

Federated inception-multi-head attention models for cyber-attacks detection

Imad Tareq AL-Halboosi¹, Bassant M. Elbagoury^{1,2}, Salsabil El-Regaily¹, El-Sayed M. El-Horbaty¹

¹Department of Computer Science, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

²Faculty of Computer Science and Computer Engineering, King Salman International University, El Tor, Egypt

Article Info

Article history:

Received Nov 24, 2023

Revised Feb 20, 2024

Accepted Mar 21, 2024

Keywords:

Artificial intelligence

Cybersecurity

Deep learning

Federated learning

Internet of things

ABSTRACT

With the proliferation of internet of things (IoT) devices, ensuring the security of these interconnected systems has become a critical concern. Cyberattacks targeting IoT devices pose significant threats to individuals and organizations due to the generation of vast amounts of data across many connected devices, which traditional centralized methods cannot solve. Federated learning (FL) could be a promising solution to mitigate privacy concerns associated with centralized approaches and address cybersecurity concerns. This paper uses FL and deep learning (DL) approaches to cybersecurity in IoT applications. The goal of cyber security is achieved by forming a federation of acquired and shared models at the head of the various participants. We use inception time and multi-head attention (CNN) algorithm based on FL to detect cyber-attacks and avoid data privacy leaks under two distribution modes, namely IID and Non-IID. In contrast, the FedAvg and FedMA algorithms aggregate local model updates. A global model is produced after several communication rounds between the IoT devices and the model parameter server. Cyber threats are simulated using edge-IIoT datasets. Experiment results show that the federated inception model's best global accuracy was 93, 91%, and 93, 49% using multi-head attention.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Imad Tareq AL-Halboosi

Department of Computer Science, Faculty of Computer and Information Sciences, Ain Shams University

El-Khalyfa El-Mamoun Street Abbasya, Cairo 11566, Egypt

Email: emadtariq1982@gmail.com

1. INTRODUCTION

In recent years, governments, academia, and industry have paid close attention to the internet of things (IoT) in many fields, such as healthcare, drones, smart cities, and cyber-physical systems. As IoT devices grow, vast amounts of data that include users' private information will be generated. Without reliable security systems implemented on IoT devices, it can affect personal privacy and may be exposed to many cyber-attacks. Regardless of the benefits provided by machine learning (ML) approaches, its capacity to identify threats and extract meaningful and complicated data models stands out [1]. Traditional ML approaches collect data on a centralized server while ignoring privacy and security considerations. Because of the volume and sensitivity of information transmitted between devices in IoT applications, this problem might be amplified. In addition to protecting privacy, one of the primary issues with IoT devices is that IoT networks have set needs for processing resources, which poses a significant barrier. As a result, one solution is to handle data in a decentralized manner [2], [3].

Federated learning (FL) is the most promising solution to this issue. In FL, training ML models is a collaborative effort across several clients that do not require the local data to be transported to a centralized

location, reducing storage costs transmission while retaining a high level of user privacy. This allows businesses to create a shared global model without storing training data in a centralized location [4]. FL enables multiple actors to collaborate on developing a single, robust system without sharing data, addressing critical issues such as data privacy, access rights, security, and access to diverse data. Instead of a pool to update the device, FL obtains the current model and computes a modified model on the system. These locally trained models are then sent from the machines to the central server and aggregated before being distributed to the devices as a unified and enhanced global model [5], [6].

Google already uses FL, allowing incredible predictive input features for the Android keyboard, on-device search phones, and many other applications. Recent developments have concentrated on lowering statistical barriers and improving FL security [7], [8]. With decentralized training data, top service providers have used FL approaches to support privacy-sensitive systems. FL outperforms centralized, traditional systems and the costs and risks associated with sensitive data management by excelling in bandwidth and power-constrained contexts and offering a simple, efficient platform for scaled personalization. It also gives users ownership over their data, allowing developers to design creative apps that use data insights [9]. In centralized learning, data for learning models are collected from multiple sources and then connected to a cloud server to create a standard model that can be deployed and shared across devices. Figures 1(a) and 1(b) shows the difference between centralized and FL. However, there are various limitations when applying FL in IoT applications, including the reliability of the learning model. If the global model is maliciously modified or broken, the update of all local models will be negatively affected [10]. Therefore, methods for using deep FL are fundamental in this context.

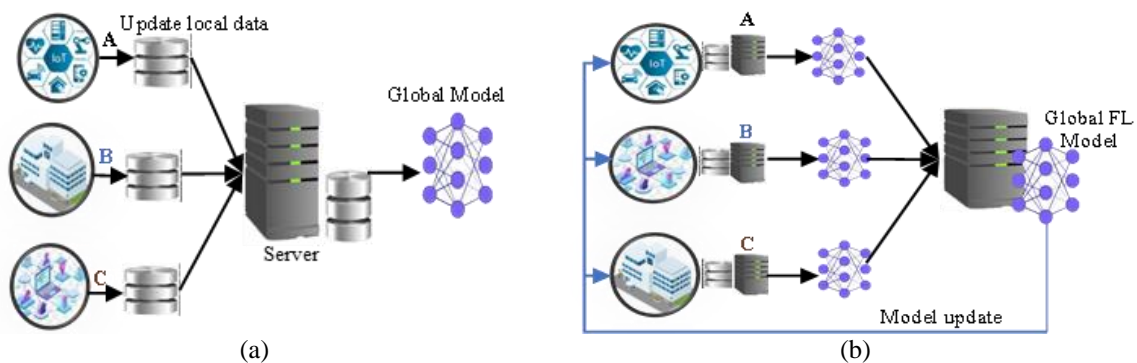


Figure 1. Architecture of (a) centralized learning and (b) FL process

In this study, we use two models of deep learning (DL) approaches based on FL to detect cyber-attacks (hybrid inception time and multi-head attention (CNN)) on the edge-industrial internet of things (IIoT) datasets. Using two types of distribution: non-independent and identically distributed (non-IID) and independent and identically distributed (IID). The federated averaging algorithm (FedAvg) and federated matched averaging (FedMA) algorithms, on the other hand, aggregate local model updates. The following are our research contributions:

- We provide tests evaluating the proposed model's performance detecting cyber-attacks on IoT devices. The following situations were compared to that goal: i) a centralized strategy in which all data is shared; and ii) a federated approach in which a global model is developed while local model changes are exchanged.
- Detection of cyber-attacks and IoT architecture federated to solve centralized problems, such as single points of failure, and preserve the privacy of the locally trained data, which is required for diverse applications.
- To our knowledge, we are the first to use the hybrid inception time and multi-head attention (CNN) model with FL to detect cyber-attacks on IoT devices.
- We are the first to compare synthetic minority oversampling method (SMOT) and class-weight techniques based on FL.
- We use the FedAvg and FedMA algorithms to aggregate and compare local model updates, note that there is no previous study that compares this.

The rest of this work is structured as follows. The related work is summarized in section 2. Section 3 discusses the methods. Section 4 describes the system design. Section 5 describes the experimental results. Finally, section 6 conclusion and discussion.

2. RELATED WORK

The increasing popularity of FL and the use of the internet of things has created many interesting research paths, one of which is the discovery and classification of cyber-attacks in IoT devices, and there are many research works proposed to secure IoT networks from malicious attacks. In this part, we will go through the most recent research. Table 1 provides an overview of these suggestions.

The edge-IIoT dataset, proposed by Ferrag *et al.* [11], is an investigational resource that utilizes centralized intrusion detection and federated DL with standard assessment criteria. The research explored the prediction and detection efficacy of different threat models and cyber-attacks using binary, six-class, and fifteen-class categorization. The best results for the 15-class are achieved with IID accuracy of 93.89% and the method of non-IID accuracy of 91.45%. Tabassum *et al.* [12] present FEDGAN-IDS, a FL intrusion detection system (IDS) based on generative adversarial networks (GAN) for detecting cyber-attacks in intelligent internet of things systems such as smart homes, smart cities, and intelligent e-healthcare systems. Extensive testing of the distributed intrusion detection model's accuracy, performance, and convergence utilizing three standard datasets: UNSW-NB15, KDD99, and NSL-KDD.

According to Campos *et al.* [13], a FL-enabled IDS technique based on a multiclass classifier is evaluated for detecting various attacks in an IoT scenario. Using three distinct settings obtained by partitioning the ToN-IoT dataset according to attack type and IoT device IP address. In addition, the International Business Machines (IBM) federated learning framework is used to implement and assess the impact of various aggregation functions. Instead of the naïve FedAvg, Zhang *et al.* [14] suggest fed detect, a FL algorithmic framework for on-device anomaly data detection that employs an adaptive optimizer and a cross-round learning rate scheduler. Use the N-BaIoT dataset to assess the Fed IoT platform and the fed detect algorithm regarding model and system performance. Popoola *et al.* [15] propose a FL strategy for detecting zero-day botnet attacks, and an appropriate deep neural network architecture is used for network traffic classification in IoT devices to limit data privacy leakage to aggregate local model updates. The FedAvg method is employed. Mothukuri *et al.* [16] suggested a threat detection and classification technique in IoT networks based on FL. The authors combine gated recurrent units (GRUs) and random forest (RF) models to form their ensemble. In other words, they used RF to integrate the GRU model predictions to improve the classification performance. Fed-IIoT, a federated technique for identifying Android malware in IIoT, was created [17]. Fed-IIoT, in particular, creates and injects hostile material into the dataset using two GAN models. The server uses a GAN to detect malicious models and reject polluted data. According to Nuaimi *et al.* [18], multiple IDSs are developed using conventional data analytics methods and their performance on Edge-IIoT is evaluated, and IDSs are compared to previous work to illustrate that highly accurate binary-class IDSs can be produced, however multi-class IDSs require careful treatment.

Table 1. A summary of previous studies

Authors	Years	Data	Model	Description
Ferrag <i>et al.</i> [11]	2022	Edge-IIoTset	Deep FL	Centralized and federated DL is used to investigate the detection efficacy and traffic predictability of different cyber-attacks and threat models.
Tabassum <i>et al.</i> [12]	2022	NSL-KDD, UNSW-NB15, KDD99.	FedGAN	Federated DL IDS based on GAN for detecting cyber threats in intelligent IoT systems.
Campos <i>et al.</i> [13]	2021	ToN-IoT	Logistic regression	An FL-enabled IDS technique based on a multiclass classifier is assessed for identifying various threats in an IoT context.
Zhang <i>et al.</i> [14]	2021	N-BaIoT	Deep Autoencod	Propose fed detect, an FL algorithmic framework for on-device anomaly data detection.
Popoola <i>et al.</i> [15]	2021	Bot-IoT, N-BaIoT	DNN	The federated DL approach for zero-day botnet attack detection is proposed to limit data privacy leakage in IoT devices.
Mothukuri <i>et al.</i> [16]	2021	-	LSTM/GRU	Proposed an ensemble federated-based attack detection and classification method, combining RF and GRUs models.
Taheri <i>et al.</i> [17]	2020	Drebin, Genome, Contagio	FedGAN	Developed Fed-IIoT, an FL approach for detecting Android malware in IIoT.

There is no previous study that uses the Inception time model with FL to detect cyber-attacks on IoT devices. However, our model showed high accuracy in detecting attacks compared to previous studies. Attention architectures are used in natural language processing, and we have used them to detect cyber-attacks on IoT devices. In this paper, we apply two methods to detect modern cyber-attacks on internet of things devices. The first method introduces FL with DL approaches as a (hybrid inception time and multi-head attention (CNN) algorithm) to detect cyber-attacks in IoT devices using the Edge-IIoT database to aggregate local model updates in FL; we compare the FedAvg and FedMA algorithms.

3. THE PROPOSED METHODS

3.1. Federated-inception time

This section describes the architecture of the enhance Inception time model (inception time v2-inception time v3). We employ class weights and smote after collecting and processing data. To avoid overfitting, the dataset is divided into testing and training. The Adam and stochastic gradient descent (SGD) optimizer uses an optimization technique rather than the standard stochastic gradient descent approach to repeatedly update network weights based on training data. With a depth of six and four layers and a filter of 32, with a short layer over three, inception time is employed. Three kernels will be generated for each inception module from $40/(2^i)$ as i increases from 0 \rightarrow 3.

We employed an inception time model with a one-dimensional (1D) input vector form in the inception network. Each block has three inception time modules. A bottleneck layer reduces the number of parameters and processing costs by lowering the input dimensions (i.e., the depth). A 1D sliding filter can filter out its discriminating zone in a time series. Improves generalization and speeds up training. The bottleneck output feeds three one-dimensional convolution layers with 10, 20, and 40 kernel sizes. Furthermore, the inception time module's inputs pass through the first layer consisting of (CNN 1D, filters=32, kernel=10, CNN1D, filters=32, kernel=10, Max-Pooling 1D, pooling size=20, stride=1, CNN1D, filters=32, kernel=10). Second layer (CNN1D, filters=32, kernel=20, CNN1D, filters=32, kernel=40, CNN1D, filters=32, kernel=10). Padding=same, activation=rectified linear unit (ReLU) each all. The depth concatenation layer, relu activation layer, and batch normalization layer sequence the outputs of the four convolution layers along the depth dimension. All layers, except the sequence layer, have the same stride and padding. As illustrated in Figure 2, all convolution layers have 32 filters, and residual connections are employed for every third inception time module.

Every three modules will make a residual layer consisting of CNN1D (filter=128, kernel=1, padding=same) batch normalization; after that, we will add the output from the residual and the last inception module with add a layer. Then, the activation layer with relu repeats this operation with depth number. Ultimately, we add a global average pooling layer dense layer with units the equal number of classes with activation function softmax. Figure 3 shows the residual layer.

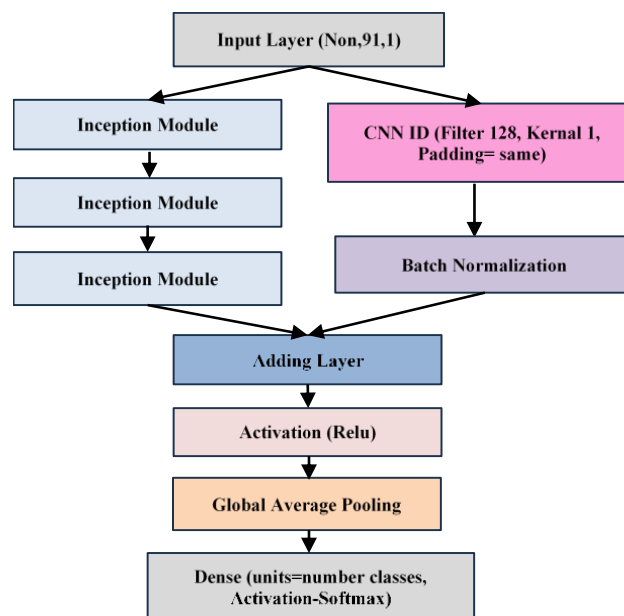


Figure 2. The custom inception model

3.2. Federated-multi-head attention

This section describes the architecture of the multi-head attention (CNN) model. The attention mechanism establishes a weighted average of (sequence) elements, the weights of which are dynamically generated depending on the elements' keys and an input query. The goal is to average the properties of several components. Rather than giving each element an identical weight, we prefer to weigh them based on their underlying values. In other words, we need to know which inputs we want to "attend" to more than others; further information is provided as shown in Figure 3.

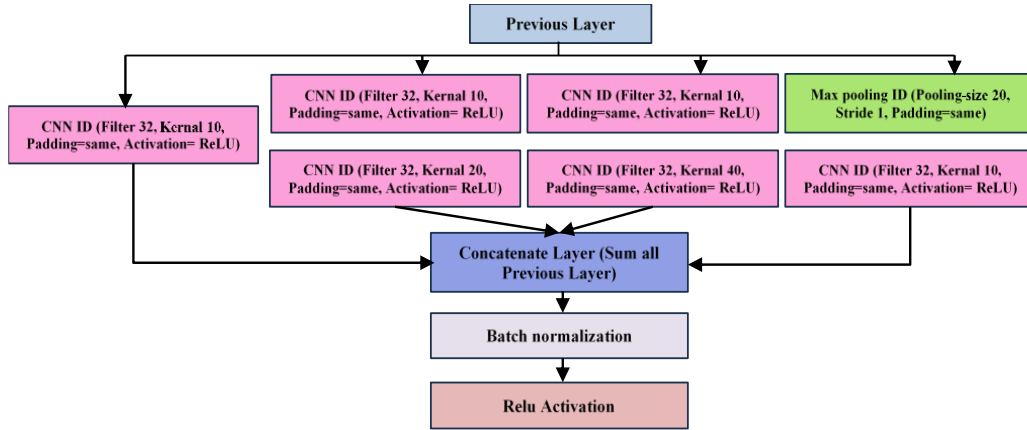


Figure 3. The residual layer (inception module)

3.2.1. Attention

The attention mechanism covers a novel category of layers in neural networks that has sparked considerable interest in recent years, particularly in sequence tasks. The attention mechanism specifies a weighted average of (sequence) items, the weights of which are determined dynamically based on the input query and the elements' keys. His attention consists of parts (query, keys, values, score function), you can learn about it in more detail in the following sources [19]–[23]. A softmax calculates the average weights across all score function outputs. Assign more significant importance to those value vectors whose matching key is more comparable to the query, which can be calculated as (1):

$$a_i = \frac{\exp(\int \text{attn}(\text{key}_i, \text{query}))}{\sum_j \exp(\int \text{attn}(\text{key}_j, \text{query}))}, \text{out} = \sum_i a_i \cdot \text{value}_i \quad (1)$$

Most attention methods differ in the queries utilized, the key and value vectors' definitions, and the score function employed. Self-attention refers to the attention applied within the architecture. Each sequence element in self-attention provides a query, value, and key. We conduct an attention layer for each element, checking the similarity of all sequence members' keys based on its query and returning a separate, averaged value vector for each element.

3.2.2. Scaled-dot product attention

Scalable-dot product attention is the fundamental idea of self-attention. We want to create an attention mechanism that allows any element in a sequence to pay attention to any other element while being computationally efficient. The dot-product attention is fed a series of inquiries [24].

$Q \in \mathbb{R}^{T \times d_k}$, keys $K \in \mathbb{R}^{T \times d_k}$ and values $V \in \mathbb{R}^{T \times d_v}$, T is the sequence length, and d_v and d_k denote the hidden dimension for queries V values and K keys, respectively. The similarity of the key K_j and query Q_i , using the dot-product as the similarity measure, determines the attention value from j to i as shown in Figure 4.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \quad (2)$$

The matrix combination QK^T computes the dot product for all conceivable combinations of keys and queries, resulting in a $T \times T$ matrix. The attention logits for a single element to all other components in the sequence are depicted in each row. To obtain a weighted mean, apply a softmax and multiply by the value vector [25]. (The attention determines the weights.)

3.2.3. Multi-head attention

Multi-head attention is a module that executes an attention mechanism in parallel. A network can attend to a sequence using scaled-dot-product attention [26]. A single weighted average is ineffective when a sequence element has to address many issues. As a result, we broaden the attention techniques to include many heads, i.e., numerous searches on the same features with distinct key-value triplets. We precisely restructure a query, value, and key matrix into sub-queries, sub-values, and sub-keys that are then individually run through the scaled-dot product attention [27]. Afterward, the heads are concatenated and mixed with a final weight matrix, As shown in Figure 5. This procedure may be expressed as follows:

$$\text{Multihued}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O \quad (3)$$

Where $\text{head}_i = \text{attention}(QW_i^Q, KW_i^K, VW_i^V)$ that is referred to as the multi-head attention layer with learnable parameters. $W_{1 \dots h}^Q \in \mathbb{R}^{D \times d_k}$, $W_{1 \dots h}^K \in \mathbb{R}^{D \times d_k}$, $W_{1 \dots h}^V \in \mathbb{R}^{D \times d_v}$ and $W^O \in \mathbb{R}^{h \cdot d_k \times d_{out}}$ (D being the input dimensionality).

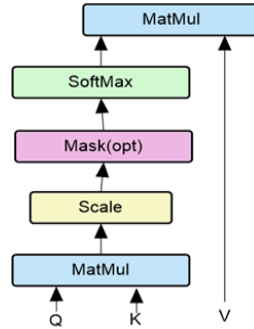


Figure 4. An illustration of the scaled-dot-product attention. A weighted sum value generates the result, with weights defined by the dot-product to query all keys [25].

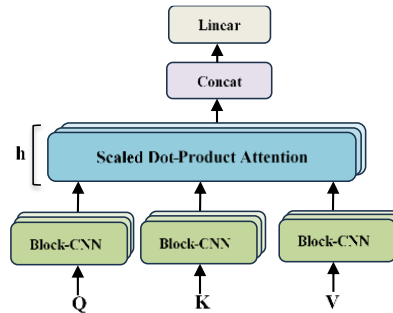


Figure 5. The multi-head attention. The attention mechanisms are activated several times in parallel

Each multi-head attention block consists of four successive levels. Three CNN (dense) layers accept keys, queries, or values in the first level. A scaled dot-product attention function is used on the second level. The processes on the first and second levels are repeated h times in parallel, depending on the number of heads in the multi-head attention block. On the third level, a concatenation process connects the outputs of the several heads. A final linear (dense) layer produces the output on the fourth level. The following are the primary components of attention:

- k and q are dimension vectors, d_k , carrying the keys and queries.
- V denotes a dimension vector, d_v , containing the values.
- K , Q , and V are matrices that bundle together keys, queries, and values.
- WK , WQ , and WV generate distinct subspace representations of the key, query, and value matrices.
- WO , designating a multi-head projection matrix.

The attention function is a mapping from a query to a set of key-value pairs and then to an output. The result is a weighted sum of the values, with the weight assigned to each value determined by the compatibility function of the query with the relevant key. In this study, we use one-dimensional convolutions with multi-head CNNs, where the dimension dictates how it analyzes input data. To handle the input, the single-channel CNN employs a single convolutional head with a single channel. It processes each process variable individually using independent single-channel convolutional heads. Each block CNN Branch will include a sequential layer of Conv1D (filters 32, kernel size 3), Conv1D (filters 16, kernel size 5), and Conv1D (filters 8, kernel size 7). The hyperparameters are batch size 64, epochs 50, initial learning rate 0.5, and end learning rate 0.008. The multi-head CNN story is seen in Figure 6.

One advanced use of multi-head attention is in the context of self-attention mechanisms; the input sequence is processed by attending to different parts of the sequence to compute a weighted sum of its elements, with the weights determined by the similarity between the elements. Multi-head attention allows

the neural network to govern the mixing of information between segments of an input sequence, resulting in richer representations and improved performance on ML tasks.

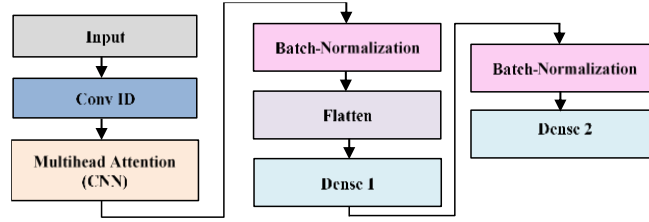


Figure 6. The multi-head CNN structural

3.3. Federated learning process

The FL structure consists of multiple clients and one server; mainly, the server provides a global shared model. In each communication between the server and the client, the client downloads the model and trains with local datasets while updating model parameters. The server receives the current model parameters distributed to each client following training, and then the updated model parameters are uploaded to the server [28].

We tested two methods to aggregate client model parameters; the first used the FedAvg method [29]. The server is principally controlled by three essential parameters: C, the proportion of clients; B, the local batch size for client updates; and E, the number of local batch updates. The aggregation server initially chooses a C percentage of K clients to join the FL process and execute R\ FL rounds. At random, the aggregation server builds a generic model with a random set of initial weights w. The generic model is then obtained by each client k from the aggregate server. Every client retrains the generic model locally using its data and computes a new set of weights for the newly created local model. They applied the following (4) to update the model weights.

$$W_{t+1} = \sum_{k=1}^K \frac{nk}{n} w_{t+1}^k \quad (4)$$

Where n denotes the total size of all client datasets and nk the size of each. W_{t+1} It is the iteration's updated global model. The modified model has been distributed to the clients. After that, the server combines all client parameters ($k=1$). The new global model is subsequently sent to the clients, who utilize the updated parameters to enhance the global model. This procedure is continued until the model has reached convergence. The procedures used to train the various customer sets are depicted in Algorithm 1.

In the second tested method, we used the FedMA algorithm [30]. The FedMA algorithm employs the following layer-wise matching scheme. To begin, the data center collects just the weights of the first layers from the clients and conducts one-layer matching to generate the federated model's first-layer weights. A data center then broadcasts these weights to clients, who subsequently train all subsequent layers on their datasets while freezing the matching federated layers. This technique continues until we reach the last layer, where we do weight averaging based on the class proportions of data points per client. The FedMA technique necessitates communication rounds equal to the number of network layers. Local clients receive the matched global model at the start of a new round and reconstruct their local models with the same size as the original local models based on the previous round's matching results. In contrast to the naive method of using a wholly matched global model as a starting point across clients on each cycle, this procedure allows us to keep the size of the global model short. FedMA identifies matched sets of convolutional filters and averages them to form the global convolutional filters. FedMA's matched and global filters extract the same feature from the incoming data. The FedAvg's global filter is the average of the client's filter. Figure 7 shows the article FedMA algorithms. Algorithm 1: FedAvg, considering K clients B represents the local mini-batch size, E represents the number of local epochs, the number of global rounds is R, C represents the proportion of clients and is the learning r .

Algorithm 1: Federated learning (FedAvg) [29]

1. Server (K,C,R)
2. $W1 \leftarrow$ generic model ()
3. For each round $t = 1, \dots, R$ do
4. $M \leftarrow \max(C.K, 1)$
5. $S_t \leftarrow$ (random set of m clients)
6. For each client $K \in S_t$ in Parallel, do

7. $w_{t+1} \leftarrow w_t + \eta \Delta f(w_t, K)$ client update (wt. K)
8. End
9. $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$
10. End
1. Client (w,k)
2. $B \leftarrow (\text{split } P \text{ into batches of size } B)$
3. For each local epoch i from 1 to E, do
4. For batch $b \in B$, do
5. $W \leftarrow W - \eta \Delta f(W, b)$
6. End
7. End
8. Send w to server

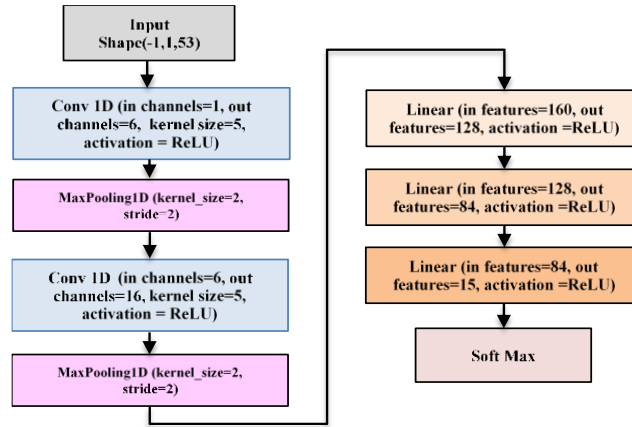


Figure 7. The FedMA algorithm structure

In this work, CNN makes use of one-dimensional convolutions, where the dimension dictates how it analyzes input data. The size of the parameters used in our experiment, batch_size=512, learning_rate=0.01, retrain_lr=0.01, fine_tune_lr=0.01, epochs=5, retrain_epochs=10, fine_tune_epochs=10, partition_step_size=6, local_points=5000, n_nets=10 (number of clients). used optimizer SGD. The latest algorithm on FL was done on FedMA in 2020, so we tried to experiment. Algorithm 2 summarizes our FedMA.

Algorithm 2: Federated learning (FedMA) [30]

Input: local weights of N-layer architectures $\{W_j^{(1)}, W_j^{(2)}, \dots, W_j^{(N)}\}_{j=1}^J$ from J clients

Output: global weights $\{W^{(1)}, W^{(2)}, \dots, W^{(N)}\}$ $n = 1$;

while $n \leq N$ do

if $n < N$ then

$\{\Pi_j^{-1}\}_{j=1}^J = \text{BBP} - \text{MAP}(\{W_j^{(n)}\}_{j=1}^J)$;

$W^{(n)} = \frac{1}{J} \sum_j W_j^{(n)} \Pi_j^{-1}$;

else

$W^{(n)} = \sum_{k=1}^K \sum_i p_{ik} W_{ji}^{(n)}$ where p_k is a fraction of data points with labels k on the worker j ;

End

for $j \in \{1, 2, \dots, J\}$ do

$W_j^{(n+1)} \leftarrow \Pi_j^{(n)} W_j^{(n+1)}$; // permute the next-layer weights

Train $\{W_j^{(n+1)}, \dots, W_j^{(L)}\}$ with $W^{(n)}$ frozen;

End

$n = n + 1$;

End

Our final algorithm with federated average and models arch plus smote will be. The summary of the proposed Algorithm 3, in steps:

- Step 1: We split the IoT dataset into two groups, train (X_{train} , y_{train}) and validation (X_{val} , y_{val}) sets with a ratio of 20% for validation and remain for training with stratify option.
- Step2: Normalize X_{train} , X_{val} with MinMax normalization techniques
- Step3: We have two options to overcome the imbalance problem using class weights or smote techniques
- Step 4: Sampling our train dataset between n clients with two techniques: IID or non-IID
- Step 5: Batch size each n client dataset to batches with size batch size
- Step 6: We reduce the optimizer-learning rate every five epochs with 0.5
- Step 7: We keep tracking the global model accuracy to get the best model weights.

Algorithm 3: Process federated learning

Input: IoT Dataset, train Data Mode, n Clients, sampling Technique, batch-size, common Rounds, init Learning Rate

Output: Model Weights

1. Divide IoT Dataset into training and validation set
2. Normalize X_{train} , X_{val}
3. Check the imbalance dataset to solve
 - If train Data Mode = SMOTE:
 - Upsampling the X_{train} Dataset
 - Set class-Weights <- None
 - Else
 - Class-Weights <- Get Class Weights for y_{train}
4. Check the sampling technique to split the data between n Clients
 - If sampling Technique == IID:
 - Set clients dataset <- Split train data set between n Clients with Identically Distributed
 - Else
 - Set clients dataset <- Split train data set between n Clients with Non-Identically Distributed
5. Set clients Batched <- Split each n Clients in client's dataset to batches with size batch Size
6. Initialize the global model
7. Set best Global Model Weights <- global Model Get Weights ()
8. Set best Accuracy Result <- 0
9. Set learning rate <- init Learning Rate
10. Initialize optimizer with learning Rate
11. Set loss Func <- loss function to be Categorical Cross-entropy
12. Set evaluation Metrics <- Evaluation metrics (accuracy, precision, recall, f1-score)
13. For each round in common Rounds:
 - Set global Weights <- global Model Get Weights ()
 - Set client Names <- get client name from clients Batched in a list of clients batched
 - Shuffle client Names
 - Set learning rate <- lrScheduler (learning Rate, round)
 - Set optimizer learning rate <- Learning Rate
 - Set empty round Model Weights list
 - For a client in client Names:
 - Initialize the local model
 - Build local model with (input shape, num Classes)
 - Compile local Model With (optimizer=optimizer, loss=loss Func, metrics=evaluation Metrics)
 - Set Local model <- Local model set weights (global Weights)
 - Fit local Model with (clients Batched[client], class Weights)
 - Set local Model Weights <- get local Model Get Weights ()
 - Append local Model Weights to round Model Weights
 - Set average Weights <- average the round Model Weights list
 - Set global model <- global model. set weights (average Weights)
14. Set current Global Model Accuracy <- Evaluate the global Model with (X_{val} , y_{val})
15. Check for the best global model weight
 - If current Global Model Accuracy > best Accuracy Result:
 - Set best Global Model Weights <- global Model Get Weights ()
16. Return best Global Model Weights.

3.4. Multi-stage framework of trusted federated learning

The multi-stage architecture of the FL execution, as illustrated in Figure 8, may be separated into two stages: training and prediction. At each stage of FL execution, the model encounters security and privacy concerns. FL needs appropriate security and privacy safeguards at each stage.

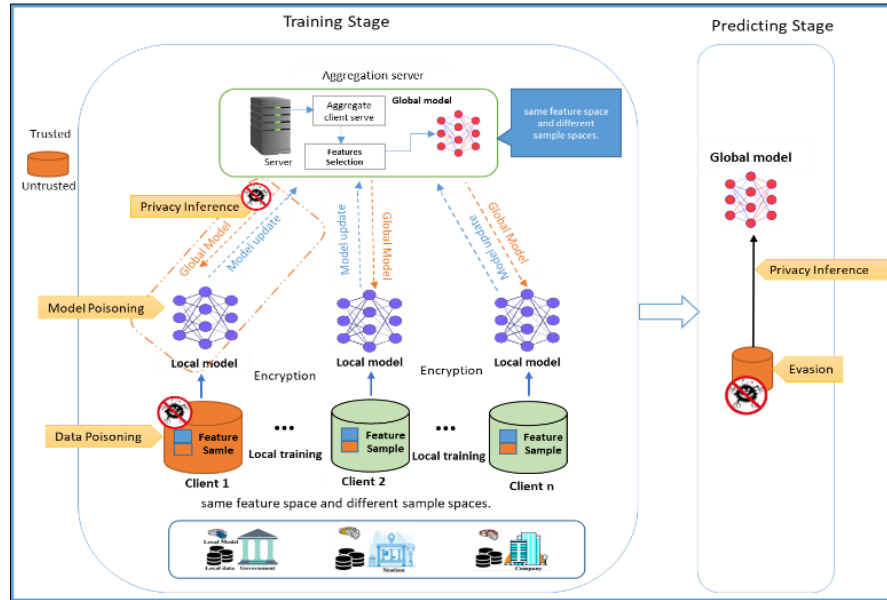


Figure 8. The models of the multi-stage FL, including the training models

3.4.1. Training stage

FL needs the collaboration of numerous local IoT devices to train a global model. Malicious IoT can modify its data, model gradients, and parameters during the model-training step. As a result, if adversaries penetrate the IoT devices, they can disrupt the integrity of the training dataset or model, impairing the performance of the global model. Furthermore, the server can perform passive or aggressive inference assaults. Furthermore, during the upload and download of model updates, intermediates in the communication route may attack the models, resulting in tampered with or stolen model updates. As a result, securing the communication of model updates between IoT devices and the server is critical.

3.4.2. Predicting stage

Once the model has been trained, the global model is delivered to all IoT devices, regardless of whether they took part in the training or not. Cybersecurity threats are common during this stage. Attacks typically do not alter the target model but trick it into giving incorrect predictions. After obtaining the optimal global model weights, we can broadcast to clients, make predictions for the test dataset, and then assess the outcomes using accuracy, precision, F1-score, recall, and loss.

4. SYSTEM DESIGN

4.1. Datasets

The cybersecurity Edge-IIoT dataset for IIoT and IoT applications is utilised in ML-based IDS to assess the models in this paper. The Edge-IIoT dataset covers fourteen distinct forms of attacks separated into five classifications (distributed denial-of-service (DDoS) and denial-of-service (DDoS) assaults, injection attacks, information gathering, malware attacks, and man in the middle (MITM) attacks). Edge IIoT [11] has 20,952,648 typical attack statistics, which comprise 11,223,940 normal and 9,728,708 attacks. We divided this dataset into 20% for testing and 80% for training, with the option of stratifying to maintain the percentages constant across all classes. The dataset yielded 61 features and 1,909,671 samples, 1,527,736 and 381,935 data sets are available for training and testing, respectively, which were divided into 15 categories as shown in Table 2. We used class weights since the data were imbalanced. Following various attempts to fine-tune the hyper-parameters, the batch size was set at 64, which is appropriate for memory that maintains the training process without fluctuation and converges with general performance. The depth was lowered to four or six for certain Edge-IIoT data because some datasets require a more sophisticated model to converge to a

satisfying F1-score, while others converge with a module depth of four or less. We attempted, however, to preserve the residual layer after every three modules. In terms of epochs, 50 is an overestimation that may not be attained since the models were terminated early if they began to overfit by tracing the validation accuracy. We employed the Adam, RMSprop optimizer, and a learning scheduler to converge rapidly and determine the optimal learning rate.

Table 2. The overall number of records in the Edge-IIoTset data set as well as the various kinds of records

IoT traffic	Kinds of event	Data record
Attack	DDoS-HTTP	38835
	DDoS-ICMP	54351
	DDoS-UDP	97253
	SQL-injection	40661
	DDoS-TCP	40050
	Vulnerability-scanner	40021
	Password	39946
	Ransomware	7751
	port-Scanning	15982
	Backdoor	19221
	Uploading	29446
	XSS	12058
	MITM	286
	Fingerprinting	682
Normal	Normal	1091198

4.2. Data pre-processing

Before the classification task, the data is pre-processed. The following tasks are completed during the pre-processing stage.

- Encoding: The classification values in each dataset are converted into numeric features to make them machine-readable.
- Min-max scalar: A data-preprocessing step used by several ML methods for numerical features. The lowest and maximum features are equivalent to zero and one, respectively. The min-max scaler decreases data within a defined range, often between zero and one. To modify data, it scales attributes to a given range. It fits the values within a specific range while keeping the original distribution's shape. The Min-Max scaling is carried out using the:

$$X_{std} = ((x - x.min(axis = 0)) / (x.max(axis = 0) - x.min(axis = 0))) \quad (6)$$

$$X_{Scaled} = x - std * (max - min) + min.$$

These datasets have many challenges, including missing values, behavioral replication, and superfluous features that degrade model performance. Edge-IIoT is divided into five classifications and is available in CSV format. We looked for missing data in examples like Nan and eliminated them, along with any duplicates. Furthermore, we removed static characteristics with the same value across the whole dataset (for example, Tls port, dns, icmp, unused, http, qry, msg_decoded_as, Type, MQTT).

- Feature selection: The importance of a feature is determined as the (normalized) total decrease of the criterion brought by that feature. After applying the extra trees as feature importance, we can reduce the features from 91 to 53 in the Edge-IIoT dataset.

4.3. Imbalanced classification problems

The number of instances in each class is called the class distribution. Imbalanced classification is a predictive modeling issue in which the number of instances in the training dataset for each class label is not balanced, resulting in a biased or skewed class distribution rather than equal or nearly equal. As we can see, we have an imbalance problem because not all classes are equally distributed in the dataset, so we must solve it using (Class weights, SMOTE).

4.4. Hyper parameters tuning

We tried many experiments with validation sets with 20% of trainsets to reach the best parameters, then we retrained on full trainsets and tested on the test set to give the evaluation as shown below.

- For epochs, 50 or a considerable number is just overestimated. It may not reach because we use early stopping if the model begins to overfit by tracing the validation accuracy, so some models stopped at 34 epochs.

- For the learning rate, we already used Adam optimizer + the learning scheduler to lower this number from 0.001 to less as model training sees that validation accuracy not increasing after waiting for epochs (#5 epochs) on centralized and will reduce automatic in federated.
- Early stopping was applied to stop training after ten epochs without change in validation accuracy (#10 epochs) only on centralized.
- The batch size was 64, suitable for memory, and converged smoothly without oscillations on training visualization. Moreover, as we go smaller, we go to general performance, so 64 was good enough only used with class weights models. However, in SMOTE, we go up to 512, the data set is almost close to the 16M sample, and 512, after several tries, was suitable for model converge and a model train.

4.5. Use cases

We train two deep FL models, an inception time and a multi-head attention (CNN), for cyber threat identification in IoT devices. The results are then compared to the centralized versions, non-IID-FL, and IID-FL). we carried out our experiments on Google Collab and kaggle, employing well-known libraries such as tensor flow and keras. Several open-source FL frameworks are available for simulating and testing FL algorithms. We used two use cases to evaluate our experiment, which is as follows:

4.5.1. Centralized learning strategy

The data is stored in a single location with a DL classifier, Such as the inception time and multi-head attention model. For the centralized model training, a reduced learning rate is applied to minimize the learning rate if there is no change in loss with factor 0.5 after (number epochs/10). A checkpoint was applied to save the lowest-loss model. Early stopping was applied to the validation accuracy to prevent the overfit after number epochs/10) without change in better accuracy. Batch size=64, Epochs=50, Start Learning rate=0.001.

4.5.2. Federated learning strategy

The data is distributed across multiple clients, and an aggregation server aggregates the client models. Use a custom schedule-learning rate for each client to reduce it after around 10% to get the best converge and lowest loss. We also used the same classifier as in the previous method. Over 50 FL rounds, we used an EdgeIoT Set of client distributions C=10, batch size=512, start learning rate=0.001, with two data distribution methods, non-IID and IID.

5. EXPERIMENTAL RESULTS

For each mission, various indicators and measurements can be used to evaluate any learning model, such as accuracy, false positive rate, precision, and detection rate, as well as their F1 score. True negative (TN), true positive (TP), false negative (FN), and false positive (FP) data were used to create these measurements. The improperly identified legitimate and attack vectors were FP and FN, respectively. TP and TN refer to the number of legitimate attack vectors successfully classified [31], [32].

Accuracy: the percentage of samples and applications correctly classified in a dataset. The higher accuracy value indicates that the classifier is accurate.

$$Accuracy = (TN + TP) / (FN + TP + TN + FP) \quad (7)$$

Precision: measures how many benign, positive samples and applications were correctly identified in the dataset. When the precision value of a classifier is higher, it performs better and is more desirable.

$$Precision = TP / (TP + FP) \quad (8)$$

F1-Score: The F1 score represents the balance of recalls and a classifier's precision in a single metric by taking the harmonic mean of these two values.

$$F1 - score = 2. (Recall * Precision) / (Recall + Precision) \quad (9)$$

Recall: this measure computes fraction of valid positive predictions made from all possible positive predictions.

$$Recall = TP / TP + FN \quad (10)$$

5.1. Performance evaluation

In this section, inception time and multi-head attention (CNN) were applied to the Edge-IIoT datasets to predict our tests. We evaluate the performance of the precision, accuracy, loss, F1-score, and recall. We also show the results of the cyber-attack detection model's multiclass categorization on IoT devices. We perform all experiments using EdgeIoT-set standard datasets: We perform all the necessary pre-

processing steps for training the centralized, non-IID-federated, and IID-federated models. We tried once with class weights and another with smote and compared them with each other and with other studies.

5.1.1. Results of centralized

For evaluation, we employed Inception-time and multi-head attention on the Edge-IIoTset dataset, partitioned into 80% training and 20% testing sets. This experiment aimed to compare the performance of two malware detection models utilizing the database containing 1,527,736 training data, 381,935 test data, and 53 feature selections to ensure the evaluation's reliability. We employed four multiclass measures: precision, accuracy, recall, and F1-score. Using class-weight our best accuracy was 94.90%, using inception time and with multi-head attention, the best accuracy was 93.83%, with SMOTE our best accuracy was 94.88%, using inception time. All results are shown in Table 3, and the confusion matrix is illustrated in Figures 9(a) to 9(c), which displays the percentage of assaults correctly predicted for fifteen classes. In addition, using a beginning learning rate of 0.001 and epochs of 50, a batch size of 64, each was determined independently. The learning rate at the end was 0.001.

Table 3. The evaluation results of centralized approaches

Model	Class weight Accuracy	Class weight Accuracy	Class weight Accuracy	Class weight Accuracy
Inception Time	94.90	94.90	94.90	94.90
Multi-head attention	93.83	93.83	93.83	93.83
Inception Time	94.80	94.80	Smote	94.80
				94.80

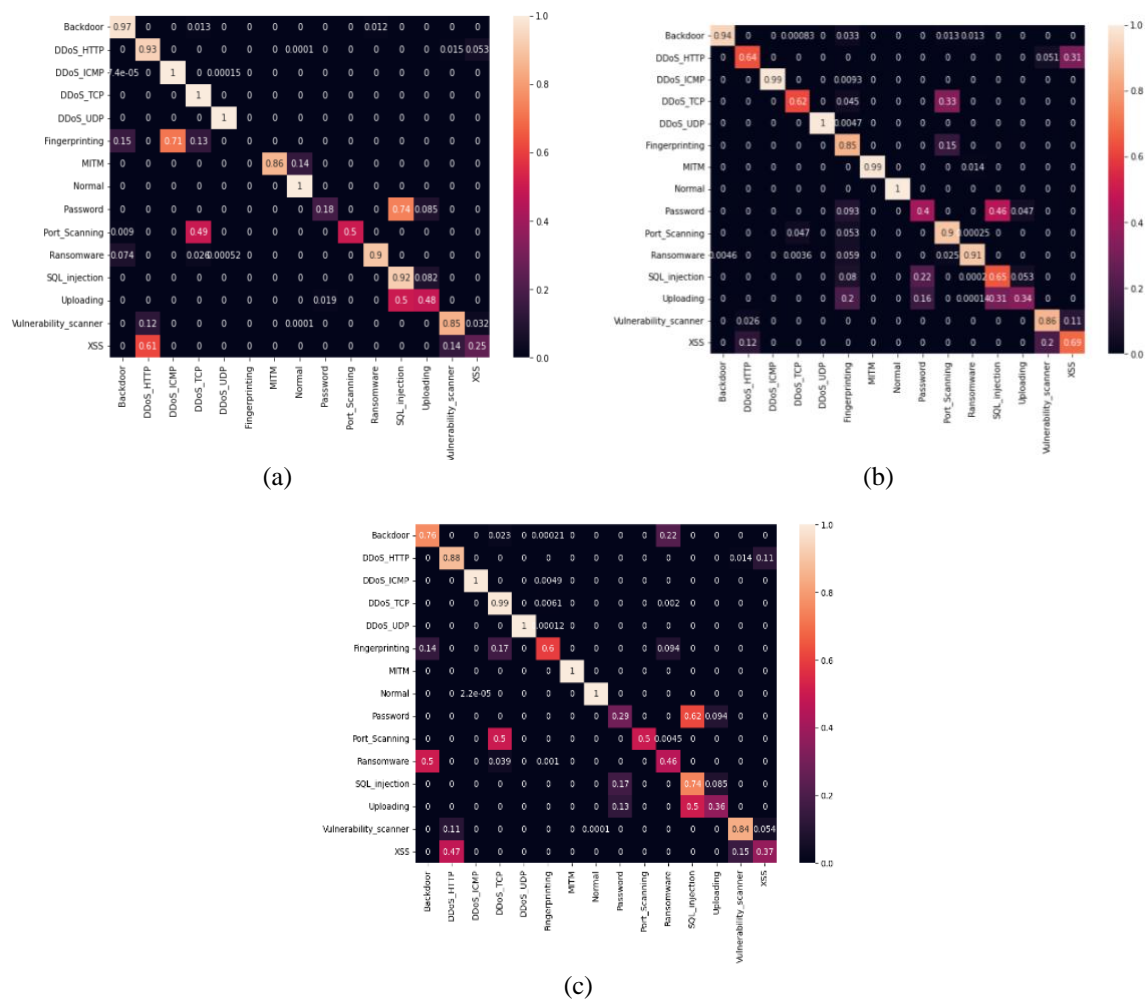


Figure 9. Confusion matrix for fifteen classes in the edge-IIoT-inception time and multi-head attention centralization; all values between 97 and 25 in class weight, with SMOTE 94 and 69 in (a) inception (class-weight), (b) inception (SMOTE), and (c) multi-head attention (CNN).

5.1.2. Result of federated

The evaluation results of the deep FL technique for multi-classification (fifteen-classes) (multi-classification) are presented in Table 4. In particular, the results present the global model accuracy. All these accuracies are obtained from several rounds of DL networks under the FL mode using FedAvg algorithms. Moreover, the results are obtained for IID modes and non-IID. In each SMOTE and class-weight with the number of clients $k=10$. Using Inception-time in the mode of (IID) class weight, the best global model accuracy achieves 93.85%, and using multi-head attention, the best global model accuracy reaches 93.49%. With the SMOTE, accuracy achieves 93.91%, With using inception time in the mode of (non-IID) class weight, the best global model accuracy achieves 93.90%, and using multi-head attention, the best global model accuracy reaches 91.57%. With the SMOTE, accuracy achieves 93.91%. With the FedMA algorithms Using multi-head attention in the mode of (IID) class weight, round stable after 10, the best global model accuracy achieves 92.15%; using (non-IID) class weight, the best global model accuracy achieves 90.33%, are presented in Table 5. Figures 10(a) to 10(d) shows the confusion matrix, which illustrates the proportion of attacks predicted correctly for fifteen classes.

Table 4. Evaluation results of the FL strategy employing fedavg algorithms with ten customers

Model	Type	Accuracy	Precision	F1-score	Recall
Inception time	IID-FED- class weight	93.86	96.96	94.29	91.76
	NON-IID-FED- class weight	93.83	93.83	93.83	93.83
	IID-FED- SMOTE	93.91	97.62	94.42	91.43
	NON-IID-FED- SMOTE	93.91	97.62	94.42	91.43
Multi-head attention	IID-FED- class weight	93.49	97.22	93.63	90.29
	NON-IID-FED- class weight	91.57	91.54	91.56	91.59

Table 5. Evaluation results of the FL strategy employing FedMA algorithms with ten customers, in the mode of (IID) class weight and using optimizer (SGD)

Model	Type	Accuracy	Precision	F1-score	Recall
Multi-Head Attention	IID-FED- Class weight	92.15	92.39	91.81	92.15
	NON-IID-FED- Class weight	90.33	89.59	90.55	90.69

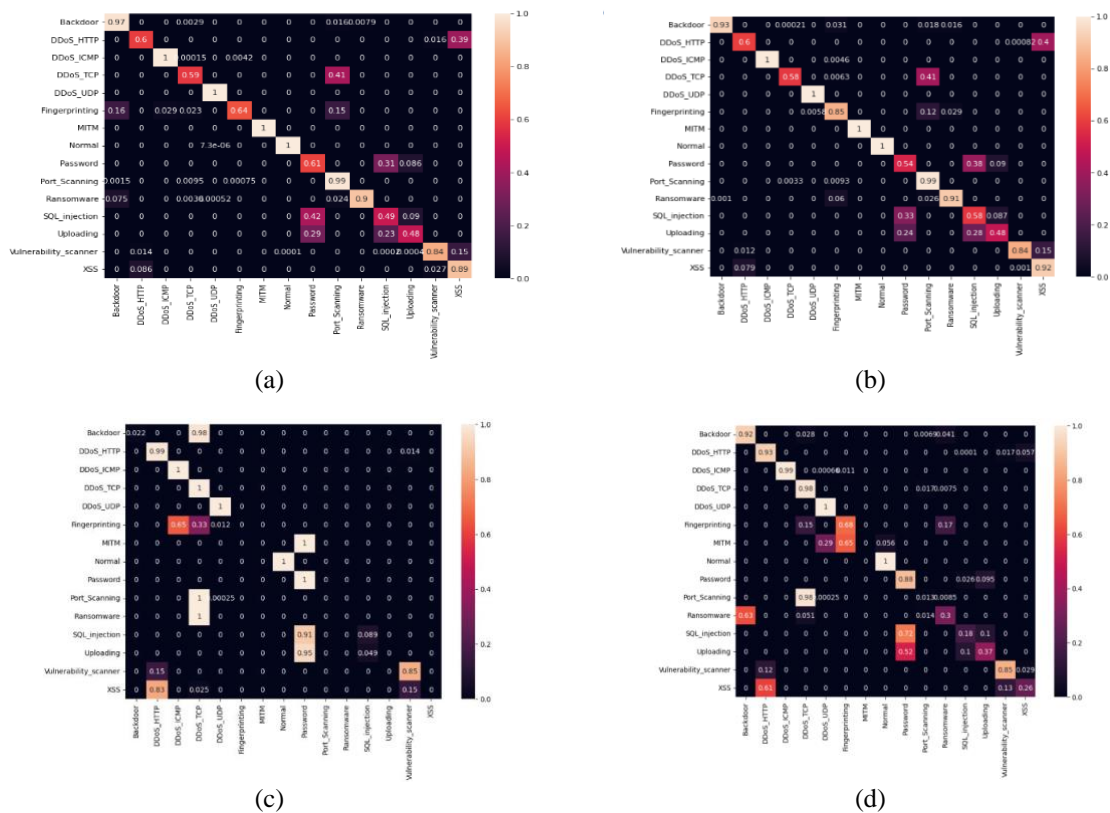


Figure 10. Confusion matrix for fifteen classes in the Edge-IIoT-inception time and multi-head attention federated; (a) Inception time (IID) class-weight, (b) Inception time (IID, non-IID) SMOT, (c) multi-head attention (non-IID), and (d) multi-head attention (IID).

Table 6 compares our results with previous studies using the same database but with different models, where we used the hybrid inception time and multi-head attention (CNN) with ten clients. We used SMOT and class-wight in centralization and FL in two cases (IID, non-IID). The results show the superiority of the inception-time model in centralization 94.90, and FL in the global model results with an accuracy of 93.91, with each of (IID, non-IID) better.

Table 6. Compares the inception time and multi-head attention models with previous studies

Authors	Model	Centralized	Federated			
			SMOTE		SMOTE	
			IID	NON-IID	IID	NON-IID
Ferrag <i>et al.</i> [11]	DNN	94.67	93.89	91.45	-	-
Rashid <i>et al.</i> [33]	CNN	93.9	91.13	90.73	-	-
	RNN		92.28	91.53	-	-
Ahakonye <i>et al.</i> [34]	CNN	-	90.83	-	-	-
Singh <i>et al.</i> [35]	ResNet	92.94	83	-	-	-
Proposed	Inception	94.80	93.91	93.91	93.86	93.83
	Multi-Head Attention	93.83	-	-	93.49	91.57

6. DISCUSSION

All previous FL outcomes used the FedAvg aggregates algorithm, as shown in Table 4. We also tested the CNN model with the FedMA aggregates algorithm, as shown in Table 5. Therefore, our results have been shown to outperform other related works, as shown in Table 6. To summarize the findings of our investigations, we reach the following conclusions. Federated vs centralized. In terms of accuracy, the centralized solution outperforms any federated option. This performance degradation is caused by the division of the entire dataset among the several clients. As a result, a centralized solution provides more accuracy at the expense of less privacy. Depending on how the aggregate data is partitioned between several clients, different federated setups with different privacy settings result in varying levels of accuracy. In general, the greater the privacy, the lower the level of accuracy. Because raising the level of privacy reduces the quantity of data accessible for model training, the federated strategy can help mitigate this loss, which is nevertheless visible in our data. However, when we compare the loss of accuracy in absolute terms, we can see that the loss is not very significant, and hence the adoption of a federated strategy is still viable.

In future work, we hope to improve the models' reliability when malicious edge nodes are on the network. Furthermore, we will concentrate on the filtering process used to detect poisoning attempts. The FL approach will likely cause privacy difficulties if there are untrusted servers or clients. Hence, more research on how to make FL more resilient to cyber assaults must be done. Develop an adaptive FL system that dynamically adjusts the allocation of computational resources based on cyber-attack severity. By continuously monitoring the network and device behavior, the system can identify periods of increased attack activity and allocate more resources to the detection model on affected devices. This adaptive approach improves the responsiveness and accuracy of cyber-attack detection.

Furthermore, several constraints impact the global model's accuracy, such as devices that cease operating, and delayed model updating or loading. In the future, acceptable solutions to the challenges that make the global model less accurate should be identified. These are just a few ideas to consider for future experiments. Before implementation, it is important to thoroughly research and evaluate any proposed approach's feasibility, effectiveness, and security implications.

7. CONCLUSION

This study investigated a method for accurately detecting cyber-attacks in IoT networks to protect sensitive data. Our experiments used two DL (hybrid inception time and multi-head attention) models based on federated and centralization learning models to detect unwanted cyber-attacks on the Edge-IIoT datasets.. we compare centralized and FL using the FedAvg and FedMA algorithms based on class weight and SMOT. In terms of training time and computing resources such as memory and GPU, class-weight is better while earlier studies did not show the difference between the federated and centralized model and focused only on the FedAvg algorithm using SMOT; in addition to that, we compared our method based on FL in both scenarios (IID, non-IID). There was an improvement in the global model results with an accuracy of 93.91 based on multiple classifications, with both (IID, non-IID) better; the results showed that through the federated approach, we could achieve a fairly competitive detection of cyber-attacks.

ACKNOWLEDGEMENTS

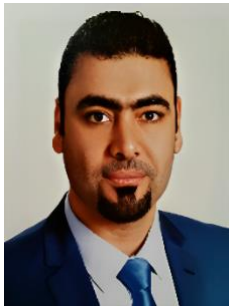
We would like to thank Eng. Mohamed Ahmed, for his tremendous efforts in coding this research, mohamed.ahm.cs@gmail.com.




REFERENCES

- [1] S. A. Rahman, H. Tout, C. Talhi, and A. Mourad, "Internet of things intrusion detection: centralized, on-device, or federated learning?," *IEEE Network*, vol. 34, no. 6, pp. 310–317, Nov. 2020, doi: 10.1109/MNET.011.2000286.
- [2] T. Iggena *et al.*, "Iotcrawler: challenges and solutions for searching the internet of things," *Sensors*, vol. 21, no. 5, pp. 1–32, 2021, doi: 10.3390/s21051559.
- [3] S. A. Jebur, K. A. Hussein, H. K. Hoomod, L. Alzubaidi, and J. Santamaria, "Review on deep learning approaches for anomaly event detection in video surveillance," *Electronics*, vol. 12, no. 1, 2023, doi: 10.3390/electronics12010029.
- [4] V. Rey, P. M. Sánchez, A. H. Celdrán, and G. Bovet, "Federated learning for malware detection in IoT devices," *Computer Networks*, vol. 204, 2022, doi: 10.1016/j.comnet.2021.108693.
- [5] B. Li, Y. Wu, J. Song, R. Lu, T. Li, and L. Zhao, "DeepFed: federated deep learning for intrusion detection in industrial cyber-physical systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5615–5624, Aug. 2021, doi: 10.1109/TII.2020.3023430.
- [6] M. A. Ferrag, O. Friha, L. Maglaras, H. Janicke, and L. Shu, "Federated deep learning for cyber security in the internet of things: concepts, applications, and experimental analysis," *IEEE Access*, vol. 9, pp. 138509–138542, 2021, doi: 10.1109/ACCESS.2021.3118642.
- [7] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [8] H. Yu, S. Yang, and S. Zhu, "Parallel restarted SGD with faster convergence and less communication: demystifying why model averaging works for deep learning," *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pp. 5693–5700, 2019, doi: 10.1609/aaai.v33i01.33015693.
- [9] M. A. -Uddin *et al.*, "Federated collaborative filtering for privacy-preserving personalized recommendation system," *arXiv-Computer Science*, pp. 1–12, 2019, doi: 10.48550/arXiv.1901.09888.
- [10] E. M. Campos *et al.*, "Evaluating federated learning for intrusion detection in internet of things: review and challenges," *Computer Networks*, vol. 203, 2022, doi: 10.1016/j.comnet.2021.108661.
- [11] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-IIoTset: a new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning," *IEEE Access*, vol. 10, pp. 40281–40306, 2022, doi: 10.1109/ACCESS.2022.3165809.
- [12] A. Tabassum, A. Erbad, W. Lebda, A. Mohamed, and M. Guizani, "FEDGAN-IDS: privacy-preserving IDS using GAN and federated learning," *Computer Communications*, vol. 192, pp. 299–310, 2022, doi: 10.1016/j.comcom.2022.06.015.
- [13] E. M. Campos *et al.*, "Evaluating federated learning for intrusion detection in internet of things: review and challenges," *Computer Networks*, vol. 203, no. 3, pp. 5693–5700, 2022, doi: 10.1016/j.comnet.2021.108661.
- [14] T. Zhang, C. He, T. Ma, L. Gao, M. Ma, and S. Avestimehr, "Federated learning for internet of things: a federated learning framework for on-device anomaly data detection," *SenSys 2021 - Proceedings of the 2021 19th ACM Conference on Embedded Networked Sensor Systems*, pp. 413–419, 2021, doi: 10.1145/3485730.3493444.
- [15] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated deep learning for zero-day botnet attack detection in IoT-edge devices," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3930–3944, 2022, doi: 10.1109/JIOT.2021.3100755.
- [16] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated-learning-based anomaly detection for IoT security attacks," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, 2022, doi: 10.1109/JIOT.2021.3077803.
- [17] R. Taheri, M. Shojafar, M. Alazab, and R. Tafazolli, "FED-IIoT: a robust federated malware detection architecture in industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. XX, no. Xx, pp. 1–11, 2020, doi: 10.1109/TII.2020.3043458.
- [18] T. Al Nuaimi *et al.*, "A comparative evaluation of intrusion detection systems on the edge-IIoT-2022 dataset," *Intelligent Systems with Applications*, vol. 20, p. 200298, 2023, doi: 10.1016/j.iswa.2023.200298.
- [19] A. Galassi, M. Lippi, and P. Torroni, "Attention in natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4291–4308, 2021, doi: 10.1109/TNNLS.2020.3019893.
- [20] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath, "An attentive survey of attention models," *ACM Transactions on Intelligent Systems and Technology*, vol. 12, no. 5, pp. 1–32, 2021, doi: 10.1145/3465055.
- [21] A. D. S. Correia and E. L. Collobini, "Attention, please! A survey of neural attention models in deep learning," *Artificial Intelligence Review*, vol. 55, no. 8, pp. 6037–6124, 2022, doi: 10.1007/s10462-022-10148-x.
- [22] G. Brauwuers and F. Frasinca, "A general survey on attention mechanisms in deep learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3279–3298, 2023, doi: 10.1109/TKDE.2021.3126456.
- [23] S. Islam *et al.*, "A comprehensive survey on applications of transformers for deep learning tasks," *Expert Systems with Applications*, vol. 241, 2024, doi: 10.1016/j.eswa.2023.122666.
- [24] P. Pobrotyn, T. Bartczak, M. Synowiec, R. Białobrzęski, and J. Bojar, "Context-aware learning to rank with self-attention," *arXiv-Computer Science*, pp. 1–8, 2020, doi: 10.48550/arXiv.2005.10084.
- [25] A. Vaswani *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 2017, pp. 5999–6009, 2017.
- [26] S. Seo *et al.*, "Hunt for unseen intrusion: multi-head self-attention neural detector," *IEEE Access*, vol. 9, pp. 129635–129647, 2021, doi: 10.1109/ACCESS.2021.3113124.
- [27] A. Rush, "The annotated transformer," in *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pp. 52–60, 2019, doi: 10.18653/v1/w18-2509.
- [28] H. Taheri Gorji *et al.*, "Federated learning for clients' data privacy assurance in food service industry," *Applied Sciences*, vol. 13, no. 16, 2023, doi: 10.3390/app13169330.
- [29] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2017, pp. 1273–1282.
- [30] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *8th International Conference on Learning Representations, ICLR 2020*, pp. 1–16, 2020.
- [31] K. Shaikat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu, "A survey on machine learning techniques for cyber security in the last decade," *IEEE Access*, vol. 8, pp. 222310–222354, 2020, doi: 10.1109/ACCESS.2020.3041951.
- [32] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. N Anwar, "TON-IoT telemetry dataset: a new generation dataset of IoT




- and IIoT for data-driven intrusion detection systems,” *IEEE Access*, vol. 8, pp. 165130–165150, 2020, doi: 10.1109/ACCESS.2020.3022862.
- [33] M. M. Rashid, S. U. Khan, F. Eusufzai, M. A. Redwan, S. R. Sabuj, and M. Elsharief, “A federated learning-based approach for improving intrusion detection in industrial internet of things networks,” *Network*, vol. 3, no. 1, 2023, doi: 10.3390/network3010008.
- [34] L. A. C. Ahakonye, C. I. Nwakanma, J. M. Lee, and D.-S. Kim, “FED-MARINE: federated learning framework for DDoS detection and mitigation in maritime-SCADA network”.
- [35] M. P. Singh, A. Anand, L. A. Prateek Janaswamy, S. Sundararajan, and M. Gupta, “Trusted federated learning framework for attack detection in edge industrial internet of things,” in *2023 8th International Conference on Fog and Mobile Edge Computing, FMEC 2023*, 2023, pp. 64–71. doi: 10.1109/FMEC59375.2023.10305910.

BIOGRAPHIES OF AUTHORS






Imad Tareq Al-Halboosi    is currently, an employee of the Dewan AL-Waqf AL-Sonny in Iraq, received a master's degree in computer science—Middle East University, Jordan, in 2016, where he is currently pursuing a Ph.D. degree. His current research interests include cyber-security, network and computer security, the internet of things, and artificial intelligence applications. He can be contacted at email: emadtariq1982@gmail.com.






Bassant Mohamed Elbagoury    received the Ph.D. and M.Sc. degrees in computer science from the Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt, in 2009 and 2005, respectively., She was a Ph.D. researcher with the NAO Robot team Humboldt, Germany. She finished her Ph.D. thesis in only two and half years. Since 2003, she has participated in many international conferences in Paris, Poland, Germany, Jordan, and the USA. Since 2005, she has been a Reviewer in AAAI USA conferences. She is currently an Assistant Professor of computer science with the Faculty of Computer Science and Computer Engineering, King Salman International University, Faculty of Computer and Information Sciences, Ain Shams University. Her research areas are robotics, artificial intelligence, mobile computing, and cloud computing. She can be contacted at email: drbassantcs@gmail.com.



Salsabil Amin El-Regaily    is currently a lecturer in the Faculty of Computer and Information Sciences, Ain Shams University in Cairo Egypt. She got her Ph.D. degree in Computer Science in 2019 and currently works in the Department of Basic Sciences in the faculty. She worked as a teaching assistant and assistant lecturer before the current position. Her areas of interest are structured programming, image processing, and deep learning. She is a member of the Ain Shams University Ranking team and the Deputy Director of the Assessment and Evaluation Unit at the Faculty of Computer and Information Sciences. Her professional activities include being an academic advisor for the credit hours programs, a member of the quality assurance unit, and a member of the Egypt government excellence award team. She has presented many research papers in local and international journals and conferences related to the image processing field, especially medical imaging. Her scopus H-index is 3, with 103 citations to date. In 2021, she received an award from the Association of Arab Universities for the second-best Ph.D. thesis in the Artificial Intelligence field. In 2023, she was awarded the Ain Shams Incentive Award in the advanced medical technological science field. She can be contacted at email: salsabil.amin@gmail.com.



Professor El-Sayed M. El-Horbaty    received his Ph.D. (1985) in Computer Science from London University, UKhis M.Sc. (1978), and B.Sc. (1974) in Mathematics from Ain Shams University, Egypt. His work experience as an academic in Egypt (Ain Shams University), Qatar (Qatar University), and Emirates (Emirates University, Ajman University, and ADU University). His current areas of research are distributed and parallel computing, cloud computing, mobile cloud computing, edge computing, e-health computing, IoT, and optimization of computing algorithms. His work appeared in many journals such as: Parallel Computing, International Journal of High Performance and Grid Computing, International Journal of Information Security, International Journal of Computer Communication and Information Security, and International Review on Computers and Software, Advances in Intelligent Systems and Computing, International Journal of Mobile Network Design and Innovation, and International Journal of Bio-Medical Information and e-Health. He can be contacted at email: sayed.horbaty@yahoo.co.