is used as the activation function for each neuron in this layer. Tanh is a standard activation function used in LSTM structures. After the first layer, a dropout layer with a value of 0.5 is used to prevent overfitting. With this layer, 50 of the neurons are randomly turned off. This layer is followed by another LSTM layer. In this layer, which we can call the second LSTM layer, there are 150 neurons. Following this layer, there is a dropout layer with a value of 0.5.

Finally, the LSTM-MC architecture is completed with the output layer with 4 neurons. Since this architecture will be used to solve a multiclass classification problem, as in FNN-MC, the output layer contains as many neurons as the number of classes to be predicted in the data set. Softmax activation function is used for the neurons in the last layer. Figure 6 shows the neurons and layers of the LSTM-MC architecture.

4.3 Proposed Deep Learning Models

In this study, two types of models based on FNN and LSTM are proposed. FNN- and LSTM-based models are divided into binary (FNN-BC and LSTM-BC) and multiclass (FNN-MC and LSTM-MC). Binary and multiclass models are also divided into federated and non-fed types.

In non-fed models with a classical machine learning approach, the number of users is assumed to be 1 since the entire dataset is hosted and processed on a central server. In federated models, 8, 16, 32, 64 and 128 users are proposed. For models with multiclass and federated learning approaches, a model with 128 users is not proposed because the training time of the simulation is too time consuming. The information about the 22 basic models and other variant models proposed in addition to these models and the evaluations about these models are discussed in detail in the Results and Discussion section.

4.4 Simulation Environment

In this study, it is aimed to create a system which is called FEDetect and to enable local users in the system to perform malware detection and malware type detection on their own data, while maintaining data confidentiality. In this context, a simulation environment was created on a basic device to represent all local users in the FL system and the FL server. This simulation first identifies each local device and generates local models for each of them, and then combines these models on behalf of the FL server. The local user devices and the FL server are treated as virtual, and there is no direct communication between them. The whole process is carried out representatively on a mainframe computer.

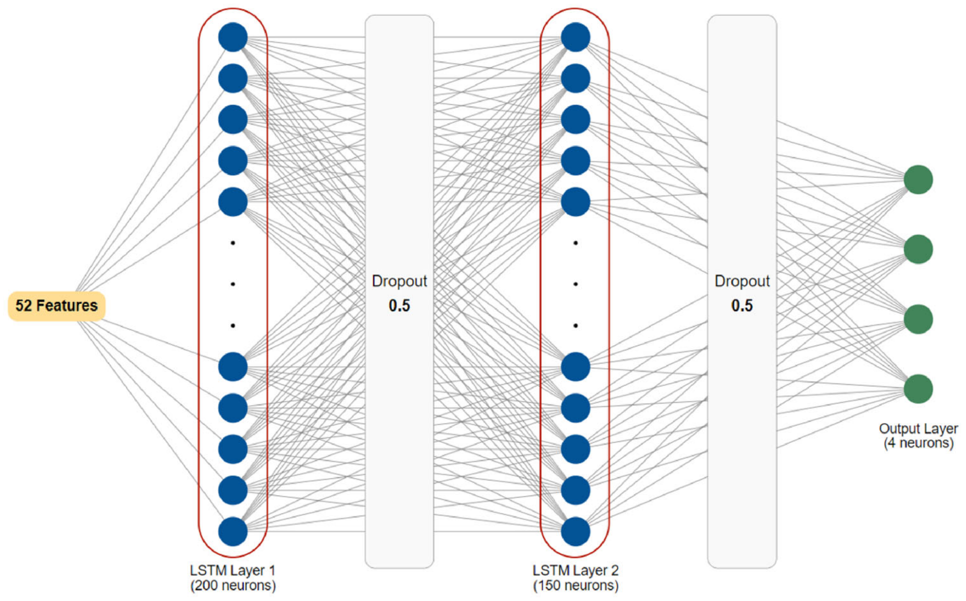

Fig. 6 Layers and neurons of the LSTM-MC architecture

4.4.1 Executing Federated Learning Simulation (FEDetect)

FL is a machine learning method that protects data privacy by keeping users' raw data on local devices. The FL simulation used in this study sends model parameters (i.e., weight coefficients) trained independently on each local device to a central FL server, while ensuring that user data remain on the devices themselves. This process significantly reduces the attackable surface area by eliminating the need for centralized data storage. The FL server only combines model updates from local devices to create a global model. In this way, information sharing between devices occurs only at the parameter level, and access to any individual user's private data is not possible.

The FEDetect basically consists of five parts. The system is executed by re-running some parts. In the first part, an initial model is created in FL server. This model is then referred to as the Global Model. In the second part, the global model is sent to the local devices one by one. In the third part, local devices train the global model with their own data. In the fourth part, these models updated on the local devices are sent back to the FL server one by one. Meanwhile, the global models updated on the local devices are then called local model. In the fifth part, the FL server merges the local models sent from the local devices using one of the model merging algorithms. Thus, a Round is completed. If the targeted accuracy value is not achieved, the execution of the system continues with the operations performed in the second part. In other words, the final global model is sent back to the local devices and the operations in the other sections are performed repeatedly until a target accuracy value is reached. It is important to determine in advance when and under which condition the FEDetect will terminate. In this study, we first generated non-fed models that can be used as benchmarks for Federated models in an environment where all data are available together. Then, it is tested whether the federated models can reach the accuracy values achieved by the non-fed models. The details of the simulation execution are shown in Fig. 7.

FedAvg (federated averaging) was used to combine the local models. FedAvg is a popular algorithm for distributed federated learning applications. It is used in a scenario where data are distributed across different devices, with each local device training with its own data. The goal of FedAvg is to merge the local models created on the local devices into a single global model. FedAvg averages the model weights from each local device and produces the weights of the global model. This process balances the contribution of all devices in the training process and enables distributed machine learning without a central server or coordination. This allows issues such as privacy and security to be better managed [60, 68].

Many studies using FedAvg use the SGD (stochastic gradient descent) [80] optimizer. However, recent studies in the literature have shown that using the Adam optimizer [47] for FedAvg gives better results [65, 98, 107]. Therefore, in this study, the Adam optimizer is used together with FedAvg (FedAvg-Adam).

4.4.2 Systems, Tools and Libraries Used in Simulation

The basic device used for model training is a business computer with Windows 10 operating system, Intel Core i7-12650 H CPU 2.30 GHz processor, 16GB RAM and 512GB SSD capacity, NVIDIA GeForce RTX 3060 GPU graphics card. Jupyter Notebook v6.5.2 was used as the development

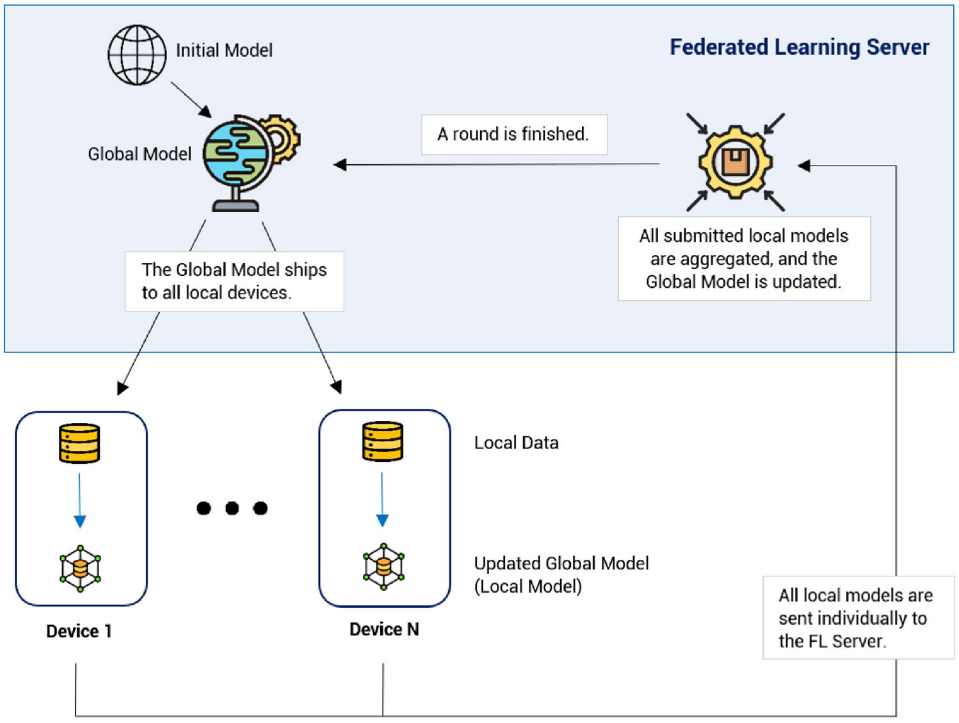
Initial Model
Federated Learning Server
Global Model
A round is finished.
All submitted local models are aggregated, and the Global Model is updated.
The Global Model ships to all local devices.
Local Data
Updated Global Model (Local Model)
Device 1
Device N
All local models are sent individually to the FL Server.

Fig. 7 Executing Federated Learning Simulation (FEDetect)

environment and Python v3.11.3 was chosen as the programming language. Tensorflow [1], Keras [28] Numpy [38], Pandas [59] and Scikit-learn [70] libraries that support the use of Python language were utilized to build deep learning models.

4.5 Proposed Federated Learning Method for the Experiment

The main recommended federated learning method is the function called *make_federate*. All operations regarding federated models are carried out by triggering this function. This function has 10 parameters. These parameters are: X, which is the matrix of independent variables; y, which is the target variable vector; arc, where the model architecture is defined; *model_num*, where the model number is specified; users, where the number of users is a federated model; acc, where the target accuracy value is given; local_epochs, where the number of epochs of training in local models should be specified; optimizer, where the optimizer is specified. The ts parameter is where the percentage of the data will be used as test data, and the bs parameter is where the batch size value is specified. All these parameters are prepared before this function is triggered. For each federated model, the parameters of this function were re-adjusted and executed.

Adam was used for the optimizer parameter. The parameters of Adam optimizer used in the study are: learning_rate, beta_1, beta_2 and epsilon are the parameters. The same optimizer parameter values were used in all federated models. learning_rate is a parameter related to how fast the optimization algorithm should learn. The default value for Adam is 0.001. Another parameter beta_1, determines the decay rate of the weighted average for the first moment, that is, how much of the past gradients will be remembered. beta_2 is a parameter that determines how much of the square of past gradients will be remembered. beta_1 and beta_2 values were used as 0.9 and 0.999. epsilon is a parameter used to prevent division by zero error and is taken as 1e-08 [23]. Table 4 shows the lists all the hyper parameters used across the models.

In this study, "binary cross-entropy" was used for binary classification and "sparse categorical cross-entropy" was used for multiclass classification with Adam optimizer. Binary cross-entropy is one of the most commonly used loss functions for measuring the accuracy of predictions between two classes and allows for establishing a balanced relationship between classes. It is particularly a suitable choice for optimizing probability predictions in binary classification problems. Sparse categorical cross-entropy, designed for use in multiclass classification tasks, is effective when target variables are expressed with indices rather than a dense vector. The effectiveness of these loss functions has been extensively documented in the literature. Goodfellow and colleagues [34] have discussed these loss functions' applications across different problems extensively. Additionally, the Adam optimizer, proposed by Kingma and Ba, is preferred due to its good optimization performance in nonlinear and high-dimensional problems, enhancing convergence speed

Table 4 Hyperparameters for all models

Learning rate	Test size	Optimizer	Beta_1	Beta_2	Epsilon	Binary loss	Multiclass loss
0.001	0.2	Adam	0.9	0.999	$1e-08$	Binary cross-entropy	Sparse categorical cross-entropy

and accuracy [48]. The combination of this optimization method and loss functions has played a significant role in enhancing the performance of our study.

In the proposed method, the parameter of the loss function is also adjusted for each architecture. For the models in FNN-BC and LSTM-BC architectures, binary_crossentropy loss function is used [40]. This loss function is used in binary classification problems. Here, the output of the model is expected to be between 0 and 1 and express the probability of belonging to a particular class. Since the FNN-BC and LSTM-BC architectures are used in binary classification models, the target variable in the data set was rearranged in the data preprocessing phase to be compatible with the binary_crossentropy loss function (i.e., to include 0 and 1 values). For the models in FNN-MC and LSTM-MC architectures, the sparse_categorical_crossentropy loss function was used [18]. This loss function is frequently used in multiclass classification problems. Each value of the target variable represents the index of a particular class. Since there are 4 different malware types in the dataset in this study, a target variable column consisting of 0, 1, 2 and 3 values representing each class was created in the data preprocessing stage to be suitable for the use of the sparse_categorical_crossentropy loss function.

The ts parameter in the proposed method is used to divide the data set into two as testing and training, and to specify the proportions in which this separation will be made. This parameter was set as 0.2 in all federated models. That is, 20% of the data set was used as test data. In the simulation environment using test data, the global model after each round; accuracy, precision, recall and F1 Score metrics were calculated. These calculated metrics were added to the history file created for each federated model, one line for each round. Later, these history files became available for graphic drawing and analysis of the models. Additionally, in the proposed method, the test data for LSTM-based federated models are rearranged. The test data were rendered three-dimensional in the form of (rows_num, time_step, features_num) in accordance with the LSTM algorithm. The time_step parameter in three-dimensional data is taken as 1.

The batch size parameter bs is used to divide the data into small pieces at a certain rate. In this way, data training is carried out faster and more effectively. With this method, called mini-batch, training is performed not on the entire data, but on a smaller portion that is specific and capable of representing the entire data. The mini-batch method is especially useful in terms of reducing training time. In addition, it is possible to train large models easily with less memory usage, a more stable gradient and better generalization of the relevant model [51]. The pseudo code of the make_federate function, whose various features have been explained so far, is given in Algorithm 1.

Algorithm 1 Pseudo code for `make_federate` function

```
1: FUNCTION make_federate(X, y, arc, model_num, users, acc,
   local_epochs, optimizer, ts, bs)
2: BEGIN FUNCTION
3: SET loss TO None
4: SET global_model TO None
5: SET metrics TO ['accuracy']
6: SET verbose TO 0
7: if arc EQUALS "FNN_BC" then
8:     SET loss TO "binary_crossentropy"
9:     SET global_model TO FNN_BC.build()
10: else if arc EQUALS "FNN_MC" then
11:     SET loss TO "sparse_categorical_crossentropy"
12:     SET global_model TO FNN_MC.build()
13: else if arc EQUALS "LSTM_BC" then
14:     SET loss TO "binary_crossentropy"
15:     SET global_model TO LSTM_BC.build()
16: else
17:     SET loss TO "sparse_categorical_crossentropy"
18:     SET global_model TO LSTM_MC.build()
19: end if
20: CALL global_model.compile(loss, optimizer, metrics)
21: SET history_file TO
22:      f"../results/model_(model_num).history"
23: with open(history_file, "w", encoding="UTF-8")
   as file: pass
24: SET round_count TO 0
25: while True do
26:     SET round_count TO (round_count + 1)
27:     SET global_weights TO global_model.get_weights()
28:     SET (X_train, X_test, y_train, y_test) TO
29:      train_test_split(X, y,
30: test_size=ts, stratify=y)
31:     if arc EQUALS "LSTM_BC" OR arc EQUALS "LSTM_MC"
   then
32:         SET X_test TO np.reshape(X_test,
33: (X_test.shape[0],1, X_test.shape[1]))
34:     end if
35:     SET scaled_local_weight_list TO []
36:     SET devices_data TO data_shuffle(X_train,
37: y_train, arc, users, bs)
38:     SET devices_names TO list(devices_data.keys())
39:     SET rtime TO 0
40:     for device IN devices_names do
41:         SET local_model TO None
42:         SET start_time TO time.time()
43:         if arc EQUALS "FNN_BC" then
44:             SET local_model TO FNN_BC.build()
45:         else if arc EQUALS "FNN_MC" then
46:             SET local_model TO FNN_MC.build()
```

```
        else if arc EQUALS "LSTM_BC" then
            SET local_model TO LSTM_BC.build()
        else
            SET local_model TO LSTM_MC.build()
        end if
        CALL local_model.compile(loss, optimizer, metrics)
        CALL local_model.set_weights(global_weights)
        CALL local_model.fit(devices_data[device],
      epochs=local_epochs, verbose=verbose)
        SET end_time TO time.time()
        SET rtime TO (rtime
+ (end_time - start_time))
        SET scaled_weights TO scale_calculate
(local_model.get_weights(),
 devices_data, device)
        CALL scaled_local_weight_list.append
(scaled_weights)
        CALL tensorflow.keras.backend.
clear_session()
    end for
    SET round_time TO round((rtime / users), 4)
    PRINT(round_count, round_time)
    SET average_weights TO
scaled_total_weights(scaled_local_weight_list)
    CALL global_model.set_weights(average_weights)
    SET (_loss, _accuracy, _precision, _recall, _f1) TO
test_model(X_test, y_test, global_model, arc)
    with open(history_file, "a", encoding="UTF-8")
as file:
    file.write("(" + round_count + ")
(" + round_time + ")
(" + _loss + ") (" + _accuracy + ")
(" + precision + ") (" + recall + ")
(" + f1 + ") \n")
    if _accuracy EQUALS acc OR _accuracy GREATER THAN
acc then
        BREAK WHILE
    end if
end while
SET history TO history_load(history_file)
SET total_rounds TO len(history['round'])
SET total_time TO round(sum(history['time']), 2)
PRINT(total_rounds, total_time)
RETURN history
END FUNCTION
```

For federated models, make_federate, which is the function where every transaction takes place, also includes some other functions. One of them is the data_shuffle function. For a better understanding of federated model training, it would be appropriate to provide information about this function. It performs fragmentation and mixing of the X_train and y_train data allocated for training, according to the specified number of users (users) and batch size value (bs), in accordance with the architecture specified in the arc parameter. For each round, the data are shuffled as if each user produces new data, and the shuffled data are shared among the specified number of users using the IID (independent and identically distributed) method. According to this method, when the data set is divided into a certain number of parts, a sample reflecting the general distribution of the data set before the division is obtained in each part.

In general, this study proposes a federated learning system that works under optimum conditions. Therefore, it is aimed to provide one of these optimum conditions with the IID method. Non-IID method is also used in the federated learning study [110]. Non-IID is a method in which the distribution of the data before division is not preserved. In the process of dividing the data using this method, some differences occur in terms of distribution between the resulting parts. While some parts contain some classes more heavily, other parts may consist of data that does not contain some classes at all. This subject can be explained more clearly with a simple example. For example, let's say a data set consisting of 100 rows and two classes that can be expressed as 0 and 1 is used. Considering that in 100 rows of data, 30 data represent class 0 and 70 data represent class 1, it is understood that the ratio (i.e., distribution) of 0 classes to 1 classes in this data set is 3/7. No matter how many parts this data set is divided into, if the distribution between 0 and 1 classes is preserved in each part with a ratio of 3/7 as in the whole data, this data set is divided by the IID method. In all other cases, the data can be said to be split using the Non-IID method.

With the IID method, after the data are distributed by preserving the current distribution of the data set, the data for each user are predetermined. Thus, the stage of producing local models on local devices begins. It is possible to see the codes that perform this operation in the scope of the for loop in the make_federate function. Then, the model parameters of all local models are combined and the global model is updated. The global model is tested with reallocated test data, and the accuracy value of the model is calculated. If the accuracy value of the newly updated global model is equal to or higher than the targeted accuracy value, the training of the relevant federated model is completed.

5 Results and Discussion

In this study, 22 basic models configured in 4 different deep learning architectures, with and without a federated learning approach, and a total of 28 models were developed in the whole study when variant models were added for experimental purposes. In this section, we first analyze the performance of the base models in detail. Then, additional evaluations are made about the effects of the variant models on the study. This section analyzes the performance of the base models in detail. Table 5 shows the architecture of each base model, whether it has a federated learning approach, the number of users of the federated model and how many rounds it takes to train the federated model.

In addition, since the mini-batch method is used in model training, the batch size value for each model and the amount

Table 5 Properties, hyperparameters and accuracies of models

Model	Model architecture	Learning type	Users	Epochs	Local epochs	Batch size	Data amount	Train time (s)	Accuracy
1	FNN-BC	**Non-fed**	**1**	**5**	**NA**	**32**	**1472**	**6.559**	**0.999**
2		Federated	8	5	1	128	368	3.000	0.999
3			16	6	1	64	736	2.720	0.999
4			32	9	1	32	1472	3.960	0.999
5			64	15	1	16	2944	4.930	0.999
6			128	30	1	8	5888	10.320	0.999
7	LSTM-BC	**Non-fed**	**1**	**5**	**NA**	**32**	**1472**	**10.923**	**0.999**
8		Federated	8	4	1	128	368	4.740	0.999
9			16	4	1	64	736	4.760	0.999
10			32	5	1	32	1472	5.890	0.999
11			64	11	1	16	2944	13.470	0.999
12			128	24	1	8	5888	28.230	0.999
13	FNN-MC	**Non-fed**	**1**	**5**	**NA**	**32**	**1472**	**1345.313**	**0.837**
14		Federated	8	422	1	128	368	362.090	0.837
15			16	65	20	64	736	170.340	0.837
16			32	100	20	32	1472	285.890	0.837
17			64	280	20	16	2944	1726.830	0.837
18	LSTM-MC	**Non-fed**	**1**	**5**	**NA**	**32**	**1472**	**2429.625**	**0.845**
19		Federated	8	875	1	128	368	6267.960	0.845
20			16	152	10	64	736	10286.890	0.845
21			32	313	10	32	1472	18935.751	0.845
22			64	562	10	16	2944	31412.170	0.845

of data corresponding to this value are also included. Finally, the total number of seconds each model training takes for a user, considering all rounds, is also included.

When analyzed in terms of various parameters and metrics, Table 5 contains a large amount of important findings related to the study. The first and most striking observation is that all federated learning models in the same architecture achieve the same accuracy values as all non-fed models. This is an empirical result that proves the effectiveness and applicability of the federated learning paradigm. This result eliminates the need to collect all the data on a central server, which is often used in classical machine learning. This is an important advantage, especially for applications where data privacy is critical. In this context, the federated learning approach provides a precise solution that does not compromise model performance while ensuring data privacy.

The ability of federated learning to achieve similar accuracy rates as non-federated methods fundamentally relies on the effectiveness of the optimization processes of local models. In non-federated models, all data are collected on a central server, whereas in federated learning, users share only the model parameters trained with their own data. This approach, especially in cases of homogeneous data distribution (IID), enhances performance by integrating the rich information from the combination of local models into a global model. Additionally, the FedAvg algorithm used in federated learning successfully maintains model accuracy at a level close to central methods by evenly assessing contributions from different users. When all models with FNN-BC architecture are compared with each other, it is possible to observe that rounds and training time increase as the number of users increases. This is a natural result and is generally observed in all models with equal local epochs. However, this increase is not proportional. When comparing Models 5 and 6, there is a twofold increase in the number of rounds, while there is a more than twofold increase in the training time.

Comparing Models 2 and 3, despite the doubling of the number of users, the number of rounds increased by only 1 and the training time decreased instead of increasing, which reveals the complexity of the convergence dynamics of the model. This is an exceptional case. Many factors can be suggested as to why this is the case. One of them is the randomization of the initial weights and other random factors. A poorly tuned initial model, when user data are involved, can cause training to start from a very wrong point, causing the local model to converge more slowly and thus requiring more rounds. Whether the model converges to a local optimum or the speed of convergence is variable can also affect the number of rounds and hence the training time [5, 26].

Another interesting result about federated models in the FNN-BC architecture is the low training time. It was determined that the training times of all federated models except model 6 with 128 users in the same architecture were less than the training time of the non-federated model. Moreover, these federated models trained with more data and consumed less time. This finding shows that the federated learning models performing binary classification in this study demonstrate effective performance in terms of data distribution and scalability. When the findings obtained from models with LSTM-BC architecture are examined, it is seen that they are generally similar to the findings in the FNN-BC architecture. As the number of users increases, the number of rounds and training time increase are the first striking similarities. The fact that these increases are not proportional is also a situation encountered in federated models in the FNN-BC architecture.

When we compare all federated learning models that perform binary classification, it is possible to see that all of them have very small round values, not exceeding 30. The main reason for this situation is that the models have low complexity. Binary classification models with a small number of neurons and layers confirmed that the training processes were completed in a short time. When binary classification FNN and LSTM architectures are compared, it has been determined that LSTM-based models consume more time than FNN. The most well-known reason for this is that LSTM-based models have a more complex structure than FNN-based models. Therefore, due to this complexity, training LSTM models requires more time. On the other hand, LSTM contains more parameters than FNN and LSTM models keep the information of previous steps for each time step while working on the data, which are other factors that slow down the training process. There are many studies in the literature that specifically discuss this issue and include the comparison of LSTM and FNN models [14, 54, 72, 85].

Table 5 contains a lot of information about federated and non-fed models that perform multiclass classification as well as binary classification. The training of multiclass non-fed models was more challenging (let’s use the word challenge) than that of binary non-fed models. In this respect, it would be appropriate for a better understanding of the subject to give a little more detail about the procedures performed during the training process of multiclass non-fed models.

The fact that multiclass models require more neurons and more layers in order to achieve a reasonable accuracy, necessitated the search for an appropriate number of neurons and layers. In this respect, after a series of tests, the architecture with the minimum possible number of neurons and layers was created and multiclass non-fed models were built. First, FNN-MC and LSTM-MC-based non-fed models were constructed and then federated models using these architectures were trained.

The goal of the FNN-MC-based non-fed model 13 is to have as high accuracy as possible while avoiding overfitting. In accordance with this aim, the validation dataset and early stopping method were used in the training process. Today, there are many different techniques of early stopping. In this study, the most widely known technique of early stopping was used. The training was terminated at 520 epochs with early stopping at the optimal moment of the training process, i.e., at the first point where the validation loss starts to exceed the training loss [66, 73]. The loss–epochs plot for Model 13 is shown in Fig. 8.

When the federated models in the FNN-MC architecture are analyzed, a general increase in round and training time is observed in parallel with the increase in the number of users, just as in the architectures previously commented. However, in the federated models in this architecture, a technique that is not needed in binary classification models has been tried. This technique is to increase the number of epochs in the local models.

For example, the fact that the 8-user model 14 takes 422 rounds of training time compared to its equivalents in FNN-BC and LSTM-BC (other 8-user federated models) suggests that the number of local epochs could be increased. For this purpose, three new models, called variant models, were constructed as derivatives of model 14. The values obtained as a result of the tests with these three variant models are shown in Table 6.

When Table 6 is examined, it has been determined that there are decreases in round and training time with local epoch values increased by variant models. The number of rounds, which was 422 in Model 14, dropped to 43 in Model 14A, which is a variant model. When the training times of the same models were examined, it was observed that there was a decrease from 362.09 to 225.72. This shows that increasing the number of local epochs can accelerate the training of the federated model. However, in real-world applications of federated learning, the processor power, disk capacity and similar device-related features of the devices in the federated system may be limited. For example, a device with insufficient processor power may experience various contraction problems during the training phase of the local model if the local epoch value is selected too high and it may take a long time to develop the relevant model. In fact, it may not be able to fully fulfill its duty within the federated system. In this respect, it is important to choose the local epoch number by taking such factors into consideration. Since all operations in this study are carried out in a simulation environment installed on a single computer, increasing the number of local epochs prevents the above-mentioned problems from occurring. Since increasing the local epoch value to 20 in the variant models of Model 14 gave successful and efficient results, the local epoch value was used as 20 in all other federated models in FNN-MC. When the round values of models 15, 16 and 17 were examined, it was seen that all of them were lower than the round value of model 14.

However, the effects of federated learning on communication load and training duration should be discussed. For example, the increase in training time observed as the number of users increases can be reduced by techniques such as optimizing the number of local epochs. In this study, model variants were created to explore ways to achieve the target accuracy level with fewer communication rounds. For instance, when examining the FNN-MC-based Model 14 and its variants, it was observed that increasing the number of local epochs reduced the number of rounds, thereby optimizing training time (Table 4). The implementation of such techniques offers significant advantages for the real-world applications of federated learning. Before LSTM-MC-based federated models were trained, the non-federated model (model 18) in this architecture was configured, as in FNN-MC. Again, as in FNN-MC, the training of the non-fed model was stopped at the most appropriate moment with early stopping and the metric values were recorded. It has been observed through comparisons whether LSTM-MC-based federated models also meet these metrics. In this sense, the loss–epochs graph of model 18 is shown in Fig. 9.

When LSTM-MC- and FNN-MC-based federated models are compared, it is found that there is a large increase in round and training time for LSTM-MC-based models.

This result shows that LSTM models are not efficient, especially for multiclass classification based on round and training time. However, in terms of accuracy, FNN-MC models had an accuracy of 0.837, while LSTM-MC models were higher with 0.845. In a real-world application, this relationship between training time and accuracy should be evaluated. When the round value of LSTM-MC model 19 was high as in FNN-MC, variant models were created for model 19. Three variant models were trained by gradually increasing the local epoch values, and the results are shown in Table 7.

Table 7 shows that, just like the variants of Model 14, there is a general decrease in round and training time values with the increase in the local epoch value. Due to the complexity of the LSTM-MC architecture, the number of local epochs in Model 19 and the following LSTM-MC-based federated models is used as 10 instead of 20. The reason for this can be explained in more detail as follows. There is a difference of approximately 30 s between the training times of Models 14B and 14C given in Table 6. There is a difference of approximately 250 s between Model 19B and 19C. Since these training times are the sum of the time each user spends in a given model over all rounds, it is possible to see that the training time in the LSTM-MC models increases tremendously as the number of users increases. Therefore, for model 19 and subsequent federated models, it was deemed sufficient to take the local epoch value as 10 in order to save time in the study. Loss–rounds plots of model 14 for variant models are given in Fig. 10.

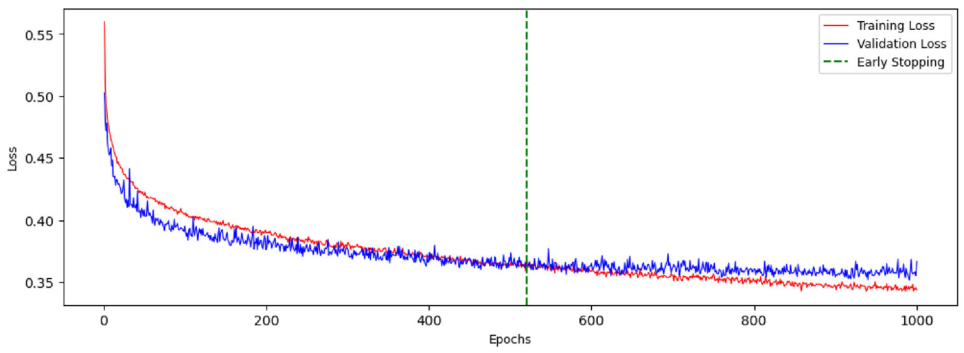

Fig. 8 Loss–epochs plot for Model 13 in the non-fed structure

Table 6 Variant federated models of the model 14

Model	Model architecture	Learning type	Users	Rounds	Local epochs	Train time (s)	Accuracy
14	FNN-MC	Federated	8	422	1	362.09	0.837
14A	FNN-MC	Federated	8	130	5	239.95	0.837
14B	FNN-MC	Federated	8	66	10	196.61	0.837
14C	FNN-MC	Federated	8	43	20	225.72	0.837

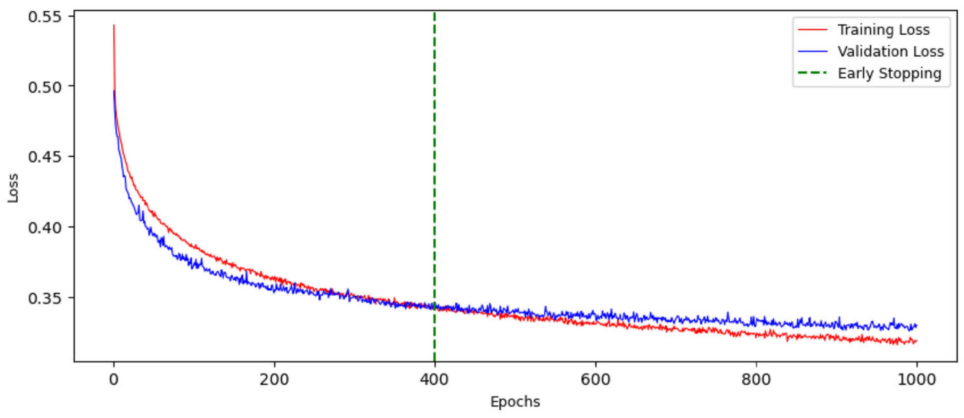

Fig. 9 Loss–epochs plot for Model 18 in the non-fed structure

Figure 10 shows that increasing the local epoch value makes the loss–rounds graph simpler. This is considered as an important result in terms of reducing the complexity of the model and the oscillations in the graph. Loss–rounds plots of the variant models of Model 19 are given in Fig. 11.

When we look at the graphs of the 19B model and the 19C model in Fig. 11, there is no difference as in the other variant models. In other words, the graphs of model 19B with a local epoch value of 10 and model 19C with a local epoch value of 20 are very similar. This shows that there is a limit to increasing the number of local epochs. Therefore, in order to finish the study faster, the local epoch value is taken as 10 in all LSTM-MC models after model 19.

A confusion matrix provides a more comprehensive analysis of classifier accuracy, not just through a general metric like accuracy, but also by showing errors such as false positives and false negatives. However, in our study, while comparing different models, our primary focus was on general perfor-

Table 7 Variant federated models of the model 19

Model	Model architecture	Learning type	Users	Rounds	Local epochs	Train time (s)	Accuracy
19	LSTM-MC	Federated	8	875	1	6267.96	0.845
19A	LSTM-MC	Federated	8	159	5	2827.83	0.845
19B	LSTM-MC	Federated	8	81	10	1858.88	0.845
19C	LSTM-MC	Federated	8	58	20	2120.10	0.845

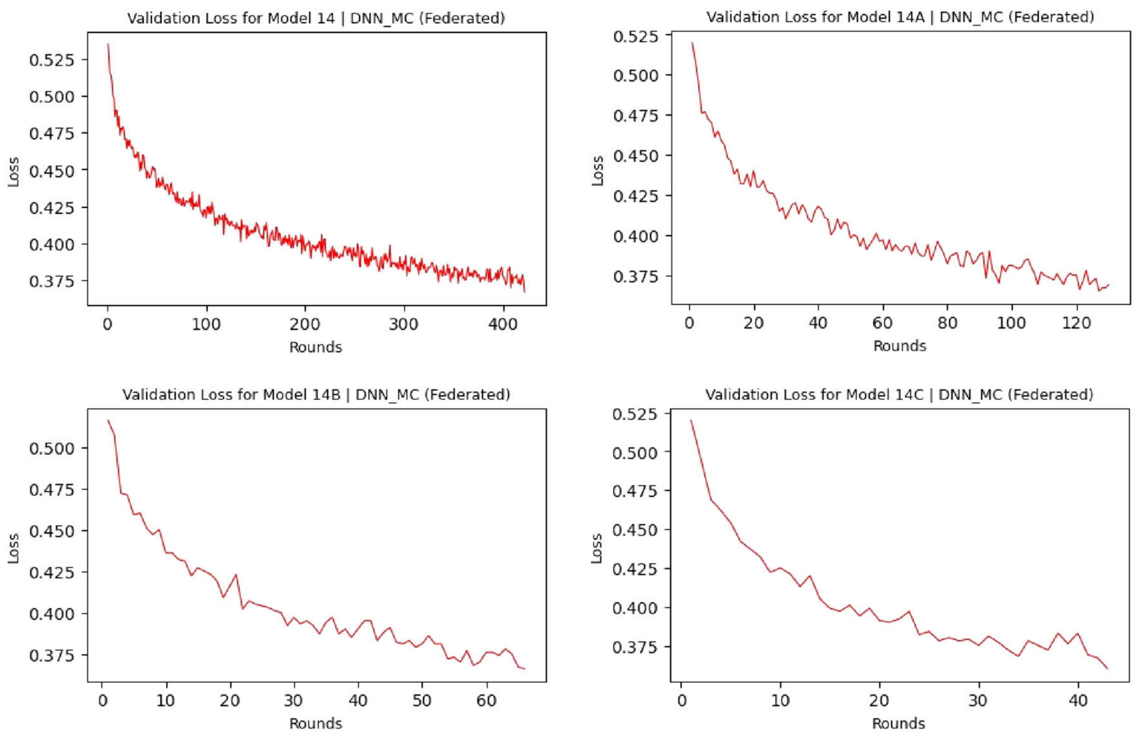

Fig. 10 Loss–rounds graphs of Model 14 and variant models

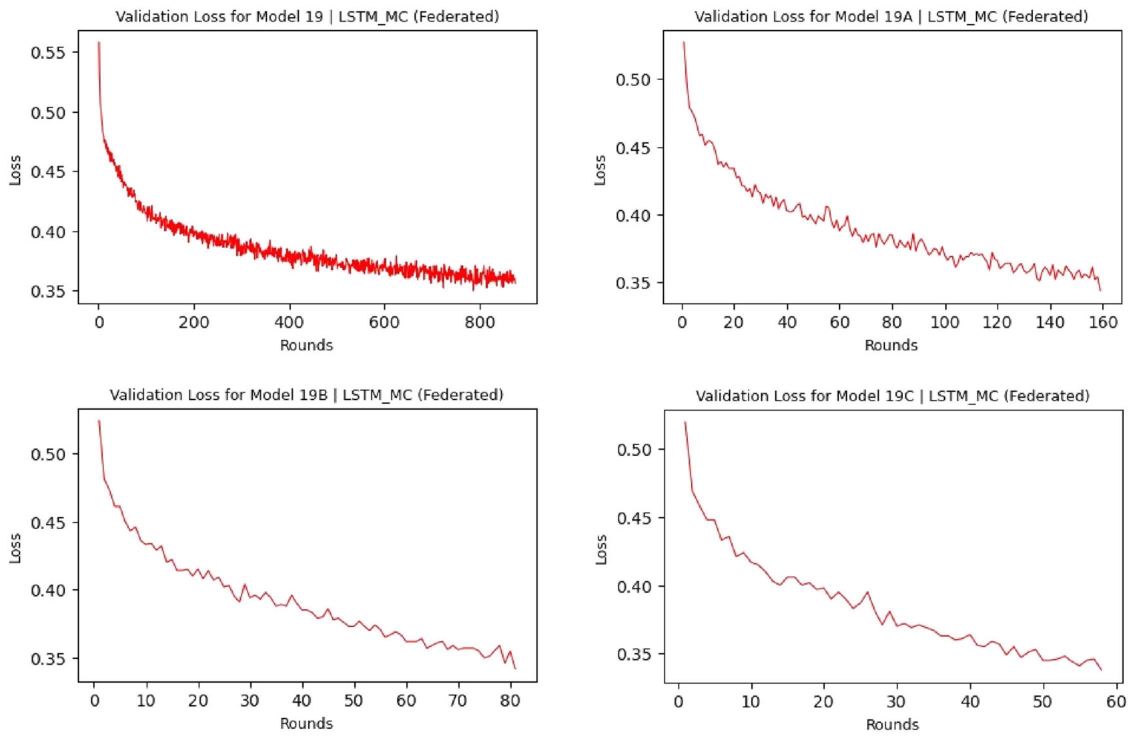

Fig. 11 Loss–rounds graphs of model 19 and variant models

mance measurements (such as accuracy, precision, recall, and F1-score) and evaluating the effects of federated learning. Therefore, detailed analyses like confusion matrices have been included only in limited cases. For this purpose, evaluations based on four different criteria were conducted, resulting in a total of 6 confusion matrix graphs being drawn.

I. Confusion matrix based on highest accuracy A confusion matrix can be drawn for models with the highest accuracy. These models are important as they best represent the performance of the classifiers. In this sense, Model 1 and Model 7 (binary classification) have been considered the most successful models in two different architectures. Figure 12 shows the Model 1 and Model 7 confusion matrix separately.

When examining Fig. 12, it is possible to say that both models demonstrated almost perfect performance. The presence of only 28 false positives and 28 false negatives between the benign and Malware classes indicates that the models achieved high accuracy in a balanced manner for both classes. This situation particularly highlights the effectiveness of the optimization algorithms (Adam) and the loss function (binary cross-entropy) used in binary classification problems. The high rate of correct predictions and the low rate of incorrect predictions for both models demonstrate their usability in practical applications.

II. Different learning types (non-federated vs. federated comparison) The comparison of non-federated Model 1 and Federated Model 5 shows that federated learning exhibits similar performance to non-federated methods. Both models have the same accuracy rate (Accuracy: 0.999) and similar numbers of incorrect predictions in their confusion matrices (28 false positives and 28 false negatives). However, the confusion matrix for Federated Model 12 (LSTM-BC, Federated) shows slightly lower performance compared to Federated Model 5, with 38 false positives in the benign class and 58 false negatives in the Malware class. This situation indicates that an increase in the number of users and communication delays can negatively affect federated learning processes, especially in more complex networks like LSTM. Confusion matrix for Models 5 and 12 can be seen in Fig. 13.

III. Multiclass vs. binary classification comparison Multiclass classification models (Model 13 and Model 18) have lower accuracy rates compared to binary classification models (Model 1 and Model 7). For instance, the confusion matrix for Model 13 (DNN-MC, non-fed) shows a higher number of false positives (3000) and false negatives (2900) particularly in the benign class. However, a more balanced performance is observed for Model 18 (LSTM-MC, non-fed); the numbers of false positives (2990) and false negatives (3000) in the benign class are slightly lower.

This situation indicates that multiclass classification problems are more complex and that models make more errors in these classes. Additionally, it may present an opportunity to use more advanced techniques to improve the performance of the "sparse categorical cross-entropy" loss function used in multiclass classification. For Model 13 and 18, confusion matrix can be seen in Fig. 14.

Table 8 Results for all basic models

Model	Accuracy	Precision	Recall	F1 Score
1	0.999	0.999	0.999	0.999
2	0.999	0.998	0.999	0.999
3	0.999	0.999	0.999	0.999
4	0.999	0.999	0.999	0.999
5	0.999	0.999	0.999	0.999
6	0.999	0.998	0.999	0.999
7	0.999	0.999	0.999	0.999
8	0.999	0.999	0.998	0.999
9	0.999	0.999	0.999	0.999
10	0.999	0.998	0.998	0.999
11	0.999	0.999	0.998	0.999
12	0.999	0.999	0.998	0.999
13	0.837	0.837	0.837	0.837
14	0.837	0.837	0.837	0.837
15	0.837	0.836	0.837	0.836
16	0.837	0.837	0.837	0.837
17	0.837	0.839	0.838	0.837
18	0.845	0.845	0.845	0.845
19	0.845	0.844	0.845	0.844
20	0.845	0.845	0.845	0.845
21	0.845	0.845	0.845	0.844
22	0.845	0.844	0.845	0.845

IV. The impact of different user numbers Comparing Federated Model 5 (32 users) with Model 12 (128 users), a slight decrease in performance is observed as the number of users increases. Model 5, in a scenario with fewer users, demonstrates very high accuracy in its confusion matrix with only 28 false positives and 28 false negatives. In contrast, Model 12 shows higher numbers of false positives (38) and false negatives (58). This situation suggests that the convergence speed of the network during federated learning may slow down as the number of users increases, or inconsistencies arising from communication costs could affect model performance. The impact of increased user numbers may become more pronounced, especially in more complex networks and multiclass classification scenarios.

In this study, in addition to the accuracy metric, precision, recall and f1 score metrics for the 22 base models were also calculated and saved in the history files of the models. The metrics for all base models are shown in Table 8.

when Table 8 is examined, it is seen that the metrics except accuracy are almost the same as the accuracy value. It was determined that there was only a change in a few models and

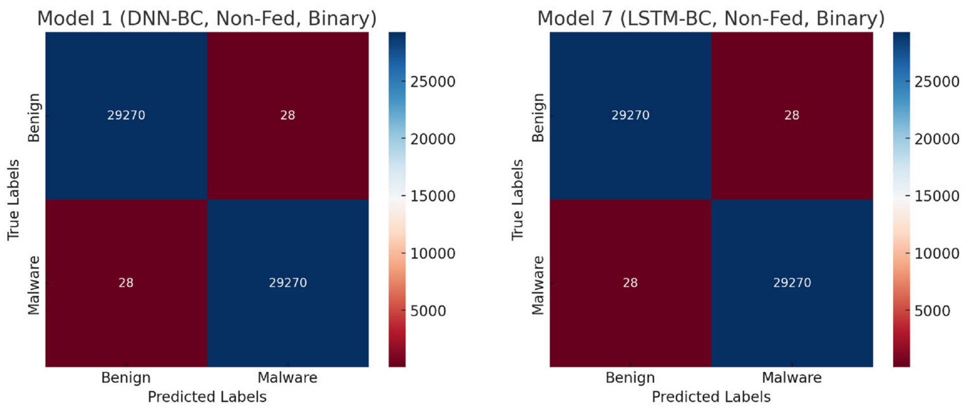

Fig. 12 Confusion matrices for Model 1 and Model 7

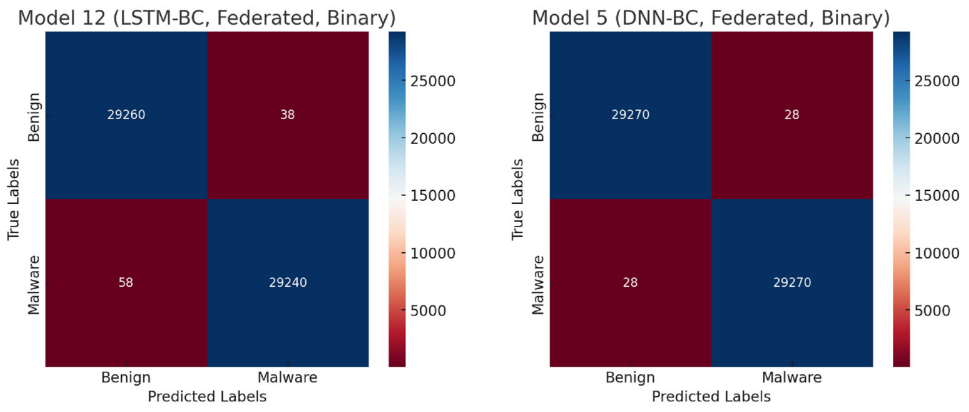

Fig. 13 Confusion matrices for Model 12 and Model 5

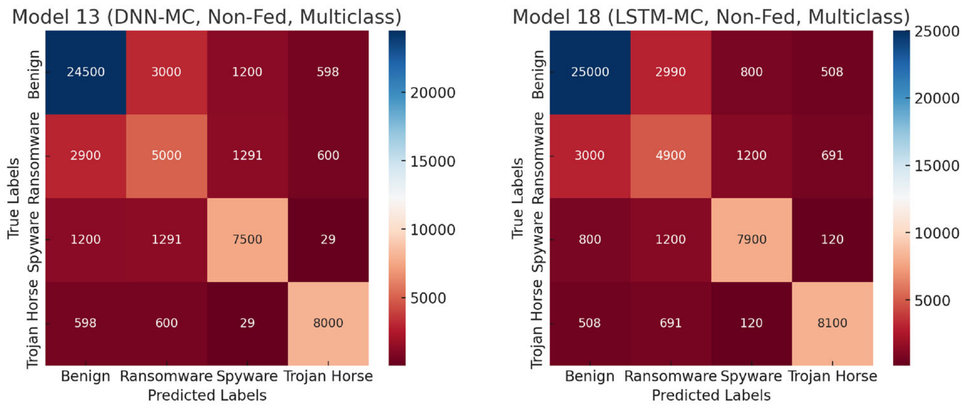

Fig. 14 Confusion matrices for Model 13 and Model 18

a maximum change of around 0.001. The fact that the metric values are so close to each other shows the robustness of all 22 models.

When the experimental findings in the history files of the models were examined, it was observed that the training of many models (especially multiclass models) could be completed more quickly if the accuracy value was sacrificed. For example, model 14, which had an accuracy value of 0.837, remained in training for 422 rounds to achieve this accuracy value. But if the accuracy value is sacrificed by 0.001, 408 rounds, it has been determined that if 0.007 is waived, the training of the relevant federated model may be completed within 325 rounds. This emerges as an important finding in terms of reducing the number of communications with the central server, especially in real-world applications of federated learning. Because each round corresponds to a connection to the central server.

The technique, which can be explained as early termination of federated model training, is similar to the early stopping technique used when training classical deep learning models. If training a federated model faster is more important than deviating from the accuracy value by around 0.001, it is anticipated that using this technique in such a study may be beneficial. Table 9, where the early stop round values and early stop accuracy values of multiclass federated

Table 9 Round and accuracy values of some multiclass federated models that were stopped early

Federated MC models	Last round	Last accuracy	Early stop round	Early stop accuracy
14	422	0.837	408	0.836
			325	0.830
14A	130	0.837	127	0.835
			100	0.830
14B	66	0.837	65	0.836
			52	0.830
14C	43	0.837	41	0.835
			39	0.830
15	65	0.837	60	0.836
			52	0.830
16	100	0.837	96	0.832
			93	0.830
17	280	0.837	250	0.836
			192	0.830
19	875	0.845	804	0.844
			670	0.840
19A	159	0.845	136	0.843
			135	0.840
19B	81	0.845	75	0.841
			56	0.839
19C	58	0.845	54	0.844
			45	0.839
20	152	0.845	125	0.844
			104	0.839
21	313	0.845	280	0.844
			223	0.839
22	562	0.845	496	0.844
			401	0.839

models are given, obtained from the findings in the history files containing the experimental results, is as follows.

Binary classification models are not included in Table 9. This is because the binary classification models in this study are models that can be trained in a short time and have a maximum of 30 rounds. It was thought that applying the early stopping technique in federated models with binary classification would not have much effect. Also, according to Table 9, it is seen that generally, sacrificing the accuracy value in variant models is not very effective. For example, in model 14B, when the accuracy value was sacrificed around 0.001, only 1 round reduction was realized. When model 19C is analyzed, there was a decrease from 58 rounds to 54. The reason why the early stopping technique is not very effective in variant models is that there is already enough round reduction with the increase of the local epoch value. However, in cases where it is desirable to train federated models on local devices faster, since the local epoch value will be taken as 1, in such federated systems, the accuracy value can be sacrificed to some extent in order to reduce the round values of the models.

After giving the details of this study, it can be compared with other studies that have been done in the past. Table 10 lists the studies that used the CIC-MalMem-2022 dataset for malware detection with binary and multiclass classification.

This study has a significant advantage among previous similar studies. When examined in terms of binary classification, a better performance was shown than all other studies, except for studies [55, 96]. These studies also reached a better accuracy value than this study, only around 0.0001.

Considering the other achievements of this study, such a small loss of accuracy is negligible. Because none of the studies in Table 10 used the federated learning approach. However, in this study, on the one hand, the classical machine learning approach was used, like other studies, and on the other hand, a broader perspective was offered to researchers with the federated learning approach. Federated models also reached all the accuracy values achieved by non-fed models.

Table 10 Comparison of studies using the CIC-MalMem-2022 dataset and non-fed approach studies

Study	Classification	Architecture	Accuracy
Our proposed approach	Binary	FNN-BC	**0.9999**
		LSTM-BC	**0.9999**
Louk and Tama [55]		RF	1.0
Talukder et al. [96]		XGBoost & RF	1.0
Smith et al. [91]		Ada Boost	0.9999
Mezina and Burget [64]		RF	0.99
Dener et al. [24]		LR	0.9997
		GBT	0.9994
		NB	0.9841
Ghazi and Raghava [32]		PSO & kNN	0.99
		PSO & SVM	0.99
Naeem et al. [67]		CNN & MLP	0.998
Our proposed approach	Multiclass	FNN-MC	**0.837**
		LSTM-MC	**0.845**
Mezina and Burget [64]		CNN	0.8353

While all other studies do not take data privacy into consideration, this study achieved better accuracy than other studies, and on the other hand, the protection of users' sensitive data was also ensured. This study was a study that tested more different models and contained more information in different areas compared to the related studies mentioned above. When evaluated in terms of multiclass classification, there are not many studies done before. Almost all malware detection studies carried out with the CIC-MalMem-2022 dataset include only binary classification. Only study [64] includes a multiclass classification example and has an accuracy value of 0.8353. In this study, this accuracy value was exceeded in both FNN- and LSTM-based models. Moreover, both non-federated and federated learning-based models showed better performance in multiclass classification, reaching accuracy values of 0.837 and 0.845, respectively.

Apart from the studies carried out with the CIC-MalMem-2022 data set, there are also malware detection studies carried out with different data sets. This study has significant advantages when compared to studies using different data sets. Table 11 shows studies carried out with different data sets other than CIC-MalMem-2022.

In some studies, the results were only given as accuracy or F1 score, so a comparison was made on these metrics. According to this comparison, it is seen that our study is more successful than all the studies given in Table 11 in both metrics. In addition, all of the other studies given in the table only performed binary classification. In this respect, it is possible to say that our study is more successful.

6 Conclusion

in this study, the classification results obtained by federated learning-based models and traditional (non-fed) models using the same deep learning architectures were compared from various perspectives. Using the CIC-MalMem-2022 dataset, both malware detection (binary classification) and malware type detection (multiclass classification) were performed. Throughout the study, it was demonstrated that the federated learning method could achieve similar accuracy rates as traditional centralized methods while preserving data privacy. The most successful non-fed model in binary classification achieved an accuracy rate of 0.999, while this value was 0.845 for multiclass classification. These values are shown as bold in Tables 10 and 11. Federated learning models achieved the same accuracy rates while offering the advantage of data privacy.

Traditional and federated learning methods are considered together using a recent dataset such as CIC-MalMem-2022. In addition, the study is carried out in a simulation environment. Simulation allowed issues such as performance and scalability to be analyzed in a secure and controllable environment before building a real system. This approach is a valuable preparation for understanding the real-world applicability of federated learning systems. Furthermore, analyzing the effect of different numbers of users and local epoch parameters provided clues to improve the scalability of the system.

A mechanism called early round stopping was developed to reduce the communication overhead and optimize the training time in the federated system and variant models were designed to analyze the effects of this mechanism. With this mechanism, the global model reaches the target accuracy

Table 11 Comparison of studies using federated learning approach in detecting malware

Study	Classification	Dataset	Architecture	Metric
Our proposed approach	Binary	CIC-MalMem-2022	FNN-BC	**Accuracy: 0.9999**
			LSTM-BC	**F1: 0.9999**
Lin and Huang [53]		VirusTotal	SVM	Accuracy: 0.9167
Rey et al. [78]		N-BaIoT	SVM	Accuracy > 0.97
Hsu et al. [42]		Opera Mobile Store	SVM	Accuracy: 0.9345
Galvez et al. [30]		MaMaDroid	LiM	F1: 0.95
Jiang et al. [44]		Androzoo	FedHGCDroid	Accuracy: 0.913
Taheri et al. [95]		Genome	FedGAN-BK	Accuracy: 0.9324
Pei et al. [71]		DREBIN	FedMalDE	F1: 0.9764

level with fewer rounds. This can be considered as an important step toward a faster and more efficient implementation of federated learning.

While emphasizing the strengths of FEDetect, the study also points to areas for future research. For example, a scenario where the data distribution is homogeneous (IID) was studied, but the effects of real-world data heterogeneity (non-IID) remain an important research topic for FL models. Furthermore, this study was conducted in a simulation environment and does not include the effects of real-time communication and system delays. Therefore, how federated learning will perform on more complex and heterogeneous data sources is an important research area for future work.

It was observed that the computer system used during the execution of the FEDetect simulation was very time consuming. Federated models were tested with up to 128 users. In future studies, federated learning models with more users can be built to take advantage of faster computer systems.

For future studies, it is also recommended to test federated learning methods in real-world applications. For example, the performance of an FL system with heterogeneous and real-time data sources such as mobile devices or IoT systems can be analyzed. Additionally, besides the FNN and LSTM models used in this study, newer deep learning architectures and optimization algorithms can be examined. Especially, Non-IID optimization techniques that address data heterogeneity could further enhance the practical use of federated learning.

In conclusion, this work provides a valuable framework for how federated learning can be applied while balancing data privacy and model performance. Rather than real-world applications, this simulation provides a solid foundation for understanding the potential challenges of FL systems and developing more effective systems.

Funding source Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK). This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data Availability The data used in this study are provided in the main text of the article.

Declarations

Conflict of interest The authors declare that there is no conflict of interest regarding the publication of this paper.

Ethical approval There is no any ethical conflict.

References

1. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.: TensorFlow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16) (2016)
2. Akhmetova, S.; Ibrayeva, A.: The European data protection model as a global privacy standard. Farabi J. Soc. Sci. **9**(1), 79–86 (2023)
3. Alahi, A.; Goel, K.; Ramanathan, V.; Robicquet, A.; Fei-Fei, L.; Savarese, S.: Social LSTM: Human trajectory prediction in crowded spaces. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2016)
4. Ali, P.J.M.; Faraj, R.H.; Koya, E.; Ali, P.J.M.; Faraj, R.H.: Data normalization and standardization: a technical report. Mach Learn Tech Rep **1**(1), 1–6 (2014)
5. Aljarah, I.; Faris, H.; Mirjalili, S.: Optimizing connection weights in neural networks using the whale optimization algorithm. Soft. Comput. **22**, 1–15 (2018)
6. Allix, K.; Bissyande, T.F.; Klein, J.; Le Traon, Y.: Androzoo: collecting millions of android apps for the research community. In: Proceedings of the 13th International Conference on Mining Software Repositories (2016)

7. Alvarez, J.M.; Salzmann, M.: Learning the number of neurons in deep networks. In: Advances in Neural Information Processing Systems, 29 (2016)
8. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C.: Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In: NDSS (2014)
9. Baldi, P.; Sadowski, P.J.: Understanding dropout. In: Advances in Neural Information Processing Systems, 26 (2013)
10. Bengio, Y.; Simard, P.; Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
11. Bettany, A.; Halsey, M.; Bettany, A.; Halsey, M.: What is Malware? In windows virus and malware troubleshooting, 1–8 (2017)
12. Borges, L.: Malware-Exploratory. https://www.kaggle.com/code/lucaslba/malwareexploratory/data (2022)
13. Bostock, M.; Ogievetsky, V.; Heer, J.: D^3 data-driven documents. IEEE Trans. Visual Comput. Graphics **17**(12), 2301–2309 (2011)
14. Bubashait, M.; Hewahi, N.: Urban sound classification using DNN, CNN & LSTM: a comparative approach. In: 2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT) (2021)
15. Butwall, M.: Data normalization and standardization: impacting classification model accuracy. Int. J. Comput. Appl. (2021)
16. Buyuktanir, B.; Yildiz, K.; Eyup, U.; Buyuktanir, T.: du-CBA: data-agnostic and incremental classification-based association rules extraction architecture. J. Fac. Eng. Archit. Gazi Univ. 38(3). (2023)
17. Carrier, T.; Victor, P.; Tekeoglu, A.; Lashkari, A.H.: 2022. Detecting Obfuscated Malware Using Memory Feature Engineering, In ICISSP (2022)
18. Chaithanya, B.; Swasthika Jain, T.; Usha Ruby, A.; Parveen, A.: An approach to categorize chest X-ray images using sparse categorical cross entropy. Indones. J. Electric. Eng. Comput. Sci. 1700–1710 (2021)
19. Chang, X.; Li, Z.; Zhang, Q.; Gao, Y.; Wang, Y.; Tian, Q.; Tian, F.; Xin, X.: Modulation format identification for coherent optical communication systems based on long short-term memory networks. In: 2022 20th International Conference on Optical Communications and Networks (ICOCN) (2022)
20. Cheelamanthula, K.: Implementation of long short term memory neural network model for electrical load. In: 2022 4th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA) (2022)
21. Chowdhury, A.; Kassem, H.; Padoy, N.; Umeton, R.; Karargyris, A.: A review of medical federated learning: applications in oncology and cancer research. In: Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 7th International Workshop, BrainLes 2021, Held in Conjunction with MICCAI 2021, Virtual Event, September 27, 2021, Revised Selected Papers, Part I (2022)
22. De Ryck, T.; Lanthaler, S.; Mishra, S.: On the approximation of functions by tanh neural networks. Neural Netw. **143**, 732–750 (2021)
23. Defossez, A.; Bottou, L.; Bach, F.; Usunier, N.: A simple convergence proof of adam and adagrad. arXiv preprint arXiv:2003.02395. (2020)
24. Dener, M.; Ok, G.; Orman, A.: Malware detection using memory analysis data in big data environment. Appl. Sci. **12**(17), 8604 (2022)
25. Dubey, S.R.; Singh, S.K.; Chaudhuri, B.B.: Activation functions in deep learning: a comprehensive survey and benchmark. Neurocomputing (2022)
26. Faris, H.; Aljarah, I.; Mirjalili, S.: Training Feedforward neural networks using multi-verse optimizer for binary classification problems. Appl. Intell. **45**, 322–332 (2016)
27. Fox, R.: Linux with Operating System Concepts. CRC Press, Boca Raton (2021)
28. Francois, C.: Keras. https://github.com/fchollet/keras (2015)
29. Fromholz, J.M.: The European union data privacy directive. Berk. Tech. LJ **15**, 461 (2000)
30. Galvez, R.; Moonsamy, V.; Diaz, C.: Less is More: A privacy-respecting Android malware classifier using federated learning. arXiv preprint arXiv:2007.08319. (2020)
31. Gers, F.A.; Schmidhuber, J.: Recurrent nets that time and count. In: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium (2000)
32. Ghazi, M.R.; Raghava, N.: Machine learning based obfuscated malware detection in the cloud environment with nature-inspired feature selection. In: 2022 5th International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT) (2022)
33. Goddard, M.: The EU general data protection regulation (GDPR): European regulation that has a global impact. Int. J. Mark. Res. **59**(6), 703–705 (2017)
34. Goodfellow, I.; Bengio, Y.; Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
35. Graves, A.; Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. **18**(5–6), 602–610 (2005)
36. Greff, K.; Srivastava, R.K.; Koutnik, J.; Steunebrink, B.R.; Schmidhuber, J.: LSTM: A search space odyssey. IEEE Trans. Neural Netw. Learn. Syst. **28**(10), 2222–2232 (2016)
37. Hara, K.; Saito, D.; Shouno, H.: Analysis of function of rectified linear unit used in deep learning. In: 2015 International Joint Conference on Neural Networks (IJCNN) (2015)
38. Harris, C.R.; Millman, K.J.; Van Der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020)
39. Hassoun, M.H.: Fundamentals of Artificial Neural Networks. MIT Press, Cambridge (1995)
40. Ho, Y.; Wookey, S.: The real-world-weight cross-entropy loss function: modeling the costs of mislabeling. IEEE Access **8**, 4806–4813 (2019)
41. Hochreiter, S.; Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
42. Hsu, R.H.; Wang, Y.C.; Fan, C.I.; Sun, B.; Ban, T.; Takahashi, T.; Wu, T.W.; Kao, S.W.: A privacy-preserving federated learning system for Android malware detection based on edge computing. In: 2020 15th Asia Joint Conference on Information Security (AsiaJCIS) (2020)
43. Jain, A.K.; Mao, J.; Mohiuddin, K.M.: Artificial neural networks: a tutorial. Computer **29**(3), 31–44 (1996)
44. Jiang, C.; Yin, K.; Xia, C.; Huang, W.: FedHGCDroid: an adaptive multi-dimensional federated learning for privacy-preserving android malware classification. Entropy **24**(7), 919 (2022)
45. Kasmaiee, S.; Kasmaiee, S.; Homayounpour, M.: Correcting spelling mistakes in Persian texts with rules and deep learning methods. Sci. Rep. **13**, 19945 (2023)
46. Kiltz, S.; Lang, A.; Dittmann, J.: Malware: specialized trojan horse. In Cyber Warfare and Cyber Terrorism (pp. 154–160). IGI Global (2007)
47. Kingma, D.P.; Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980. (2014)
48. Kingma, Diederik P.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).
49. LeCun, Y.; Bengio, Y.; Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
50. Li, J.: CS 332-006: principles of operating systems (2020)

51. Li, M.; Zhang, T.; Chen, Y.; Smola, A.J.: Efficient mini-batch training for stochastic optimization. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2014)
52. Li, Z.; Shao, J.; Mao, Y.; Wang, J.H.; Zhang, J.: Federated learning with GAN-based data synthesis for non-IID clients. Trustworthy Federated Learning: First International Workshop, FL 2022, Held in Conjunction with IJCAI 2022, Vienna, Austria, July 23, 2022, Revised Selected Papers (2023)
53. Lin, K.Y.; Huang, W.R.: Using federated learning on malware classification. In: 2020 22nd International Conference on Advanced Communication Technology (ICACT) (2020)
54. Liu, Y.; Wang, Y.; Yang, X.; Zhang, L.: Short-term travel time prediction by deep learning: a comparison of different LSTM-DNN models. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC) (2017)
55. Louk, M.H.L.; Tama, B.A.: Tree-based classifier ensembles for PE Malware analysis: a performance revisit. Algorithms **15**(9), 332 (2022)
56. Mahdavifar, S.; Kadir, A.F.A.; Fatemi, R.; Alhadidi, D.; Ghorbani, A.A.: Dynamic Android malware category classification using semi-supervised deep learning. In: 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). (2020)
57. Manaswi, N.K.; Manaswi, N.K.: RNN and LSTM, pp. 115–126. Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras, Deep Learning with Applications Using Python (2018)
58. Mariconti, E.; Onwuzurike, L.; Andriotis, P.; De Cristofaro, E.; Ross, G.; Stringhini, G.: Mamadroid: detecting android malware by building Markov chains of behavioral models. arXiv preprint arXiv:1612.04433. (2016)
59. McKinney, W.: Data structures for statistical computing in python. In: Proceedings of the 9th Python in Science Conference (2010)
60. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics (2017)
61. McMahan, B.R.; Daniel.: Federated learning: Collaborative machine learning without centralized training data. Retrieved from https://ai.googleblog.com/2017/04/federated-learning-collaborative.html (2017)
62. Meidan, Y.; Bohadana, M.; Mathov, Y.; Mirsky, Y.; Shabtai, A.; Breitenbacher, D.; Elovici, Y.: N-BaIoT–network-based detection of IoT botnet attacks using deep autoencoders. IEEE Pervasive Comput. **17**(3), 12–22 (2018)
63. Mercioni, M.A.; Holban, S.: A brief review of the most recent activation functions for neural networks. In: 2023 17th International Conference on Engineering of Modern Electric Systems (EMES) (2023)
64. Mezina, A.; Burget, R.: Obfuscated malware detection using dilated convolutional network. In: 2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT) (2022)
65. Mills, J.: Federated Machine Learning in Edge Computing. University of Exeter, Exeter (2022)
66. Montavon, G.; Orr, G.; Muller, K.R.: Neural networks: tricks of the trade (Vol. 7700). Springer. (2012)
67. Naeem, H.; Dong, S.; Falana, O.J.; Ullah, F.: Development of a deep stacked ensemble with process based volatile memory forensics for platform independent malware detection and classification. Expert Syst. Appl. **223**, 119952 (2023)
68. Nilsson, A.; Smith, S.; Ulm, G.; Gustavsson, E.; Jirstrand, M.: A performance evaluation of federated learning algorithms. In: Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning (2018)
69. O'Kane, P.; Sezer, S.; Carlin, D.: Evolution of ransomware. IET Netw. **7**(5), 321–327 (2018)
70. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
71. Pei, X.; Deng, X.; Tian, S.; Zhang, L.; Xue, K.: A knowledge transfer-based semi-supervised federated learning for IoT malware detection. IEEE Trans. Depend. Secure Comput. (2022)
72. Pikus, M.; Wąs, J.: Using deep neural network methods for forecasting energy productivity based on comparison of simulation and DNN results for central Poland-Swietokrzyskie Voivodeship. Energies **16**(18), 6632 (2023)
73. Plaut, D.C. (1986). Experiments on learning by back propagation.
74. Priddy, K.L.; Keller, P.E.: Artificial neural networks: an introduction (Vol. 68). SPIE Press (2005)
75. Rani, C.J.; Devarakonda, N.: An effectual classical dance pose estimation and classification system employing convolution neural network-long short-term memory (CNN-LSTM) network for video sequences. Microprocess. Microsyst. **95**, 104651 (2022)
76. Rasamoelina, A.D.; Adjailia, F.; Sincak, P.: A review of activation function for artificial neural network. In: 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI) (2020)
77. Regulation, G.D.P.: General data protection regulation (GDPR). Intersoft Consulting. Accessed in October 24(1) (2018)
78. Rey, V.; Sanchez, P.M.S.; Celdran, A.H.; Bovet, G.: Federated learning for malware detection in IoT devices. Comput. Netw. **204**, 108693 (2022)
79. Richardson, R.; North, M.M.: Ransomware: evolution, mitigation and prevention. Int. Manage. Rev. **13**(1), 10 (2017)
80. Robbins, H.; Monro, S.: A stochastic approximation method. Ann. Math. Stat. 400-407 (1951)
81. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386 (1958)
82. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)
83. Rustad, M.L.; Koenig, T.H.: Towards a global data privacy standard. Fla. Law Rev. **71**, 365 (2019)
84. Shafi, S.; Tariq, N.; Khan, F.A.; Ali, A.: Federated learning for enhanced malware threat detection to secure smart power grids. In: International Conference on Ubiquitous Computing and Ambient Intelligence (pp. 692-703). Cham: Springer Nature Switzerland (2024)
85. Shah, D.; Campbell, W.; Zulkernine, F.H.: A comparative study of LSTM and DNN for stock market forecasting. In: 2018 IEEE International Conference on Big Data (Big Data) (2018)
86. Shanker, M.; Hu, M.Y.; Hung, M.S.: Effect of data standardization on neural network training. Omega **24**(4), 385–397 (1996)
87. Sharma, S.; Sharma, S.; Athaiya, A.: Activation functions in neural networks. Towards Data Sci. **6**(12), 310–316 (2017)
88. Sherstinsky, A.: Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D **404**, 132306 (2020)
89. Sillman, J.: Analog implementation of the SoftMax function. arXiv preprint arXiv:2305.13649 (2023)
90. Simon, J.R.; Geetha, K.: Block mining reward prediction with polynomial regression, long short-term memory, and Prophet API for Ethereum blockchain miners. In: ITM Web of Conferences (2021)

91. Smith, D.; Khorsandroo, S.; Roy, K.: Supervised and unsupervised learning techniques utilizing malware datasets. In: 2023 IEEE 2nd International Conference on AI in Cybersecurity (ICAIC) (2023)
92. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
93. Stafford, T.F.; Urbaczewski, A.: Spyware: the ghost in the machine. Commun. Assoc. Inf. Syst. **14**(1), 49 (2004)
94. Szandała, T.: Review and comparison of commonly used activation functions for deep neural networks. Bio-Inspired Neurocomput. 203-224 (2021)
95. Taheri, R.; Shojafar, M.; Alazab, M.; Tafazolli, R.: FED-IIoT: a robust federated malware detection architecture in industrial IoT. IEEE Trans. Indust. Inf. **17**(12), 8442–8452 (2020)
96. Talukder, M.A.; Hasan, K.F.; Islam, M.M.; Uddin, M.A.; Akhter, A.; Yousuf, M.A.; Alharbi, F.; Moni, M.A.: A dependable hybrid machine learning model for network intrusion detection. J. Inf. Secur. Appl. **72**, 103405 (2023)
97. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A.: A detailed analysis of the KDD CUP 99 data set. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications (2009)
98. Tham, C.-K.; Yang, L.; Khanna, A.; Gera, B.: Federated learning for anomaly detection in vehicular networks. In: 2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring) (2023)
99. Tirumala, S.S.; Sarrafzadeh, A.; Pang, P.: A survey on internet usage and cybersecurity awareness in students. In: 2016 14th Annual Conference on Privacy, Security and Trust (PST) (2016)
100. Tu, X.; Zhu, K.; Luong, N.C.; Niyato, D.; Zhang, Y.; Li, J.: Incentive mechanisms for federated learning: from economic and game theoretic perspective. IEEE Trans. Cogn. Commun. Netw. (2022)
101. Vinayakumar, R.; Alazab, M.; Soman, K.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S.: Deep learning approach for intelligent intrusion detection system. IEEE Access **7**, 41525–41550 (2019)
102. Virusshare malware dataset: available at: https://virusshare.com/ (2018).
103. Vujicic, T.; Matijevic, T.; Ljucovic, J.; Balota, A.; Sevarac, Z.: Comparative analysis of methods for determining the number of hidden neurons in artificial neural networks. In: Central European Conference on Information and Intelligent Systems (2016)
104. Xu, S.; Chen, L.: A novel approach for determining the optimal number of hidden layer neurons for FNNs and its application in data mining (2008)
105. Yan, G.; Wang, H.; Li, J.: Seizing critical learning periods in federated learning. In: Proceedings of the AAAI Conference on Artificial Intelligence (2022)
106. Yang, Q.; Liu, Y.; Chen, T.; Tong, Y.: Federated machine learning: concept and applications. ACM Trans. Intell. Syst. Technol. **10**(2), 1–19 (2019)
107. Yao, K.; Cao, X.: UAV land classification method based on federated learning. Acad. J. Comput. Inf. Sci. **5**(7), 73–78 (2022)
108. Zhang, X.; Hong, M.; Dhople, S.; Yin, W.; Liu, Y.: FedPD: A federated learning framework with adaptivity to non-IID data. IEEE Trans. Signal Process. (2021)
109. Zhou, Y.; Jiang, X.: Dissecting android malware: Characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy (2012)
110. Zhu, H.; Xu, J.; Liu, S.; Jin, Y.: Federated learning on non-IID data: a survey. Neurocomputing **465**, 371–390 (2021)