Article

A Heterogeneity-Aware Semi-Decentralized Model to Lightweight IDS for IoT Networks based on Federated Learning and BiLSTM

Shuroog Saad AlSaleh [*] , Mohamed El Bachir Menai , Saad A. Al-Ahmadi

Posted Date: 25 December 2024

doi: 10.20944/preprints202412.2130.v1

Keywords: Internet of Things; Intrusion Detection System; Anomaly Detection; Machine Learning; Deep Learning; Federated Learning; Energy-based; Centralized FL; Decentralized FL; Semi- Decentralized FL; Transfer Learning.

Article

A Heterogeneity-Aware Semi-Decentralized Model to Lightweight IDS for IoT Networks based on Federated Learning and BiLSTM

Shuroog Alsaleh [1,2,*], Mohamed El Bachir Menai [2] and Saad Al-Ahmadi [2]

[1] Department of Computer Science, King Saud University, Riyadh, Saudi Arabia
[2] Department of Information Technology, Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia
[*] Correspondence: 441204410@student.ksu.edu.sa

Abstract: Internet of Things (IoT) networks' wide range and heterogeneity make them prone to cyberattacks. Most IoT devices have limited resource capabilities (e.g., memory capacity, processing power, and energy consumption) to function as conventional intrusion detection systems (IDSs). Researchers have applied many approaches to lightweight IDSs, including energy-based IDSs, machine learning/deep learning (ML/DL)-based IDSs, and federated learning (FL)-based IDSs. FL has become a promising solution for IDSs in IoT networks because it reduces the overhead in the learning process by engaging IoT devices during the training process. Three FL architectures are used to tackle the IDSs in IoT networks, including centralized (client-server), decentralized (device-to-device), and semi-decentralized. However, none of them has solved the heterogeneity of IoT devices while considering lightweight-ness and performance at the same time. Therefore, we propose a semi-decentralized FL-based model for a lightweight IDS to fit the IoT device capabilities. The proposed model is based on clustering the IoT devices— FL clients—and assigning a cluster head to each cluster that acts on behalf of FL clients. Consequently, the number of IoT devices that communicate with the server is reduced, helping to reduce the communication overhead. Moreover, clustering helps in improving the aggregation process as each cluster sends the average model's weights to the server for aggregation in one FL round. The distributed denial-of-service (DDoS) attack is the main concern in our IDS model, since it easily occurs in IoT devices with limited resource capabilities. The proposed model is configured with two deep learning techniques: BiLSTM and WGAN using the CICIoT2023 dataset. The experimental results show that the BiLSTM achieves better performance and is suitable for resource-constrained IoT devices based on model size. We test the pre-trained semi-decentralized FL-based model on three datasets: BoT-IoT, WUSTL-IIoT-2021, and Edge-IIoTset, and the results show that our model has the highest performance in most classes, particularly for DDoS attacks.

Keywords: Internet of Things; intrusion detection system; anomaly detection; machine learning; deep learning; federated learning; energy-based; centralized FL; decentralized FL; semi-decentralized FL; transfer learning

---

1. Introduction

With the number of IoT devices significantly increasing, the need to preserve the security and privacy of transmitted data has become a challenging task. Many mechanisms have been used to secure IoT networks, for example, firewalls and access-control mechanisms, but these methods only ensure the confidentiality and authenticity of data within the network of IoT devices. Thus, IoT networks need security mechanisms that can serve as a line of defense for detecting intruders. An intrusion detection system (IDS) needs to be applied to detect malicious behaviors. Two common approaches for Intrusion Detection Systems (IDSs) are Signature-Based Intrusion Detection Systems (SIDSs) and Anomaly-Based Intrusion Detection Systems (AIDSs). An SIDS, that is, "heuristics-based or misuse detection," defines a set of known malicious data patterns (signatures) or attack rules (heuristics) that are used for attack detection. However, this approach can only identify known attacks for which it has stored patterns or rules for comparison. On the other hand, an AIDS, that is, "behavior-based IDS," entails developing the profiles of normal users' behaviors over a period of time so that any deviation from this profile will be viewed as an anomaly or abnormal behavior [1].

Most IoT devices have limited resource capabilities to carry out conventional IDSs. Therefore, lightweight IDS approaches have been proposed to fit IoT device capabilities. First, energy-based IDSs are based on detecting attacks by using energy consumption as an indicator of specific types of attacks. However, not all attack types can be detected using this method because many attacks have no impact on energy consumption (e.g., Sybil or data manipulation attacks) [2,3]. Second, machine learning and deep learning (ML/DL) models are used to analyze data and their dependencies to help recognize attack patterns [4]. Moreover, ML/DL-based IDSs are used as lightweight IDSs by extracting the most relevant features for attack detection. However, with the number of interconnected IoT devices increasing dramatically and generating a massive amount of data, ML/DL models are difficult to deploy within resource-constrained IoT networks [5,6].

One ML model, known as federated learning (FL), is a distributed ML model that enables IoT devices to learn a shared ML model collaboratively without disclosing local data. Thus, the introduction of FL in IDSs helps decrease the computational burden of central processing servers while preserving data privacy. Three FL architectures are commonly used: centralized FL (client-server); decentralized FL (Device-to-Device); and semi-decentralized FL. In the centralized FL architecture, each device trains the local model on its own data, and then, its parameters/weights are aggregated to produce a global model controlled by a central entity (i.e., a server or coordinator) [7,8]. Even though centralized FL improves the efficiency of the learning process, it has some drawbacks, namely a lack of scalability and connectivity and a single-point-of-failure vulnerability. Decentralized FL overcomes these issues by allowing devices to collaborate in the learning process without a coordinator. However, the aggregation process is inefficient because it is not done under the supervision of one device, resulting in low accuracy [9–11]. Semi-decentralized FL strikes a balance by clustering devices, with each cluster having a local coordinator to facilitate learning and aggregation, resulting in faster convergence and improved scalability [12–14].

In all FL architectures, heterogeneity is a core challenge that hinders FL performance [15] because heterogeneity greatly impacts the learning process, resulting in model divergence [16–18]. The resource-constrained nature of IoT devices demands lightweight models. Furthermore, most current approaches fail to achieve a balance between high performance and efficient resource utilization.

In this paper, we present a semi-decentralized FL model to detect intrusions in a heterogeneous IoT network. The primary contributions of our work are outlined below:

- The proposed semi-decentralized FL model clusters FL clients to eliminate the impact of heterogeneity in the FL learning process and reduce resource consumption.
- The clustering approach in the proposed model helps in improving the performance of FedAvg, the aggregation algorithm, as each cluster sends the average model's weights to the server for aggregation in one FL round.
- We investigated two DL techniques, BiLSTM and WGAN, as local models in the semi-decentralized FL architecture. To the best of our knowledge, we are the first to apply WGAN with one generator for all FL clients to train them on the same generated data since our target is to detect intrusion, not to know the real from the generated data. Each FL client has a discriminator model to train it on its local data.
- The semi-decentralized FL model with BiLSTM strikes a balance between performance and lightweight-ness to fit the IoT capabilities as it is configured with low complex parameters (number of layers and neurons).

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 provides a detailed explanation of the proposed solution. Section 4 presents the experimental evaluation and discusses the results along with a comparative study. Finally, Section 5 concludes the study by summarizing the findings and highlighting open research directions.

2. Related Work

As the IoT devices has limited resource capabilities, there is a need to consider this limitation when designing IDSs for IoT. Many approaches have been proposed to develop lightweight IDSs, namely energy-based IDSs, ML/DL-based IDSs, and federated learning–based IDSs.

2.1. Lightweight Energy-Based IDSs

Since conventional IDSs are not applicable for resource-constrained IoT devices, many studies have focused on minimizing energy consumption through lightweight IDSs. The energy-based approach is emphasized when analyzing energy consumption to identify attacks by estimating expected energy consumption based on past observations for each device. Thus, if a given device's energy use deviates from the expected value, it is viewed as an anomaly. Unfortunately, this approach cannot detect all types of attacks, only those that impact energy consumption, for example, various types of DoS attacks (flooding, blackholes, etc.) [2,3] .

2.2. Lightweight Machine Learning/Deep Learning–Based IDSs

Machine learning and deep learning (ML/DL) techniques are commonly used in IDSs to analyze data and their dependencies as a way to learn attack patterns [2]. In ML, feature engineering is performed to extract the relevant features for attack identification and classification. Many ML algorithms are used in IDS, for example, decision tree (DS), support vector machine (SVM), logistic regression (LR), naïve Bayes (NB), and artificial neural network (ANN). However, DL algorithms can automatically extract high-level features through multiple non-linear processing layers. The most commonly used DL models in IDSs include convolutional neural network (CNN), recurrent neural network (RNN), long–short-term memory (LSTM), and autoencoder (AE) [5,6]. To create lightweight IDSs for the IoT environment, ML/DL-based IDSs are used for feature selection and dimensionality reduction [19–21]. However, because of the significant growth in interconnected IoT devices, which generate huge amounts of data, ML/DL-based IDSs have become difficult to deploy in resource-constrained IoT networks.

2.3. Lightweight Federated Learning–Based IDSs

FL is a privacy-preserving distributed ML model that enables FL clients to learn a shared ML model collaboratively without disclosing local data. The application of FL for IoT networks has been used recently, as the ML model is distributed among FL clients, which helps in preserving the computation resources. Three FL architectures are commonly used: centralized FL (client-server), decentralized FL (device-to-device), and semi-decentralized FL.

2.3.1. Centralized FL (Client-Server)

In this architecture, each device independently trains its local models using its own data. The parameters or weights from these local models are then aggregated to form a global model, which is managed by a central entity, such as a server or coordinator, as shown in Figure 1. The training process involves several rounds. In each round, clients send their weights to the server, which updates the global model. This process continues until the desired accuracy is achieved or a predetermined number of rounds is completed [7,8].

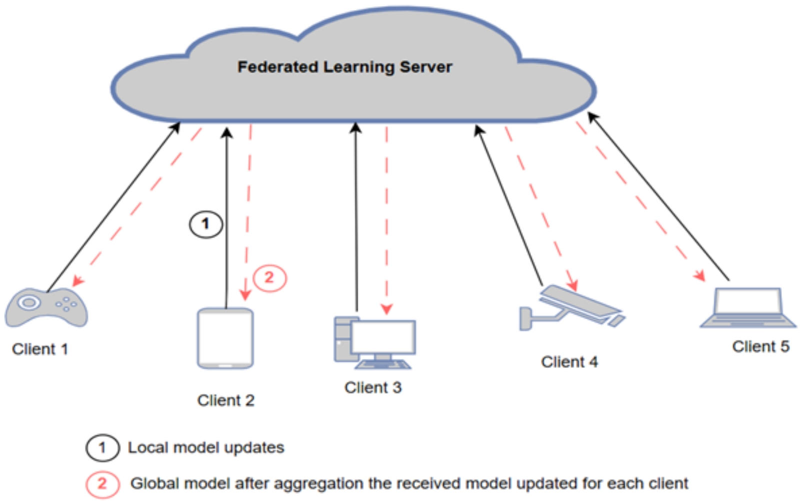

Federated Learning Server
1
2
Client 1
Client 2
Client 3
Client 4
Client 5
1 Local model updates
2 Global model after aggregation the received model updated for each client

Figure 1. *Centralized FL (Client-Server) Architecture*

The deployment of centralized FL for IoT networks comes in two schemes. First, the edge-cloud architecture, in which a cloud server functions as an FL server and an edge device functions as an FL client. Second, the edge layer architecture, in which the edge device functions as the FL server and the IoT device (end node) functions as the FL client. Many studies have been proposed based on cloud-edge architecture. Huong et al. [22] developed a low-complexity cyberattack detection in IoT edge computing (LocKedge), in which the cloud server extracts the most relevant features using the principal components analysis (PCA) method, and the NN model's hyper-parameters and initial weights can be set. This information is then sent to all edge devices to train their models with their own data using SGD. Once all edge devices finish the training process, they send the updated weights of their model to the server for the aggregation process. LocKedge is evaluated in two modes: traditional ML and FL, and the results showed that traditional ML has a higher detection rate than FL because of uneven data distribution—non-IID (nonindependent and identically distributed among edge devices). Rashid et al. [23] developed an FL-based model for cyberattack detection in IIoT networks. They applied CNN and RNN for both traditional ML and FL. According to their results, the FL model with a few number of rounds and the traditional ML model differ considerably (both CNN and RNN). However, as the number of FL rounds increased, the margin shrinks, and at the 50th round, the FL-based model reaches the same intrusion detection accuracy as traditional ML. Similar work is presented in [24] but with distributing datasets in a different ratio among clients. Their results show that the variation in dataset distributions would be the cause of the performance gap between FL and traditional ML. This is because the non-IID data could cause local updates from different clients to clash, which results in performance degradation of the global model. Zhang et al. [25] proposed a federated learning framework known as FedDetect for IoT cybersecurity. The proposed model has improved the performance by utilizing a local adaptive optimizer and a cross-round learning rate scheduler instead of FedAvg for local training. According to the evaluation results of two settings—traditional ML and FL—FL's detection accuracy is worse than traditional ML because, with the former, the server could not directly learn the data's features, as with traditional ML, making FL worse for feature learning. However, the system efficiency analysis indicates that both end-to-end training time and memory costs are affordable and promising for resource-constrained IoT devices. Another FL-based IDS model was developed by [26] using a deep autoencoder to detect botnet attacks using on-device decentralized traffic data. They installed a virtual machine on each edge device to conduct the local model training, and they used port mirroring on the edge devices so that network traffic flow toward the IoT devices is not interrupted. However, using a virtual machine comes with many issues, such as complexity, less efficiency, and high cost. An ensemble, multiview, FL-based model was proposed by [27], which categorizes packet information into three groups: bidirectional features; unidirectional features; and packet features. Each view is trained using NN, and then their predicted results are sent to the rain forest (RF) classifier, which acts as an ensembler that combines the three views' predictions and provides a single attack prediction based on its probability and occurrence. Similarly, Driss et al. [28] also proposed FL-based attack detection that uses RF to ensemble the global ML models—GRU with different window sizes—to detect attacks in vehicular sensor networks (VSN). According to the experimental results of both studies [27,28],the use of an ensembler unit increases the attack prediction accuracy. Friha et al. [29] applied FL in an agricultural IoT network using three DL models—DNN, CNN, and RNN— on three different datasets. Their results show that one of the datasets—namely InSDN—outperformed the traditional ML model. The results also demonstrate that time and energy consumption have been affected by the DL model and number of clients involved in the FL process.

The communication burden is one of the biggest issues in centralized FL as each client sends their model weights/parameters to the server and then receives the new weights/parameters (after aggregation) from the server. Thus, a client selection is proposed to reduce the number of participating clients in the learning process. An FL-based model deployed in the edge layer, in which the edge device is the FL server and the IoT device is the FL client, is proposed by [30,31]. Considering that IoT devices vary in their resource capabilities, one that satisfies the requirements of ML/DL algorithms must be selected. Chen et al. [30] developed an asynchronous FL model to improve training efficiency for heterogeneous IoT devices. The selection algorithm, which is based on the node's computing resources and its communication condition. Thus, instead of waiting for all nodes to send their weight vectors to be aggregated by the server, only the selected clients can send their updates, which accelerates the learning process in its convergence. Rjoub et al.[31] developed a client selection method based on two metrics—resource availability (in terms of energy level) and IoT devices' trustworthiness—to make appropriate scheduling decisions. They measure the IoT device's trust level (trust score) based on utilizing the resources (over/under a specific threshold value) during the local training. Even though the client selection mechanism helps in improving the learning process, it may affect the prediction accuracy, where not all clients are involved in the learning process. On the other hand, Chen et al. [32] proposed a communication-efficient federated learning (CEFL), which assures communication efficiency and resource optimization that differ from [30,31]. Specifically, a client selection method is proposed based on the calculation of the gradient difference between the current model parameter and the model parameter from the previous round. Then, the client sends its new local model only if the gradient difference is large; otherwise, the server will reuse the stale copy of that client. Even though the CEFL algorithm has succeeded in reducing the communication burden, it adds additional computational overhead for calculating the gradient difference.

Transfer learning (TL) is combined with FL to improve learning performance. Nguyen et al. [20] introduced a new compression approach—known as high-compression federated learning (HCFL)—to handle FL processes in a massive IoT network. HCFL applies TL to enhance learning performance by training a pre-model with a small amount of the dataset on the server. Abosata et al. [34] also incorporated TL with FL to develop accurate IDS models with minimum learning time. The main goal of the TL model is to transfer the local parameters generated using gradient-based weights from the clients to the server. Their experimental results show that their model succeeded in minimizing the complexity of the learning model and maximizing the accuracy of global model generation. Nguyen et al. [35] were the first researchers to employ FL in anomaly-based IDSs; they developed a device-type-specific model in which IoT devices are mapped to the corresponding device type for anomaly detection because IoT devices that belong to the same manufacturer have similar hardware and software configurations, resulting in highly identical communication behavior. Wang et al. [36] applied the same device-type-specific model to develop an IDS model for each type of IoT device, which improves the model's accuracy. It also addresses IoT device heterogeneity in terms of communication protocols and coexisting technologies. However, this approach cannot be applied in large-scale IoT networks where there are various types of IoT that need to be mapped, which will result in delay. Tabassum et al. [37] developed the first FL-based IDS utilizing GAN to train local data, addressing class imbalance by generating synthetic data to balance the distribution across FL clients. They evaluated their model using different datasets, with the results showing that the proposed model converged earlier than the state-of-the-art standalone IDS. Bukhari et al. [38] proposed an FL-based IDS using a stacked CNN and bidirectional long–short-term memory (SCNN-Bi-LSTM). They addressed the statistical heterogeneity of the data by applying a personalized method that could divide the model into global and local layers. Specifically, the global layers are used in the FL, and each FL client is provided with the global layers with unique customization layers designed for their specific needs. The experimental results show a significant improvement in detection accuracy compared with traditional DL approaches.

2.3.2. Decentralized FL (Device-to-Device)

The decentralized FL architecture relies on device-to-device (D2D) communication, where devices collaboratively train their local models using stochastic gradient descent (SGD) and consensus-based methods. During each consensus step, devices share their local model updates with their one-hop neighbors. Each device then integrates the model updates received from its neighbors and inputs the results into the SGD process[9,10]. In 6G networks, devices are anticipated to be interconnected in a machine-to-machine (M2M) manner, making D2D communication a promising technology for decentralized FL [7,8]. Figure 2 shows the decentralized FL architecture.

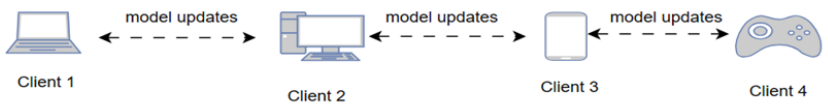


Figure 2. *Decentralized FL (D2D) architecture*

Although the centralized FL has demonstrated success in the IoT environment in terms of preserving computation resources as well as improving the learning process, it includes some drawbacks, that is, single-point-of-failure and scaling issues for increasing network size. This leads to the FL being fully distributed (server-less), in which the end devices collaborate to perform data operations inside the network by iterating local computations and mutual interactions. Savazzi et al. [9] proposed a decentralized FL-based consensus approach for IoT networks that can enable direct device-to-device (D2D) collaboration without relying on a central coordinator. This approach has been viewed as a promising framework for IoT networks because of its flexible model optimization over networks characterized by decentralized connectivity patterns. However, they applied their methods to a simple NN, which would be difficult to apply to deeper networks. A trusted decentralized FL-based algorithm was integrated with the consensus approach in [10]. Their results indicate that the trusted decentralized FL algorithm makes the model robust against poisoning attacks. A blockchain federated learning (B-FL) based on a consensus approach was proposed in [11] to prevent model tampering from malicious attacks in both local and global models. The B-FL system comprises multiple edge servers and devices that generate a blockchain based on a consensus protocol to confirm that the data in the block are correct and immutable. They applied digital signatures to guarantee that others were not tampering with the data packet information. Although digital signatures ensure the authenticity and integrity of the data, they require computation overheads that are beyond the IoT devices' capabilities.

Al-Abiad et al. [39] proposed a decentralized FL model based on device-to-device (D2D) communications and overlapped clustering to allow aggregation in a decentralized scheme without a central aggregation. The proposed model is based on the overlap between clusters. Specifically, within a cluster, the cluster head (CH) is associated with each cluster. On the other hand, the bridge device (BD) is located between two adjacent clusters. The BDs receive the updated models from the CHs and calculate the summation of the received aggregated models. As a result, CHs collect all the local models from the corresponding devices and models from the BDs to update their cluster models. Their experimental results show that the proposed model outperformed the centralized and hierarchical FL models in terms of energy consumption and accuracy, here considering efficient overlapped clustering and increasing the number of iterations, respectively.

2.3.3. Semi-decentralized FL

The semi-decentralized FL architecture aims to strike a balance between fully centralized and fully decentralized FL architectures. This approach employs multiple servers, each responsible for coordinating a group of client devices to perform local model updates and inner model aggregation. Periodically, each server shares its updated models with neighboring servers for outer model aggregation. This multi-server aggregation process, as opposed to relying on a single centralized entity, helps to speed up the learning process [12–14]. Figure 3 illustrates the semi-decentralized FL architecture.

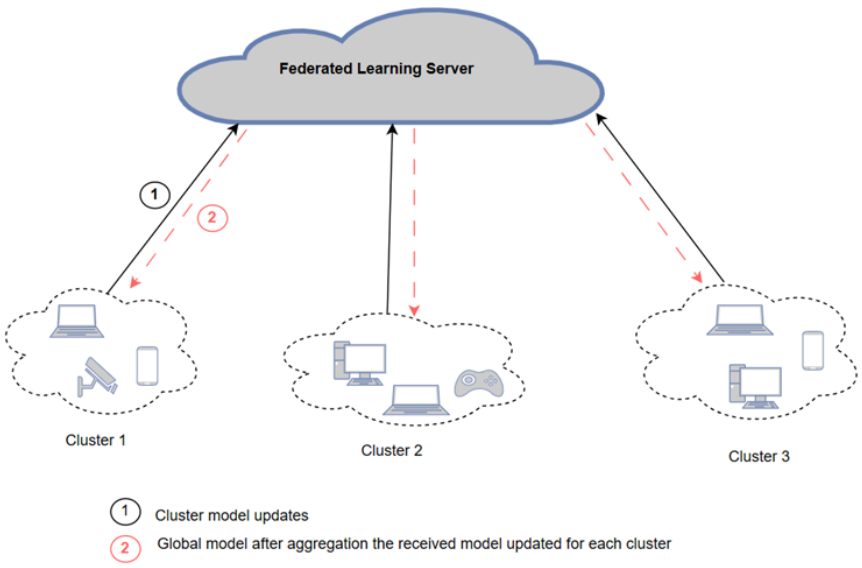

Federated Learning Server
1
2
Cluster 1
Cluster 2
Cluster 3
1 Cluster model updates
2 Global model after aggregation the received model updated for each cluster

Figure 3. *Semi-decentralized FL architecture*

A semi-decentralized approach was developed by [12] to accelerate the learning process by considering multiple edge servers. Each edge server coordinates a cluster of client nodes to perform local model updating and intracluster model aggregation. The edge servers periodically share their updated models with neighboring edge servers for intercluster model aggregation. Even though their model accelerates the learning process, it poses computation overhead on the edge server because it needs to perform aggregation for the cluster and for the neighboring servers. To overcome this problem, a hierarchical architecture was presented in [13,14] with two levels of model aggregation: the edge and cloud levels. More specifically, the edge node is responsible for local aggregation of clients, whereas the cloud facilitates global aggregation because it has a reliable connection. Both studies' experimental results demonstrate a reduction in the model training time and energy consumption of the end devices when compared to traditional cloud-based FL. The author of [40] introduced hierarchical FL based on semi-synchronous communications to solve the heterogeneity of IoT devices. The proposed model performs edge-based aggregations and then cloud-based aggregation, resulting in a significant communication overhead reduction. This is because edge-based aggregation is done via local communications, which requires fewer resources than cloud-based aggregation. Moreover, a semi-synchronous communication in both the cloud and edge layers is proposed to prevent the deadlock that comes as a result of waiting for local updates from dropped devices. Thus, a timeout is defined such that if all local models are not received by this timeout, then the edge server sends the latest aggregated edge model to the cloud server without waiting for the remaining local models to arrive. Huang and Zhang[62] proposed a semi-decentralized cloud-edge-device hierarchical federated learning framework, incorporating an incremental sub-gradient optimization algorithm within each ring cluster to mitigate the effects of data heterogeneity. This hierarchical FL architecture not only reduces communication overhead, but also enhances overall performance.

Alotaibi and Barnawi [41] developed IDSoft, an innovative software-based solution that operates throughout the network infrastructure, using 6G technologies such as virtualization of network functions, mobile edge computing and software-defined networking. This solution is designed to support FL-based IDSs. Additionally, they designed a hierarchical federated learning (HFL) model within IDSoft to detect cyberattacks in 6G networks. Synchronous and asynchronous schemes are performed in the aggregation process. The results show that asynchronous HFL has faster convergence and stable training loss compared with the centralized FL model. In addition, the HFL model achieved lower communication overhead than the centralized FL model because the centralized FL architecture requires communication between each client and the server for aggregation; however, in HFL, clustering helps in reducing the total communication load by allowing the CH to act on the behalf of the clients. However, they did not mention other hierarchical FL-based IDSs, which may indicate that they were the first to introduce this model. The same authors developed LightFIDS (Lightweight and Hierarchical Federated IDS for Massive IoT in 6G Networks) [61], a module within the IDSoft architecture they previously introduced. In the IDSoft, there are two detection phases: a fast detection phase that identifies abnormal flows at the far edge, followed by a deep detection phase that provides further analysis of malicious flows. The proposed LightFID model is implemented with fast-converging HFL. The local learning models used in the federated architecture are low-complex models with fewer parameters. However, the hierarchical federated learning architecture is more complex, requiring the cluster head to perform aggregation for multiple rounds in order to obtain the cluster model. The head master (server) also aggregates the received cluster models to get the global model. This process is computationally intensive due to the repeated aggregation and distribution within clusters.

2.4. Motivation

From all the above, FL with the three architectures has succeeded in designing IDS for IoT networks, but with some drawbacks. In centralized FL, communication latency is the main issue due to the time required to collect the trained model from clients, especially considering the heterogeneity of IoT devices. Another significant issue is communication overhead, as all clients must communicate directly with a single server, leading to increased network load and energy consumption [10,30]. The decentralized FL relies on the D2D transmission, which has a low network cost and serves as a practical solution for faster convergence. This, in turn, reduces delay and energy consumption. Additionally, decentralized FL resolves the single-point-of-failure and scalability issues inherent in centralized FL. However, it has a major drawback: the aggregation process is inefficient, just as it is in the centralized FL model, because aggregation is not performed under the supervision of one device, resulting in an accuracy drop-off [9–11].

Semi-decentralized FL solves the issues of centralized and decentralized FL architectures. Specifically, the semi-decentralized FL model is based on clustering the FL clients and then assigning a Cluster Head (CH) for each cluster that communicates with the FL server. Therefore, clustering will help in reducing the total communication load by allowing the CH to act on behalf of the FL clients, which helps in preserving the resource consumption (bandwidth) [41]. Moreover, the clustering can also be used to solve data heterogeneity, which is considered one of the biggest issues of deploying FL in IoT networks [14,25]. Clustering clients based on received model updates after training the FL clients was proposed by [43–45]. According to their theoretical findings, the cosine similarity between different clients' weight updates is highly indicative of their data distributions' similarities. After clustering the clients, each cluster trains its model independently until the global model for each cluster converges. However, most proposed semi-decentralized FL approaches primarily focus on addressing data heterogeneity and minimizing communication overhead, without considering the lightweight structure of the model, specifically its size (i.e., the number of parameters).

3. Proposed Solution

This section presents the proposed IDS model for IoT networks. It begins with the description of how FL clients are generated, considering having all types of attacks. Following that, it presents how the semi-decentralized FL is applied after clustering the FL clients. Next, it provides an overview of

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2024 doi:10.20944/preprints202412.2130.v1

the deep learning techniques, namely BiLSTM and WGAN, that serve as local models within the FL architecture.

3.1. Generating FL Clients

First of all, the dataset will be pre-processed and split into training, validation, and testing sets, as will be explained in the experimental evaluation in Section 4. The training set is then distributed across the FL clients, considering that each client should have all types of attack as well as benign data. Since the dataset's classes are imbalanced, it needs to consider this during the dataset distribution by taking a ratio from each class based on the class size (number of instances for each type of class). The way of distributing the dataset is done randomly; thus, we generate five different clients' distributions, known as "runs" . Algorithm 1 presents the method to generate FL clients.

```
Algorithm 1: CreateClient
Input: $TrainSet$: Preprocessed Training set ,
       $NumberOfClients$: Number of federated members (clients) to be generated
Output: $Clients$: array of FL Clients
1  Shuffle $TrainSet$
2  $Class$ = Labels of $TrainSet$
3  $NumberOfClasses$= Length($Class$)
4  $ClassSize$ = length($Class$) // to hold each class size
5  $ClassRatios$ =[ ] // to generate random class distribution for each client
6  $StartIndex$=[ ] $\leftarrow$ 0
7  $EndIndex$ =[ ]
8  for each client i in $NumberOfClients$ -1 do
9      for each j in Class do
10         $ClassRatios$ = rational random number
11         $EndIndex$[j] = $StartIndex$[j] + $ClassSize$[j] $\times$ $ClassRatios$[j]
12         $Client_i$ = $Class_j$ from $StartIndex$[j] to $EndIndex$[j]
13         $StartIndex$ = $EndIndex$
   // for last client
14 if $i = NumberOfClients$ then
15     for each j in Class do
16         $Client_i$= $Class[j]$ from $StartIndex[j]$ to $ClassSize[j]$
   $Clients$ + = $Client_i$
   return $Clients$
```

3.2. Proposed Semi-decentralized FL Model

The proposed semi-decentralized Federated Learning (FL) model clusters clients based on the model updates received from FL clients. Specifically, each client is trained using a simple Multi-Layer Perceptron (MLP) for one round as a pre-processing step before clustering. Next, the updated model weights are collected from each client to prepare for clustering. Since the model weights are high-dimensional, dimensionality reduction is applied before clustering the clients to avoid sparsity issues in the clusters. Alkhayrat et al. [48] conducted a comparative study between Principal Component Analysis (PCA) and Autoencoder (AE) for dimensionality reduction before clustering. Their results indicate that AE outperforms PCA in their case study. Therefore, AE is applied to the received model weights to reduce their dimensions. The reduced weights are then scaled before applying the clustering algorithm [46].

After evaluating various clustering algorithms, k-means with the Manhattan distance metric is selected to cluster the clients. A cluster head (CH) is chosen for each cluster based on the averaged Silhouette score. The Silhouette score is used because it indicates whether a client is correctly positioned within its assigned cluster and not too close to the boundary of neighboring clusters. Algorithm 2 summarizes the clustering methodology that we developed.

Algorithm 2: Clustering

```
Input: $Clients$ : Array of FL clients
Output: $Clusters$: Lists of clustered FL clients
1 $weights$=[ ] // To hold the clients' model weights
2 Create $ClientModel$
3 for each $Client_i$ in $Clients$ do
4     Train $CleintModel$[i] using $Client_i$
5     $weights$[i] = Get weights from $ClientModel$[i]
  // Reduce the dimensionality of weights-space using Autoencoder
6 $ReducedWeights \leftarrow$ Autoencoder ($weights$)
  // Apply Elbow method to find optimal number of clusters
7 $NumberOfClusters \leftarrow$ Elbow($ReducedWeights$)
  // Apply k-means clustering
8 $Clusters \leftarrow$ Kmeans($ReducedWeights$, $NumberOfClusters$, DistanceMetric= $Manhattan$)
9 return $Clusters$
```

The semi-centralized FL model starts after clustering the clients and assigning the CH for each cluster. The server broadcasts the model with initial weights and parameters to the CH, then the CH sends them to all clients within its cluster. Each client then trains its local model and sends the updated model to its corresponding CH. Once the CH receives the updated model from all clients, it calculates the average weights and sends them to the server. After the server gets all new weights from each cluster, it aggregates the received model weights from all CH and calculates the global model using FedAvg, as shown in equation 1. Then, the server broadcasts the new model to the cluster heads to update their local model. This process is repeated for a given number of rounds or until the model converges, where no significant improvement in performance is observed. To evaluate performance during training, the global model generated after each round is validated using the validation set, and performance metrics are calculated. Algorithms 3, 4, and 5 summarize the semi-decentralized model, demonstrating the training procedure on the server side, cluster side, and client side, respectively. After completing the FL training, the global model is tested on the testing set to evaluate its performance. Algorithm 6 presents the FL testing process.

3.3. Aggregation Algorithm

Once all clients have trained their data, each cluster head collects the new parameters/weights from its clients and calculates the average weights and sends it to the server for aggregation. The server then calculates the aggregated weights by averaging the received weights/parameters using the FedAvg algorithm [53] as shown in the following formula:

$$w_{\text{global}} = \sum_{k=1}^{K} \frac{n_k}{n} w_k \tag{1}$$

where:

- w_{global} is the global model's weights.
- K is the number of clusters.
- n_k is the total number of data samples in each cluster k.
- n is the total number of data samples in all clusters, $n = \sum_{k=1}^{K} n_k$.
- w_k are the weights of the model trained in the cluster k.

Algorithm 3: Semi-Decentralized FL Server Training

```
Input: $W_0$ : Initial model weights
       $NumberOfRounds$: number of FL rounds
       $NumberOfEpochs$: number of local epochs
       $Clients$: Array of FL Clients
       $ValidateSet$: Preprocessed Validation Set
Output: $W_G$: Global Model Weights
1  $W_G \leftarrow W_0$
2  $Clusters$ = Clustering($Clients$)
3  $NumberOfClusters$ = Length($Clusters$)
4  $Scores$ =[ ]
5  $AvgScore$=[ ] $\leftarrow 0$
6  Create $GlobalModel$
7  Create $ClusterModel$
8  $ClusterHeads$[$NumberOfClusters$]
9  $ClusterWeights$[ $NumberofClusters$]
10 for each $Cluster_j$ in $Clusters$ do
11     for each $Client_i$ in $Cluster_j$ do
12         $Scores$= SilhouetteScores($Client_i$)
13     $AvgScore$= Average ($Scores$)
       // Set the ClusterHead to the Client that has closed score to averaged
          score within its cluster
14     if $Scores[i]$ Close to $AvgScore$ then
15         $ClusterHeads$[j]= $Client_i$
16     for each $R$ in $NumberOfRounds$ do
           // Broadcast global weights and number of epochs to all clusters
17         for each $Cluster_j$ in $Clusters$ do
18             Send ($ClusterModel$,$W_G$, $NumberOfEpochs$) to $ClusterHeads$
               // Receive updated model from ClusterHead
19             Receive ($W_j$) from $ClusterHeads$
20             $ClusterWeights$ += ($W_j$)
           // Aggregate received model weights
21         $ScaleWeights$= ScaleValue($ClusterWeigts$)
22         $W_G$= $ScaleWeights$ / $NumberOfClusters$
23     Set $GlobalModel$ weights to $W_G$
       // Evaluated the updated $GlobalModel$ on each FL round using $ValidateSet$
24     $PerformanceMetric$ += FLModelTesting($GlobalModel$, $ValidateSet$) // Exit from the
          loop once there is no change in the performance (model convergence)
25     if $PerformanceMetric[R]$ - $PerformanceMetric[R-1]$ =0.00001 then
26         break
27 return $W_G$
```

Algorithm 4: Semi-decentralized FL Cluster Training

```
Input: $W_G$: Global Model Weights,
$NumberOfEpochs$: Number of Local Epochs
Output: $W_j$: Cluster Model Weights
1 Receive $ClusterModel$ from the server
2 Receive initial weights and parameters from the server
3 Send $ClusterModel$, weights, and parameters to the clients
4 $W_i$ += Get model weights from clients
  // Average the received client weights within the same cluster
5 $W_j \leftarrow$ Average($W_i$)
6 return $W_j$
```

Algorithm 5: Semi-decentralized FL Client Training

```
Input: $W_G$: Global Model Weights,
$NumberOfEpochs$: Number of Local Epochs
Output: $W_i$: Client Model Weights
1 Receive $ClientModel$ from the $ClusterHead$
2 Receive weights and parameters from the $ClusterHead$ ($W_G$, $NumberOfEpochs$)
3 Train each $ClientModel$ with received weights and parameters
4 $W_i \leftarrow$ Get weights from $ClientModel$
  // Return updated client weights $W_i$ to the $ClusterHead$ for aggregation
5 return $W_i$
```

Algorithm 6: FL Model Testing

```
Input: $GlobalModel$ : Global Model after FL process
$TestSet$: Preprocessed testing set
Output: $DetectionRate$: metric that used to measure model performance
1 $XTest$= Features of $TestSet$
2 $YTest$= Labels of $TestSet$
3 $PredictedY$ = Test $GlobalModel$ on $XTest$
4 Calculate $PerformanceMetric$ based on $PredictedY$ and $YTest$
5 return $PerformanceMetric$
```

The equation 1 shows that the global model weights are a weighted sum of the cluster's model weights, with the weights proportional to the number of data samples in each cluster. The FedAvg is used at the server level, considering that each cluster sends the average weights of its clients to the server. Unlike other semi-decentralized approaches [61] where the aggregation process is performed in two levels: the cluster head and the server. The cluster head conducts the aggregation process for multiple rounds to obtain a cluster model.Then, the server aggregates the received cluster models to get the global model and sends it back to the cluster heads for the next FL round. This process is computationally intensive due to the repeated aggregation and distribution within clusters. Our proposed model reduces this complexity by allowing each cluster head to collect weights from its clients, average them once, and send the averaged weights directly to the server. The server then aggregates the received cluster's models and returns the global model to the cluster heads, all within one FL round, thereby reducing the communication overhead and making it affordable for IoT resource-constrained.

3.4. Local Deep Learning Models

3.4.1. Bidirectional Long Short-Term Memory (BiLSTM)

BiLSTM is an extension of the LSTM model where two LSTM models are applied to the input data. Unlike the standard LSTM network, which only utilizes the information it has already encountered in the sequence, the BiLSTM architecture incorporates two LSTM layers—one processing the input sequence forwards (forward LSTM) and the other processing it backward (backward LSTM), as shown in Figure 4. This dual application of the LSTM significantly enhances the model's capabilities to learn long-term dependencies, ultimately leading to improved performance [38].

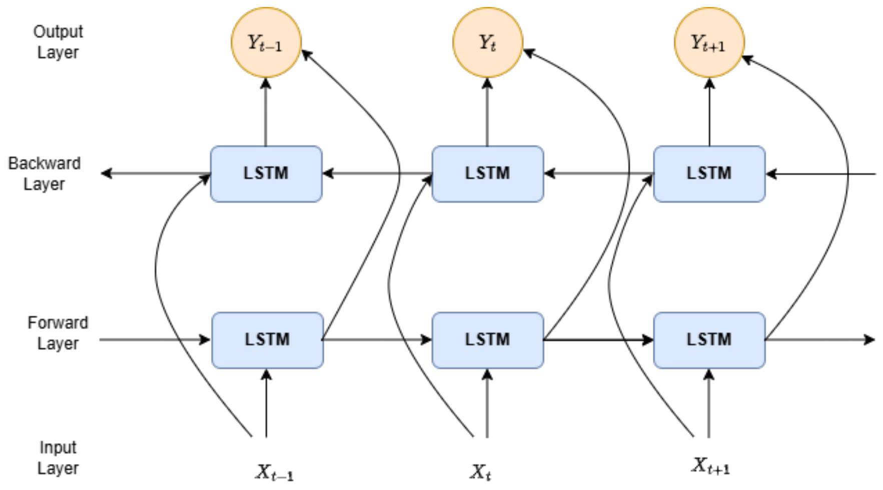


Figure 4. *Illustration of Bidirectional LSTM [38]*

3.4.2. Wasserstein Generative Adversarial Network (WGAN)

Wasserstein GAN (WGAN) is an alternative to the traditional GAN model. In GAN, there are two neural networks—the generator and the discriminator—that engage in a dynamic adversarial process. The generator aims to capture the data distribution and then produce synthetic data that looks like the real data. On the other hand, the discriminator evaluates incoming samples and predicts whether they originate from the real dataset or were generated synthetically. This competition drives both networks to improve, leading to the generation of increasingly realistic data [66].

WGAN improves the training stability of traditional GAN by replacing commonly used divergence metrics, which may not be continuous concerning the generator's parameters, with the Earth-Mover (Wasserstein-1) distance. This distance measures the minimum cost of transforming one probability distribution into another by considering both the amount of mass moved and the transport distance. Under mild assumptions, the Wasserstein distance is continuous and differentiable almost everywhere, ensuring more stable training [67,68].

The utilization of GAN in federated learning is done by having one generator for all FL clients to make them trained on the same generated data, as our goal is intrusion detection, not to know the real from generated data. Each FL client has a local discriminator model that is trained on its local data. The updated model is then sent to the global discriminator on the server for aggregation, which generates the global model. The results of training GAN in a semi-decentralized FL approach are not efficient (the generator loss is very high); thus, WGAN is used instead. This inefficiency arises because the generator struggles to learn from FL clients with different data distributions. Therefore, we apply WGAN, where the generator uses the Wasserstein loss, which measures the Wasserstein distance between the real and generated data distributions [67]. Moreover, the discriminator uses gradient penalty to improve the stability of the training process as well as training the discriminator multiple times, which is known as "critic," for each generator update [68]. Therefore, by improving gradient flow, reducing mode collapse, and enhancing training stability, WGAN has become a powerful alternative to GAN.

4. Experimental Evaluation

This section shows the experimental details of the proposed model. All experiments were conducted on Google Colab with NVIDIA Tesla T4 GPU and 12 GB RAM. The proposed semi-decentralized FL based model with both BiLSTM and WGAN was implemented using Python 3.10.12 programming language with TensorFlow 2.17.0, Keras 3.4.1, and scikit-learn 1.3.2. The clustering is implemented using PyClustering, which is an open-source library [56].

4.1. Datasets

4.1.1. Training Dataset

The dataset used is CICIoT2023, which is created by the Canadian Institute for Cybersecurity (CIC) [42]. The CICIoT2023 dataset was generated using a topology that includes 105 IoT devices. Among these, 67 devices actively participated in the attacks, and five hubs connected an additional 38 Zigbee and Z-Wave devices. This setup replicates a real-world scenario of IoT products and services within a smart home environment. The devices used in this topology include smart home devices, cameras, sensors, and micro-controllers, all of which were interconnected and configured to facilitate various attacks and to capture the resulting attack traffic. Thus, the CICIoT2023 dataset is heterogeneous as it has a diversity of device types and communication protocols.

The CICIoT2023 consists of 33 types of attacks that are divided into seven categories: Distributed Denial-of-Service (DDoS), Denial-of-Service (DoS), Recon, Web-based, Brute Force, Spoofing, and Mirai. The benign data is gathered from IoT traffic in idle states where the devices are configured with default parameters and without malicious or attacking scripts. The CICIoT2023 consists of 46,217,820 instances, each representing a network flow with 47 features. Figure 5 shows the number of instances for each attack category. We chose the CICIoT2023 dataset because it includes all classes in both the training and testing sets, unlike other datasets. Additionally, the number of instances per class is large enough to be distributed across the FL clients, allowing each client to be trained on each class. This, in turn, contributes to the development of an accurate global model. To the best of our knowledge, we are the first who use the whole CICIoT2023 dataset, considering all types of attacks, as other studies work with part of the dataset or consider specific types of attack [64,69,70].

First of all, the dataset needs to be pre-processed and divided into training, validation, and testing sets with proportions of 60%, 20%, and 20%, respectively. The features are normalized using the StandardScaler method, while the classes (labels) are encoded using One-Hot Encoding for multi-class classification (34/8 classes) and Binary Encoding for binary classification.

4.1.2. Testing Dataset

Three datasets are used for testing the proposed pre-trained model, namely BoT-IoT[54] , WUSTL-IIOT-2021 [55], and Edge-IIoTset[63]. Our goal was to test the proposed model's generalizability on additional datasets and improve its performance through fine-tuning. The reason for choosing those datasets is because they have similar features to the training dataset. Moreover, the DoS and DDoS attacks are presented in all the datasets, which are the main attacks that our model is targeting. More importantly, all datasets are designed for IoT traffic.

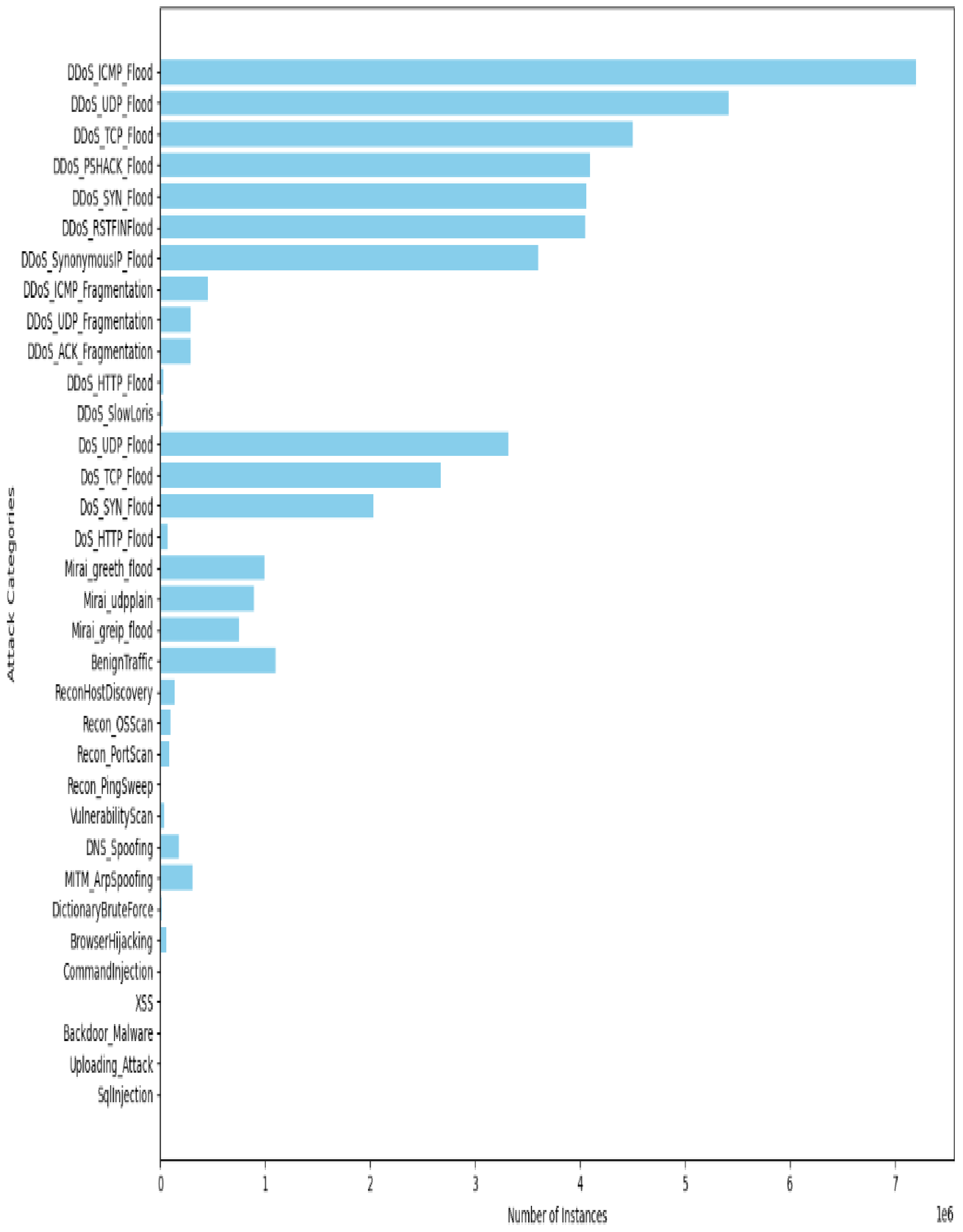


Figure 5. CICIoT2023 Class Distribution

BoT-IoT dataset was created by designing a realistic IoT network environment with five distinct IoT scenarios: a weather station, a smart fridge, remotely activated lights, motion-activated lights, and a smart thermostat. We utilized a 5% version extracted from the original dataset that has 35 features. The dataset includes 10 types of attacks: DDoS (HTTP, TCP, UDP), DoS (HTTP, TCP, UDP), OS Fingerprinting, Server Scanning, Keylogging, and Data exfiltration attacks, as shown in Table 1. The 70% of the data is used for training, and the remaining 30% is used for testing. The original model had 47 input features, whereas the BoT-IoT datasets had 35 features. Thus, we modified the input layer of the model to accommodate this difference.

Table 1. Types of Attack and Number of Samples in BoT-IoT dataset

Types of Attack	Number of samples
DoS-HTTP	1485
DoS-TCP	615800
DoS-UDP	1032975
DDoS-HTTP	989
DDoS-TCP	977380
DDoS-UDP	948255
OS Fingerprinting	17914
Server Scanning	73168
Keylogging	73
Data Theft	6
Normal	477
Totals	**3668522**

WUSTL-IIOT-2021 dataset includes network traffic from Industrial Internet of Things (IIoT) systems that were used in cybersecurity research. The dataset has 41 features after removing four features, as the publisher of the dataset mentioned that "they are unique to the attacks and would expose the type of the attack to the model; therefore, the model would not be generalized for unseen data" [55]. In addition, the unused columns that are dropped which are: 'StartTime', 'LastTime', 'SrcAddr', 'DstAddr', 'sIpId', 'dIpId'. Table 2 presents the dataset information. The dataset includes four types of different attacks: DoS, Command Injection, Reconnaissance, and Backdoor attacks. As DoS attacks typically generate high volumes of traffic and a large number of samples, 90% of the attack data are allocated to represent them. Other attack types occur less frequently, and when they do, they transmit only a limited amount of traffic data. Table 3 illustrates the statistics of the dataset. The 80% is used for training, and the remaining 20% is used for testing.

Table 2. WUSTL-IIOT-2021 dataset Specification

Number of Samples	1,194,464
Number of features	41
Number of attack samples	87,016
Number of normal samples	1,107,448

Table 3. Statistical information of the traffic types in WUSTL-IIOT-2021

Traffic's type	Percentage (%)
Normal Traffic	92.72
Total Attack Traffic	7.28
Command Injection Traffic	0.31
DoS Traffic	89.98
Reconnaissance Traffic	9.46
Backdoor Traffic	0.25

Edge-IIoTset dataset [63] was generated using a testbed that accurately simulates a real-world IoT/IIoT environment. The dataset includes real-world data, collected by executing realistic cyberattacks and capturing both legitimate and malicious network traffic. The dataset has 61 features and 14 types of attacks, which are divided into five categories: DDoS, Injection, MITM, Malware, and Scanning attacks. Table 4 illustrates the class distributions in Edge-IIoTset. The dataset consists of several files, including attack traffic, normal traffic, and selected datasets for ML and DL, which contain two CSV files: DNN-EdgeIIoT-dataset.csv and ML-EdgeIIoT-dataset.csv. We use DNN-EdgeIIoT-dataset.csv to evaluate our model. Seventy percent of the data is used for training, while the remaining 30% is reserved for testing. We apply fine-tuning to all datasets for testing our pre-trained model, as the input features differ from the pre-trained model and the number of classes varies. During fine-tuning, all layers are frozen except the input and output layers to retain the learned representations while allowing the model to adjust to the new feature space. This process significantly improves the model's performance on the unseen datasets, demonstrating the efficacy of fine-tuning in cross-dataset generalization.

Table 4. Types of Attack and Number of Samples in Edge-IIoTset

Attack's Category	Attacks	No. of Instances
DoS/DDoS Attacks	DDoS_HTTP attack	229022
	DDoS_ICMP attack	2914354
	DDoS_TCP attack	2020120
	DDoS_UDP attack	3201626
Injection Attacks	SQL_injection attack	51203
	Uploading attack	37634
	XSS attack	15915
Malware Attacks	Backdoor attack	24862
	Password attack	1053385
	Ransomware attack	10925
Scanning attacks	Port_Scanning attack	22564
	Vulnerability_scanner attack	145869
	Fingerprinting attack	1001
MITM attack		1229
Normal		11223940
Total		20952648

4.2. Performance Metrics

In this study, several performance evaluation indicators are employed to assess the effectiveness of the model. The following metrics are typically used in analyzing the intrusion detection performance:

- *True Positive (TP)*: reports the number of attack samples that are correctly classified as attacks.
- *False Positive (FP)*: reports the number of benign samples that are wrongly classified as attacks.
- *True Negative (TN)*: reports the number of benign samples that are correctly classified as benign.
- *False Negative (FN)*: reports the number of attack samples that are wrongly classified as benign.
- *Accuracy*: This metric indicates the proportion of correctly classified instances out of the total number of examples. It is calculated using:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{2}$$

- *Precision*: Precision measures the ratio of true positive predictions to the total number of positive predictions (both true and false). It is determined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

- *Recall*: Also known as sensitivity, recall quantifies the number of true positive predictions made out of all actual positives. It is given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

- *F1-Score*: This metric provides a balance between precision and recall by calculating their harmonic mean. The F1-score is computed using:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

For multi-classifications, the macro-averaging is used to calculate the recall, precision, and F1-score across all classes. As the dataset is imbalanced, the macro-averaging ensures that each class contributes equally to the evaluation by independently calculating and averaging the metrics for each class.Consequently, the overall performance measure is not biased toward any particular class. All performance metrics were computed using the scikit-learn (sklearn) library in Python. This library provides standardized implementations of these metrics, ensuring consistency in model evaluation.

4.3. Parameter Tuning

As BiLSTM has higher performance metrics, which will be shown in the performance evaluation section in Section 4.4, it is chosen as the local model for our proposed semi-decentralized FL model. The performance of any deep neural network is greatly affected by its hyper-parameters. Specifically, the BiLSTM model is influenced by the learning rate, batch size, dropout rate, activation function, number of layers, and number of neurons per layer. Thus, in this section, those hyper-parameters are tuned to make the model reach the best results. Since our model is targeting IoT environments with resource-constrained capabilities, the number of layers and the number of neurons per layer are tuned to provide a balance between performance and lightweight-ness, where the lightweight-ness is measured by the model size. Each parameter configuration is applied on each run, and the results are averaged, and the standard deviation is calculated as presented in Tables 5 and 6. After conducting a paired t-test to compare the performance metrics across different model parameters (m1, m2, m3, and m4), the results indicate no statistically significant difference in performance between models m1, m2, and m3. However, a significant difference was observed between m2 and m4, with m2 demonstrating superior performance. Given our primary objective of achieving high performance while minimizing model complexity, we selected m2. This configuration, with only two layers, provides a lightweight structure suitable for resource-constrained IoT devices while maintaining competitive performance.

Table 5. Average Performance measures for each parameter configuration in BiLSTM

Parameters	Avg. Accuracy	Avg. Recall	Avg. Precision	Avg. F1-Score
m1: 1 layer (64 neurons)	0.99	0.6741	0.7567	0.6935
m2: 2 layers (128, 64)	**0.9909**	**0.6805**	0.7948	**0.7045**
m3: 3 layers (128, 64, 32)	0.9896	0.6692	0.8143	0.6916
m4: 4 layers (128, 64, 64, 32)	0.9859	0.6664	0.7849	0.6861

Table 6. Standard Deviation of Performance measures for each parameter configuration in BiLSTM

Parameters	St. Dev Accuracy	St. Dev Recall	St. Dev Precision	St. Dev F1-Score
m1: 1 layer (64 neurons)	0.0004	0.0026	0.0098	0.0021
m2: 2 layers (128, 64)	0.0001	0.0025	0.0064	0.0028
m3: 3 layers (128, 64, 32)	0.0014	0.0011	0.0196	0.0025
m4: 4 layers (128, 64, 64, 32)	0.0079	0.0017	0.0178	0.0034

After the parameter tuning, the BiLSTM model is configured with two layers (128 and 64 neurons), knowing that the input layer has the same number of features of CICIoT2023 which is 46, and the output layer has the same number of classes as there are: 34, 8, and binary classes. The other hyper-parameters were configured after extensive experiments as follows: “ReLU” was used as the activation function in the hidden layers, with “Softmax” applied to the output layer for both the 34-class and 8-class models. However, for binary classification, a “Sigmoid” function was used in the output layer. To prevent overfitting, a dropout layer with a rate of 20% was added after each hidden layer, randomly ignoring 20% of neurons during training. The loss function used was categorical cross-entropy for multi-class classification and binary cross-entropy for binary classification. All experiments were conducted for 1 epoch with a batch size of 64, a learning rate of 0.001, and the optimizer used was “adam”.

For the federated learning (FL) settings, the number of participating clients is set to 10, with the training set divided among them, ensuring that each client has samples from all types of attacks, as shown in Algorithm 2. Since the distribution is done randomly, we generate five different client distributions or "runs". All experiments are performed on these five runs, and the results are averaged to minimize the impact of randomness.

We set the number of FL rounds to 10 and use FedAvg as the aggregation algorithm. For the semi-decentralized FL-based model, clustering is performed using k-means with the Manhattan as the distance measure. The number of clusters is set to 3 ($k = 3$) based on the Elbow method.

4.4. Performance Evaluation

As it is mentioned earlier, there are five different client distributions known as runs. The FL process is applied to all the runs, the results are averaged, and the standard deviation is calculated. The model is applied to the 34-class, 8-class, and binary classification tasks. Two deep learning (DL) models are used: BiLSTM and WGAN, as described in Section 3.4.
BiLSTM is used as a local model to train the local data in each FL client. The results of training all five runs are averaged, and the standard deviation is calculated as shown in Table 7.

Table 7. Performance Metrics (Average and Standard Deviation) for 34-class, 8-class, and Binary Classes for Semi-decentralized FL-based Model (BiLSTM)

Class Type	Accuracy	Recall	Precision	F1-score
34 Classes (AVG)	0.9855	0.6602	0.7227	0.6731
34 Classes (St.Dev)	0.0001	0.0028	0.0019	0.0016
8 Classes (AVG)	0.9909	0.6805	0.7948	0.7045
8 Classes (St.Dev)	0.0001	0.0025	0.0064	0.0028
Binary Classes (AVG)	0.9943	0.9474	0.9307	0.9389
Binary Classes (St.Dev)	0.0001	0.0033	0.0021	0.0008

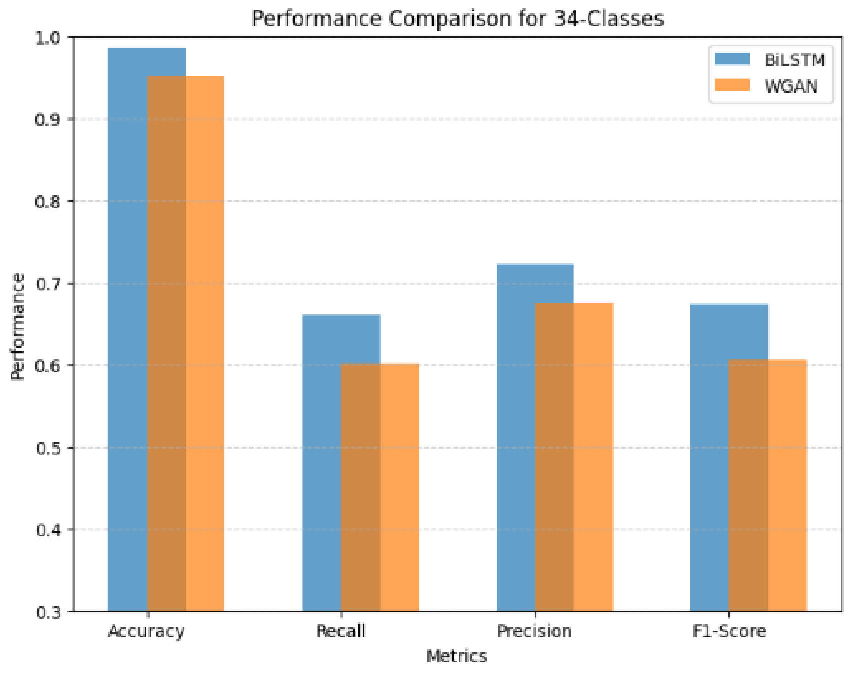
The WGAN is used as a local model to train the FL client. However, the training of the WGAN is different in the FL framework. Since the WGAN has two models, a generator and a discriminator, the FL process is done by having one generator for all FL clients to make them trained on the same generated data, as our goal is to detect intrusion, not to know the real from generated data. Each FL client has a discriminator model to train it on its local data. In WGAN, the generator uses the Wasserstein loss, which measures the Wasserstein distance between the real and generated data distributions [67]. Moreover, the discriminator uses a gradient penalty to improve training stability and is trained multiple times (five critic updates in our case) for each generator update [68]. The results of semi-decentralized FL using WGAN after training the five runs are averaged, and the standard deviation is calculated, as presented in Table 8.

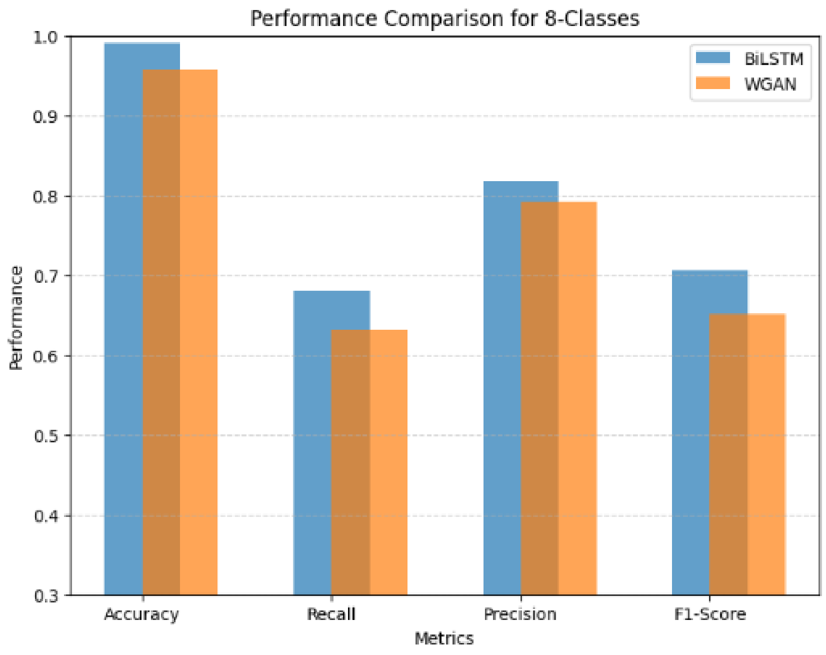
Table 8. Performance Metrics (average and standard deviation) for 34-class, 8-class, and binary classes for Semi-decentralized FL-based Model (WGAN)

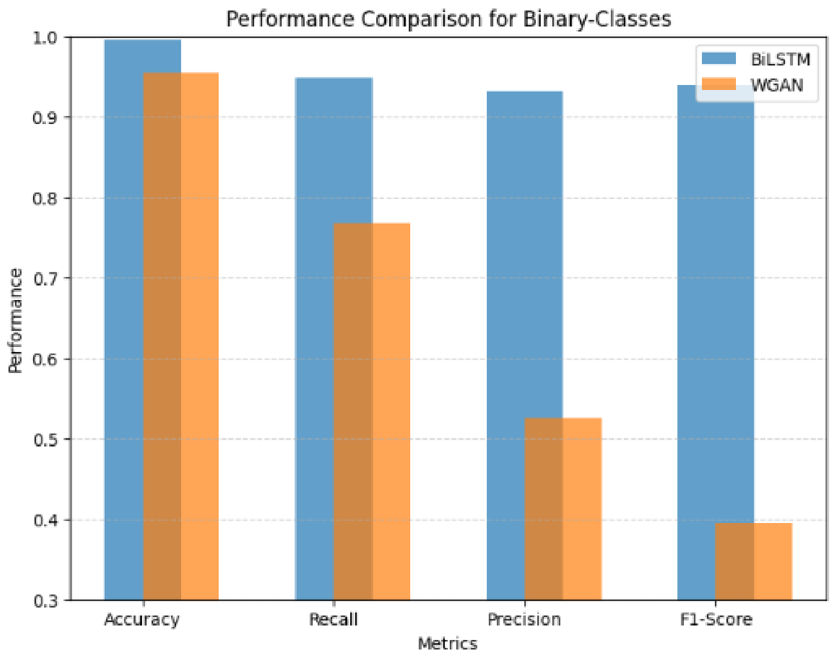
	Accuracy	Recall	Precision	F1-score
34 Classes AVG	0.9516	0.6095	0.6474	0.6099
34 Classes St.Dev	0.0061	0.0064	0.0070	0.0053
8 Classes AVG	0.9569	0.6309	0.7927	0.6511
8 Classes St.Dev	0.0235	0.0168	0.0313	0.0131
Binary classes AVG	0.5464	0.7673	0.5246	0.3957
Binary classes St.Dev	0.00635	0.00324	0.00033	0.00334

According to Figure 6, the results indicate that the BiLSTM achieves higher performance measures than WGAN across all class categories. Additionally, when we apply a paired t-test to determine the statistical significance of the difference, the results show a statistically significant difference in performance between the BiLSTM and WGAN. Therefore, we choose the BiLSTM as the local model in our proposed semi-decentralized FL architecture.

Figure 6. Comparison of Semi-Decentralized FL Models (BiLSTM, WGAN) Across Different Class Categories.







4.5. *Performance Comparisons*

The pre-trained semi-decentralized FL-based model with BiLSTM is tested on three datasets: BoT-IoT, WUSTL-IIoT-2021, and Edge-IIoTset. As it is mentioned previously, all datasets have different input features as well as numbers of classes. Thus, fine-tuning is applied on the pre-trained model where all layers are frozen except the input and output layers to retain the learned representations while allowing the model to adjust to the new feature space. This process significantly improved the model's performance on the unseen datasets, demonstrating the efficacy of fine-tuning in cross-dataset generalization. As there are five runs for the proposed semi-decentralized FL model, there will be five pre-trained models that can be used for testing an unseen dataset. For each model, we calculate the average F1-Score, as it is the harmonic mean of precision and recall, and choose the model that has the closest value to this average. Since there are three classifications per model (34-class, 8-class, and binary), one model is chosen for each classification based on the average F1-Score.

4.5.1. BoT-IoT Testing Results

The number of input features in BoT-IoT is 35; however, the pre-trained model has 46 features, as well as the number of classes being different. Therefore, the input and output layers are adjusted to match the number of features and classes of the BoT-IoT dataset, while the hidden layers are frozen. The semi-decentralized FL-based model with BiLSTM (34 classes, as it includes the different types of DoS and DDoS attacks as in BoT-IoT), as a local model, is tested on the BoT-IoT test set (which is 30% of the dataset). Then, we compare our results with the results obtained from [22] who proposed a multi-attack detection mechanism called LocKedge (Low-Complexity Cyberattack Detection in IoT Edge Computing) that uses the BoT-IoT dataset. The proposed model, Lockedge, relies on a centralized FL approach. Table 9 demonstrates the detection rate (recall) for each type of attack.

Table 9. Detection Rate (Recall) Comparison Across Different Attack Types Using Various Models

Attack Type	KNN	DT	RF	SVM	LocKedge	Proposed Semi-Decentralized-FL BiLSTM
DoS-HTTP	0.81690	0.84507	0.76056	0.74647	0.90862	0.6879
DoS-TCP	**1.0000**	0.99752	**1.0000**	**1.0000**	**1.0000**	0.9647
DoS-UDP	0.99851	0.99926	0.99926	0.99554	0.99928	0.7292
DDoS-HTTP	0.96774	0.82258	0.96774	0.97581	0.98715	**0.9998**
DDoS-TCP	0.99173	0.97746	0.99248	0.99624	0.99965	**1.0000**
DDoS-UDP	0.99217	**1.0000**	**1.0000**	0.96784	**1.0000**	**1.0000**
OS Fingerprinting	0.93478	0.93478	0.89130	0.78261	0.99258	**1.0000**
Server Scanning	0.97826	**1.0000**	**1.0000**	0.98913	0.99973	**1.0000**
Keylogging	**1.0000**	0.30000	0.90000	**1.0000**	0.99268	0.0000
Data Theft	**1.0000**	0.00000	0.00000	0.00000	0.56098	0.0000

Compared with the Lockedge model, the semi-decentralized FL-based model improves the detection rate in classes, while in some classes the detection rate is not improved. Our proposed model achieves the highest detection rate for DDoS attacks, OS fingerprint, and Server scanning attacks. However, for the Keylogging and Data Theft attack, the detection rate is 0, which comes as a result of having a low number of samples compared with other attack categories. As presented in Table 1, there is a significant imbalance in the number of samples for each class, making the distribution of these samples among FL clients a challenging task. Even when reducing the number of clients, the client still needs enough samples per class to train its local model, which in turn will aid in generating an accurate global model.

4.5.2. WUSTL-IIoT-2021 Testing Results

The WUSTL-IIoT-2021 dataset has 41 features and five classes: DoS, Command Injection, Reconnaissance, Backdoor attacks, and Normal. As the number of input features and number of classes vary from the pre-trained model, fine-tuning is applied to the input and output layers. Then, our proposed model with 8 classes is tested on the testing set (which is 20% of the dataset). Eid et al. [65] proposed an IDS for IIoT that utilized WUSTL-IIoT-2021. The proposed IDS is based on an optimized CNN model, and it examines the impact of artificially balancing the training datasets on the CNN model's performance. To facilitate this analysis, two types of training subsets were prepared: one that retained the natural imbalance of the datasets and another that was deliberately balanced to equalize the class distributions. A comparison of our testing results with the [65] results is demonstrated in Tables 10, 11, and 12 below. We compare our results with the unbalanced since we did not apply any oversampling technique and kept the dataset unbalanced.

Table 10. Precision Comparison of Testing WUSTL-IIoT-2021 for Multi-Classification

Attack scenario	Unbalanced (CNN)	Proposed Semi-Decentralized-FL BiLSTM
Backdoor	0.7685	**0.9952**
Command injection	0.9731	**0.9991**
Denial of service	0.9990	**1.0000**
Reconnaissance	0.9971	**0.9997**
Normal	**0.9995**	0.7188

Table 11. Recall Comparison of Testing WUSTL-IIoT-2021 for Multi-Classification

Attack scenario	Unbalanced (CNN)	Proposed Semi-Decentralized-FL BiLSTM
Backdoor	0.7830	**0.9979**
Command injection	0.8378	**0.9966**
Denial of service	**0.9939**	0.7045
Reconnaissance	0.9999	**0.9999**
Normal	**0.9999**	0.6053

Table 12. F1-Score Comparison of Testing WUSTL-IIoT-2021 for Multi-Classification

Attack scenario	Unbalanced (CNN)	Proposed Semi-Decentralized-FL BiLSTM
Backdoor	0.7757	**0.9966**
Command injection	0.9004	**0.9978**
Denial of service	**0.9965**	0.8267
Reconnaissance	0.9985	**0.9998**
Normal	**0.9997**	0.6571

According to the results of testing WUSTL-IIoT-2021, we can see that our proposed semi-decentralized FL with BiLSTM model achieves the highest performance measures in most types of attacks. Table 13 presents the average performance measures, demonstrating that the semi-decentralized FL model achieves the highest accuracy as well as shorter testing time due to its lightweight nature.

Table 13. Comparison of WUSTL-IIoT-2021 Testing Results

Metric	Unbalanced CNN	Proposed Semi-Decentralized-FL BiLSTM
Accuracy	0.9994	**0.9996**
Precision	**0.9474**	0.9425
Recall	**0.9229**	0.8608
F1-score	**0.9341**	0.8956
Training time (s)	264	-
Testing time (s)	89	**12.0795**

DeepIIoT is proposed by [57], which utilizes WUSTL-IIoT-2021 to train MLP to detect anomalies. They tackle the issue of class imbalance by randomly undersampling the benign class, which reduces the percentage of benign instances from 92% to 66%. This helps to improve the F1-score where the minority class is being falsely classified as the majority. We tested our proposed model in binary classification to compare it with the DeepIIoT model, as well as the model proposed in [65] . The results are presented in Table 14, which shows that our model (semi-decentralized FL with BiLSTM) achieves the highest results in accuracy and precision, knowing that the classes are kept imbalanced, resulting in lower results in other metrics.

Table 14. WUSTL-IIoT-2021 Binary Testing Results

Model	Accuracy	Precision	Recall	F1-score
DeepIIoT	0.9994	0.9992	**0.9995**	**0.9994**
Optimized CNN	0.9982	0.9469	0.9284	0.9377
Semi-decentralized FL (BiLSTM)	**0.9996**	**0.9995**	0.9979	0.9987

4.5.3. Edge-IIoTset Testing Results

Friha et al. [63] evaluated the Edge-IIoTset in both classical ML/DL (non-federated learning) and federated learning. They use DNN-EdgeIIoT-dataset.csv and split the data into 70% for training and the remaining 30% for testing. There are 93 features and three different classifications: 15, 6, and binary classifications. Therefore, the input and output layers are adjusted to match the number of features and classes of the Edge-IIoTset dataset while the hidden layers are frozen. We use the same test split to evaluate our semi-decentralized FL-based model with 34 classes because it has DDoS attack types as in Edge-IIoTset. Table 15 illustrates a comparison of our evaluation results for each type of attack with the classical ML/DL (non-federated learning).

The attack types in Table 15 are: Backdoor attack (Back), HTTP flood DDoS attack (HTTP), ICMP flood DDoS attack (ICMP), TCP SYN Flood DDoS attack (TCP), UDP flood DDoS attack (UDP), OS Fingerprinting attack (Fing), Man in the middle attack (MITM), Password cracking attack (Pwd), Port Scanning attack (Port), Ransomware attack (Rans), SQL Injection (SQL), Upload attack (Upload), Vulnerability scanning attack (Scan) and Cross-site Scripting attack (XSS).

Table 15. Performance comparison of different ML/DL algorithms (non-federated) across various attack types (Pr :Precision, Re: Recall F1: F1-Score, Metr: metrics)

Alg	**Metr**	**Normal**	**Back**	**HTTP**	**ICMP**	**TCP**	**UDP**	**Fing**	**MITM**	**Pwd**	**Port**	**Rans**	**SQL**	**Upload**	**Scan**	**XSS**
DT	Pr	**1.00**	0.86	0.44	**1.00**	0.66	**1.00**	0.00	0.00	0.00	0.71	0.80	0.34	**1.00**	**1.00**	0.15
	Rc	**1.00**	0.78	0.63	**1.00**	**1.00**	**1.00**	0.00	**1.00**	0.00	0.48	0.86	0.96	0.02	**1.00**	0.23
	f1	**1.00**	0.82	0.52	**1.00**	0.80	**1.00**	0.00	0.00	0.00	0.58	0.83	0.50	0.04	**1.00**	0.18
RF	Pr	**1.00**	0.99	0.64	0.96	0.82	**1.00**	0.77	**1.00**	0.41	0.63	0.96	0.76	0.66	**1.00**	0.65
	Rc	**1.00**	0.92	0.82	**1.00**	0.75	**1.00**	0.10	**1.00**	0.81	**1.00**	0.93	0.21	0.51	0.81	0.58
	f1	**1.00**	0.95	0.72	0.98	0.77	**1.00**	0.18	**1.00**	0.77	0.77	0.95	0.33	0.58	0.90	0.61
Knn	Pr	**1.00**	0.96	0.69	**1.00**	0.76	**1.00**	0.79	0.97	0.45	0.74	0.95	0.74	0.63	**1.00**	0.49
	Rc	**1.00**	0.94	0.74	**1.00**	0.76	**1.00**	0.10	**1.00**	0.56	0.73	0.94	0.49	0.55	0.88	0.57
	F1	**1.00**	0.95	0.72	**1.00**	0.78	**1.00**	0.74	0.99	0.50	0.73	0.95	0.55	0.55	0.90	0.53
SVM	Pr	**1.00**	0.63	0.86	**1.00**	0.74	**1.00**	0.80	**1.00**	0.60	0.64	0.69	0.42	0.66	0.95	0.61
	Rc	**1.00**	0.77	0.60	**1.00**	0.59	**1.00**	0.66	**1.00**	0.23	**1.00**	0.51	0.82	0.40	0.86	0.88
	f1	**1.00**	0.69	0.71	0.99	0.71	1.00	0.72	**1.00**	0.34	0.78	0.59	0.55	0.90	0.86	0.72
DNN	Pr	**1.00**	0.95	0.76	**1.00**	0.82	**1.00**	0.61	**1.00**	0.45	0.66	0.79	0.57	0.56	0.90	0.43
	Rc	**1.00**	0.86	0.92	**1.00**	0.90	**1.00**	0.64	**1.00**	0.38	0.50	0.85	0.71	0.48	0.85	0.37
	F1	**1.00**	0.90	0.83	**1.00**	0.90	**1.00**	0.61	**1.00**	0.45	0.66	0.79	0.57	0.56	0.90	0.43
Semi-dec FL(BiLSTM)	Pr	0.9907	0.9491	**1.00**	**1.00**	**1.00**	0.9407	0.4710	0.9462	**0.9986**	**0.9982**	0.6774	0.8565	0.4488	**1.00**	0.8344
	Rc	0.9802	0.5914	**1.00**	0.9986	**1.00**	0.8305	0.7653	0.8456	**1.00**	0.9590	0.5394	0.9965	0.8875	0.5932	0.1901
	F1	0.9854	0.7288	**1.00**	0.9993	**1.00**	0.8822	0.5831	0.8931	**0.9993**	**0.9782**	0.6006	0.9212	0.5962	0.7446	0.3097

doi:10.20944/preprints202412.2130.v1

According to the results, the semi-decentralized FL model with BiLSTM has higher performance measures in some types of attacks. Specifically, the proposed model achieves the highest performance metrics in detecting HTTP flood DDoS, TCP SYN Flood DDoS attack, Password cracking attack, and SQL Injection attacks. It also has the highest precision and F1-score in detecting Port scanning attack.

The centralized FL-based model that was introduced by [63] is evaluated using the Edge-IIoTset. Thus, we compare our test results with their results considering the same FL parameters: 10 FL rounds, 10 FL clients, and the non-IID dataset. Their model attained an accuracy of 91.45%, which is lower than the accuracy of our proposed model that achieves 95.17%.

Based on the results discussed above, we conclude that our proposed model outperforms classical ML/DL (non-federated) algorithms in certain types of attacks, particularly DDoS attacks.

4.6. Discussion

This study proposed a semi-decentralized federated learning (FL) framework that leverages client clustering to address the heterogeneity and limited resource capabilities of IoT networks. The heterogeneity of IoT devices affects the performance of FedAvg, the aggregation algorithm used in this work, which requires having similar data patterns to perform better. Thus, clustering helps to improve the aggregation process, which, in turn, improves the performance of the global model. Moreover, clustering reduces the communication overhead as the server communicates with the cluster heads instead of each client. Specifically, the cluster head collects the weights from each client within the cluster and sends them to the server. Consequently, the semi-decentralized FL approach has demonstrated improvements in performance and resource consumption, while also addressing the data heterogeneity issue. In other clustered-based FL models, the cluster head conducts multiple rounds of aggregation to obtain the cluster model. Then, the server aggregates the received cluster models to get the global model.

This two levels of aggregation is time and resource consuming. However, in our proposed model, the cluster head aggregates the clients' model updates within the cluster and sends the aggregated model to the server. The server then aggregates the cluster's models and returns the global model to the cluster heads, all within one FL round, thereby reducing the communication overhead and making it affordable for IoT resource-constrained.

The proposed model is configured with two DL techniques as local models, namely BiLSTM and WGAN, using CICIoT2023 dataset. The WGAN is applied in the FL framework with one generator for all FL clients to train them on the same generated data. This is because our concern is to detect intrusion, not to know the real from the generated data. Then, each FL client has a local discriminator model to train it on its local data, sends the updated model to the global discriminator in the server for aggregation, and generates the global model. However, in other FL models that use GAN or WGAN, each FL client has a local generator/discriminator, and the server has a global generator/discriminator. This is because their main concern is to use the local generator to balance the classes in the local dataset in each client [37,71]. The experimental results demonstrate that the BiLSTM achieves better performance results. This is due to the heterogeneity issue inherent in FL that causes WGAN to struggle in generating meaningful data distributions that effectively represent all attack classes. Consequently, discriminators trained on these generated data exhibit reduced accuracy in detecting attacks.

Our primary objective is to design a lightweight IDS that fits the capabilities of IoT networks. To achieve this, we configure the BiLSTM, with a number of layers and neurons that strike a balance between performance and lightweight-ness, with the model size serving as a gauge of lightweight-ness. After conducting extensive experiments, we found that the BiLSTM with two layers, each containing 128 and 64 neurons, yields the best results in terms of both performance and lightweight-ness.

Furthermore, we evaluate our proposed semi-decentralized FL-based model using an IoT dataset, while other semi-decentralized approaches rely on image datasets like CIFAR, MNIST, etc. Most of the proposed IDSs for IoT networks have utilized datasets that do not have IoT traces, such as NSL-KDD, UNSW-NB15, and CICDDoS2019 datasets[65]. In our experiment, the CICIoT2023 is used with large IoT traffic, whereas many papers use part of the dataset to reduce the computation overhead

[69,70]. Nevertheless, we cannot compare with their work as their models are trained on different data samples.

For the other FL approaches (both centralized and decentralized) that are being evaluated using IoT datasets, they did not consider distributing the dataset between clients, considering that each client has all types of attacks. Most of them distribute the dataset based on the IP address, as it is a source of attacks [22,58]. However, after investigating those datasets, it was found that the IP address of the attack does not contain all types of attacks. Thus, some FL clients would not be trained on these attacks, which results in low detection rates for those attacks.

Comprehensively, all the results demonstrate that our proposed semi-decentralized FL effectively produces a lightweight and heterogeneity-aware model for detecting intrusions in IoT networks. Its effectiveness is particularly notable in detecting DDoS attacks, highlighting its potential for real-world cybersecurity applications.

5. Conclusion and Future Work

In this paper, we have introduced a semi-decentralized FL-based model which is a lightweight and heterogeneity-aware intrusion detection mechanism that is suitable for deployment in IoT networks. Our proposed model is based on clustering FL clients, which solves the data heterogeneity, which, in turn, improves the learning process as well as reduces the communication overhead by working at the cluster level instead of the client level.

The proposed model is evaluated using CICIoT2023 dataset as it has a large number of IoT traces and attack categories, particularly DDoS, as it is our target attack. Two DL techniques are being evaluated as local models in a semi-decentralized FL approach, namely BiLSTM and WGAN. The results show that the BiLSTM achieves the best performance; thus, it is chosen as the local model in the proposed model. After extended experiments, the BiLSTM with two layers, each with 128 and 64 neurons, respectively, provides the best results in terms of performance, and lightweight-ness makes it affordable for IoT resource-constrained. The pre-trained semi-decentralized FL model was further tested on three additional IoT datasets—BoT-IoT, WUSTL-IIoT-2021, and Edge-IIoTset—to validate its generalizability. The results show that the proposed model achieves the highest performance measures in most classes, with exceptional performance in the detection of DDoS attacks.

In future work, we will study how to improve the performance of the FL-based models and to lighten the local model more to better suit IoT devices with low resources. Compression techniques such as weight pruning, quantization, and others can be used to lighten the IDS model. Since the aggregation algorithm plays a crucial role in the FL process, particularly in heterogeneous and resource-constrained IoT environments, this requires designing an aggregation algorithm specifically for this purpose. Finally, we aim to simulate the proposed IDS model in which the FL server serves as the edge device and the IoT device, or end node, acts as the FL client. As the cluster head is chosen based on the average Silhouette score, it will be close to all neighbors within the cluster to minimize the communication latency of inner aggregation.

Author Contributions: Conceptualization, S.A. and M.E.; methodology, S.A.; software, S.A.; validation, S.A.and M.E.; formal analysis, S.A.; investigation, S.A.; resources, S.A.; data curation, A.S.; writing—original draft preparation, S.A.; writing—review and editing, S.A. and M.E.; visualization, S.A..; supervision, M.E. and S.A.A.; project administration, M.E. and S.A.A.; funding acquisition, S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

Abbreviation	Definition
AE	Autoencoder
AIDS	Anomaly-Based Intrusion Detection System
ANN	Artificial Neural Network
B-FL	Blockchain Federated Learning
CFL	Clustered Federated Learning
CNN	Convolutional Neural Network
DT	Decision Tree
DL	Deep Learning
DDoS	Distributed Denial of Service
DoS	Denial of Service
D2D	Device-to-Device
FL	Federated Learning
HIDS	Host-Based Intrusion Detection System
IDS	Intrusion Detection System
IoT	Internet of Things
IoV	Internet of Vehicle
KNN	K-Nearest Neighbor
LSTM	Long–Short-Term Memory
ML	Machine Learning
NB	Naïve Bayes
NIDS	Network-Based Intrusion Detection System
PCA	Principal Components Analysis
RF	Random Forest
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SIDS	Signature-Based Intrusion Detection System
SVM	Support Vector Machines
TL	Transfer Learning

References

1. Stallings, W.; Brown, L. *Computer Security: Principles and Practice*, 4th ed.; Pearson: 2018; pp. 274–299.
2. Krontiris, I.; Giannetsos, T.; Dimitriou, T. LIDeA: A Distributed Lightweight Intrusion Detection Architecture for Sensor Networks. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks*, Istanbul, Turkey, 2008; pp. 1–10. doi:10.1145/1460877.1460903.
3. Riecker, M.; Biedermann, S.; El Bansarkhani, R.; Hollick, M. Lightweight Energy Consumption-Based Intrusion Detection System for Wireless Sensor Networks. *International Journal of Information Security* **2015**, *14*, 155–167. doi:10.1007/s10207-014-0241-1.
4. Derhab, A.; Aldweesh, A.; Emam, A.Z.; Khan, F.A.; Wang, X. Intrusion Detection System for Internet of Things Based on Temporal Convolution Neural Network and Efficient Feature Engineering. *Wireless Communications and Mobile Computing* **2020**, *2020*, 1–16. doi:10.1155/2020/6689134.
5. Ahmad, R.; Wazirali, R.; Abu-Ain, T. Machine Learning for Wireless Sensor Networks Security: An Overview of Challenges and Issues. *Sensors* **2022**, *22*, 4730. doi:10.3390/s22134730.
6. Thakkar, A.; Lohiya, R. A Review on Machine Learning and Deep Learning Perspectives of IDS for IoT: Recent Updates, Security Issues, and Challenges. *Archives of Computational Methods in Engineering* **2021**, *28*, 3211–3243. doi:10.1007/s11831-020-09496-0.
7. Campos, E.M.; Saura, P.F.; González-Vidal, A.; Hernández-Ramos, J.L.; Bernal Bernabé, J.; Baldini, G.; Skarmeta, A. Evaluating Federated Learning for Intrusion Detection in Internet of Things: Review and Challenges. *Computer Networks* **2022**, *203*, 108661. doi: 10.1016/j.comnet.2021.108661.

8. Agrawal, S.; Sarkar, S.; Aouedi, O.; Yenduri, G.; Piamrat, K.; Alazab, M.; Bhattacharya, S.; Maddikunta, P.K.R.; Gadekallu, T.R. Federated Learning for Intrusion Detection System: Concepts, Challenges and Future Directions. *Computer Communications* **2022**, *195*, 346–361.
9. Savazzi, S.; Nicoli, M.; Rampa, V. Federated Learning with Cooperating Devices: A Consensus Approach for Massive IoT Networks. *IEEE Internet of Things Journal* **2020**, *7*, 4641–4654. doi: 10.1109/JIOT.2020.2964162.
10. Gholami, A.; Torkzaban, N.; Baras, J.S. Trusted Decentralized Federated Learning. In Proceedings of the 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 1–6. doi: 10.1109/CCNC49033.2022.9700624.
11. Yang, Z.; Shi, Y.; Zhou, Y.; Wang, Z.; Yang, K. Trustworthy Federated Learning via Blockchain. *IEEE Internet of Things Journal* **2023**, *10*, 92–109. doi: 10.1109/JIOT.2022.3201117.
12. Sun, Y.; Shao, J.; Mao, Y.; Wang, J.H.; Zhang, J. Semi-Decentralized Federated Edge Learning for Fast Convergence on Non-IID Data. In Proceedings of the 2022 IEEE Wireless Communications and Networking Conference (WCNC), 1898–1903. doi: 10.1109/WCNC51071.2022.9771904.
13. Wu, W.; He, L.; Lin, W.; Mao, R.; Maple, C.; Jarvis, S. SAFA: A Semi-Asynchronous Protocol for Fast Federated Learning with Low Overhead. *IEEE Transactions on Computers* **2021**, *70*, 655–668. doi: 10.1109/TC.2020.2994391.
14. Liu, L.; Zhang, J.; Song, S.H.; Letaief, K.B. Client-Edge-Cloud Hierarchical Federated Learning. In Proceedings of the *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 1–6. doi: 10.1109/ICC40277.2020.9148862.
15. Ghimire, B.; Rawat, D.B. Recent Advances on Federated Learning for Cybersecurity and Cybersecurity for Federated Learning for Internet of Things. *IEEE Internet of Things Journal* **2022**, *9*, 8229–8249. doi: 10.1109/JIOT.2022.3150363.
16. Khraisat, A.; Alazab, A. A Critical Review of Intrusion Detection Systems in the Internet of Things: Techniques, Deployment Strategy, Validation Strategy, Attacks, Public Datasets, and Challenges. *Cybersecurity* **2021**, *4*(1), 18. doi: 10.1186/s42400-021-00077-7.
17. Tran, D.H.; Nguyen, V.L.; Utama, I.B.K.Y.; Jang, Y.M. An Improved Sensor Anomaly Detection Method in IoT System using Federated Learning. In Proceedings of the *International Conference on Ubiquitous and Future Networks (ICUFN)*, 466–469. doi: 10.1109/ICUFN55119.2022.9829561.
18. Tahir, M.; Ali, M.I. On the Performance of Federated Learning Algorithms for IoT. *IoT* **2022**, *3*(2), 273–284. doi: 10.3390/iot3020016.
19. Lee, S.J.; Yoo, P.D.; Asyhari, A.T.; Jhi, Y.; Chermak, L.; Yeun, C.Y.; Taha, K. IMPACT: Impersonation Attack Detection via Edge Computing Using Deep Autoencoder and Feature Abstraction. *IEEE Access* **2020**, *8*, 65520–65529. doi: 10.1109/ACCESS.2020.2985089.
20. Nguyen, X.H.; Nguyen, X.D.; Huynh, H.H.; Le, K.H. Realguard: A Lightweight Network Intrusion Detection System for IoT Gateways. *Sensors* **2022**, *22*(2), 1–18. doi: 10.3390/s22020432.
21. Roy, S.; Li, J.; Choi, B.J.; Bai, Y. A Lightweight Supervised Intrusion Detection Mechanism for IoT Networks. *Future Generation Computer Systems* **2022**, *127*, 276–285. doi: 10.1016/j.future.2021.09.027.
22. Huong, T.T.; Bac, T.P.; Long, D.M.; Thang, B.D.; Binh, N.T.; Luong, T.D.; Phuc, T.K. LocKedge: Low-Complexity Cyberattack Detection in IoT Edge Computing. *IEEE Access* **2021**, *9*, 29696–29710. doi: 10.1109/ACCESS.2021.3058528.
23. Rashid, M.M.; Khan, S.U.; Eusufzai, F.; Redwan, M.A.; Sabuj, S.R.; Elsharief, M. A Federated Learning-Based Approach for Improving Intrusion Detection in Industrial Internet of Things Networks. *Network* **2023**, *3*, 158–179. doi: 10.3390/network3010008.
24. Hamdi, N. Federated Learning-Based Intrusion Detection System for Internet of Things. *International Journal of Information Security* **2023**, *22*(6), 1937–1948. doi: 10.1007/s10207-023-00727-6.
25. Zhang, T.; He, C.; Ma, T.; Gao, L.; Ma, M.; Avestimehr, S. Federated Learning for Internet of Things. In Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems, Coimbra, Portugal, 2021; pp. 413–419. doi: 10.1145/3485730.3493444.
26. Regan, C.; Nasajpour, M.; Parizi, R.M.; Pouriyeh, S.; Dehghantanha, A.; Choo, K.-K.R. Federated IoT Attack Detection Using Decentralized Edge Data. *Machine Learning with Applications* **2022**, *8*, 100263. doi: 10.1016/j.mlwa.2022.100263.
27. Attota, D.C.; Mothukuri, V.; Parizi, R.M.; Pouriyeh, S. An Ensemble Multi-View Federated Learning Intrusion Detection for IoT. *IEEE Access* **2021**, *9*, 117734–117745. doi: 10.1109/ACCESS.2021.3107337.

28. Driss, M.; Almomani, I.; Huma, Z.E.; Ahmad, J. A Federated Learning Framework for Cyberattack Detection in Vehicular Sensor Networks. *Complex and Intelligent Systems* **2022**, *8*, 4221–4235. doi: 10.1007/s40747-022-00705-w.
29. Friha, O.; Ferrag, M.A.; Shu, L.; Maglaras, L.; Choo, K.K.R.; Nafaa, M. FELIDS: Federated Learning-Based Intrusion Detection System for Agricultural Internet of Things. *Journal of Parallel and Distributed Computing* **2022**, *165*, 17–31. doi: 10.1016/j.jpdc.2022.03.003.
30. Chen, Z.; Liao, W.; Hua, K.; Lu, C.; Yu, W. Towards Asynchronous Federated Learning for Heterogeneous Edge-Powered Internet of Things. *Digital Communications and Networks* **2021**, *7*(3), 317–326. doi: 10.1016/j.dcan.2021.04.001.
31. Rjoub, G.; Wahab, O.A.; Bentahar, J.; Bataineh, A. Trust-driven reinforcement selection strategy for federated learning on IoT devices. *Computing* **2024**, *106*(4), 1273–1295. doi: 10.1007/s00607-022-01078-1.
32. Chen, H.; Huang, S.; Zhang, D.; Xiao, M.; Skoglund, M.; Poor, H.V. Federated Learning Over Wireless IoT Networks With Optimized Communication and Resources. *IEEE Internet of Things Journal* **2022**, *9*(17), 16592–16605. doi: 10.1109/JIOT.2022.3151193.
33. Nguyen, M.D.; Lee, S.M.; Pham, Q.V.; Hoang, D.T.; Nguyen, D.N.; Hwang, W.J. HCFL: A High Compression Approach for Communication-Efficient Federated Learning in Very Large Scale IoT Networks. *IEEE Transactions on Mobile Computing* **2022**, August. doi: 10.1109/TMC.2022.3190510.
34. Abosata, N.; Al-Rubaye, S.; Inalhan, G. Customised Intrusion Detection for an Industrial IoT Heterogeneous Network Based on Machine Learning Algorithms Called FTL-CID. *Sensors* **2023**, *23*(1), Article 321. doi: 10.3390/s23010321.
35. Nguyen, T.D.; Marchal, S.; Miettinen, M.; Fereidooni, H.; Asokan, N.; Sadeghi, A.-R. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In Proceedings of the *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 756–767. doi: 10.1109/ICDCS.2019.00080.
36. Wang, N.; Chen, Y.; Hu, Y.; Lou, W.; Hou, Y.T. FeCo: Boosting Intrusion Detection Capability in IoT Networks via Contrastive Learning. In Proceedings of the *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1409–1418. doi: 10.1109/INFOCOM48880.2022.9796926.
37. Tabassum, A.; Erbad, A.; Lebda, W.; Mohamed, A.; Guizani, M. FEDGAN-IDS: Privacy-preserving IDS using GAN and Federated Learning. *Computer Communications* **2022**, *192*, 299–310. doi: 10.1016/j.comcom.2022.06.015.
38. Bukhari, S.M.S.; Zafar, M.H.; Houran, M.A.; Moosavi, S.K.R.; Mansoor, M.; Muaaz, M.; Sanfilippo, F. Secure and privacy-preserving intrusion detection in wireless sensor networks: Federated learning with SCNN-Bi-LSTM for enhanced reliability. *Ad Hoc Networks* **2024**, *155*, Article 103407. doi: 10.1016/j.adhoc.2024.103407.
39. Al-Abiad, M.S.; Obeed, M.; Hossain, M.J.; Chaaban, A. Decentralized Aggregation for Energy-Efficient Federated Learning via D2D Communications. *IEEE Transactions on Communications* **2023**, *71*(6), 3333–3351. doi: 10.1109/TCOMM.2023.3253718.
40. Herabad, M.G. Communication-efficient semi-synchronous hierarchical federated learning with balanced training in heterogeneous IoT edge environments. *Internet of Things* **2023**, *21*, Article 100642. doi: 10.1016/j.iot.2022.100642.
41. Alotaibi, A.; Barnawi, A. IDSoft: A federated and softwarized intrusion detection framework for massive Internet of Things in 6G networks. *Journal of King Saud University - Computer and Information Sciences* **2023**, *35*(6), Article 101575. doi: 10.1016/j.jksuci.2023.101575.
42. Neto, E.C.P.; Dadkhah, S.; Ferreira, R.; Zohourian, A.; Lu, R.; Ghorbani, A.A. CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment. *Sensors* **2023**, *23*(13), Article 5941. doi: 10.3390/s23135941.
43. Sattler, F.; Muller, K.R.; Samek, W. Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization under Privacy Constraints. *IEEE Transactions on Neural Networks and Learning Systems* **2021**, *32*(8), 3710–3722. doi: 10.1109/TNNLS.2020.3015958.
44. Briggs, C.; Fan, Z.; Andras, P. Federated Learning with Hierarchical Clustering of Local Updates to Improve Training on Non-IID Data. In *Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN)*; IEEE: 2020; pp. 1–9. doi: 10.1109/IJCNN48605.2020.9207469.
45. Sáez-de-Cámara, X.; Flores, J.L.; Arellano, C.; Urbieta, A.; Zurutuza, U. Clustered Federated Learning Architecture for Network Anomaly Detection in Large-Scale Heterogeneous IoT Networks. *Computers & Security* **2023**, *131*, Article 103299. doi: 10.1016/j.cose.2023.103299.
46. Hooshyar, D.; Yang, Y.; Pedaste, M.; Huang, Y.M. Clustering Algorithms in an Educational Context: An Automatic Comparative Approach. *IEEE Access* **2020**, *8*, 146994–147014. doi: 10.1109/ACCESS.2020.3014948.

47. Yin, H.; Aryani, A.; Petrie, S.; Nambissan, A.; Astudillo, A.; Cao, S. A Rapid Review of Clustering Algorithms. *arXiv preprint* **2024**. Available online: https://arxiv.org/abs/2401.07389.
48. Alkhayrat, M.; Aljnidi, M.; Aljoumaa, K. A Comparative Dimensionality Reduction Study in Telecom Customer Segmentation Using Deep Learning and PCA. *Journal of Big Data* **2020**, *7*(1), 9. doi: 10.1186/s40537-020-0286-0.
49. Irani, J.; Pise, N.; Phatak, M. Clustering Techniques and the Similarity Measures Used in Clustering: A Survey. *International Journal of Computer Applications* **2016**, *134*(7), 975–8887.
50. Shirkhorshidi, A.S.; Aghabozorgi, S.; Wah, T.Y. A Comparison Study on Similarity and Dissimilarity Measures in Clustering Continuous Data. *PLOS ONE* **2015**, *10*(12), 1–20. doi: 10.1371/journal.pone.0144059.
51. Ghazal, T.M.; Hussain, M.Z.; Said, R.A.; Nadeem, A.; Hasan, M.K.; Ahmad, M.; Khan, M.A.; Naseem, M.T. Performances of k-means Clustering Algorithm with Different Distance Metrics. *Intelligent Automation and Soft Computing* **2021**, *30*(2), 735–742. doi: 10.32604/iasc.2021.019067.
52. Schubert, E.; Sander, J.; Ester, M.; Kriegel, H.P.; Xu, X. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Transactions on Database Systems* **2017**, *42*(3). doi: 10.1145/3068335.
53. Qi, P.; Chiaro, D.; Guzzo, A.; Ianni, M.; Fortino, G.; Piccialli, F. Model Aggregation Techniques in Federated Learning: A Comprehensive Survey. *Future Gener. Comput. Syst.* **2024**, *150*, 272–293. doi: 10.1016/j.future.2023.09.008.
54. Koroniotis, N.; Moustafa, N.; Sitnikova, E.; Turnbull, B. Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset. *Future Generation Computer Systems* **2019**, *100*, 779-796. doi: 10.1016/j.future.2019.05.041.
55. Jain, R. Industrial IoT (IIoT) and Industry 4.0: Research and Educational Material. *Washington University in St. Louis*, 2024. Available online: http://www.cse.wustl.edu/ jain/iiot2/index.html. Accessed: 2024-12-11.
56. Novikov, A.V. PyClustering: Data Mining Library. *Journal of Open Source Software* **2019**, *4*(36), 1230. doi: 10.21105/joss.01230.
57. Alani, M.M.; Damiani, E.; Ghosh, U. DeepIIoT: An Explainable Deep Learning-Based Intrusion Detection System for Industrial IoT. In *Proceedings of the 2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*; IEEE: 2022; pp. 169-174. doi: 10.1109/ICDCSW56584.2022.00040.
58. Belarbi, O.; Spyridopoulos, T.; Anthi, E.; Mavromatis, I.; Carnelli, P.; Khan, A. Federated Deep Learning for Intrusion Detection in IoT Networks. *arXiv preprint* **2023**. Available online: https://arxiv.org/abs/2306.02715.
59. Moustafa, N. A New Distributed Architecture for Evaluating AI-Based Security Systems at the Edge: Network TON_IoT Datasets. *ResearchGate*, 2021. Available online: https://www.researchgate.net/publication/352055999.
60. Lazzarini, R.; Tianfield, H.; Charissis, V. Federated Learning for IoT Intrusion Detection. *AI (Switzerland)* **2023**, *4*(3), 509-530. doi: 10.3390/ai4030028.
61. Alotaibi, A.; Barnawi, A. LightFIDS: Lightweight and Hierarchical Federated IDS for Massive IoT in 6G Network. *Arabian Journal for Science and Engineering* **2024**, *49*, 4383-4399. doi: 10.1007/s13369-023-08439-8.
62. Huang, J.; Ye, L.; Kang, L. FedSR: A Semi-Decentralized Federated Learning Algorithm for Non-IIDness in IoT System. *arXiv preprint* **2024**. Available online: https://doi.org/10.48550/arXiv.2403.14718.
63. Ferrag, M.A.; Friha, O.; Hamouda, D.; Maglaras, L.; Janicke, H. Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning. *IEEE Access* **2022**, *10*, 40281-40306. doi: 10.1109/ACCESS.2022.3165809.
64. Aguru, A.D.; Erukala, S.B. A Lightweight Multi-Vector DDoS Detection Framework for IoT-Enabled Mobile Health Informatics Systems Using Deep Learning. *Inf. Sci.* **2024**, *662*, 20. doi: 10.1016/j.ins.2024.120209.
65. Eid, A.M.; Soudan, B.; Bou Nassif, A.; Injadat, M.N. Enhancing Intrusion Detection in IIoT: Optimized CNN Model with Multi-Class SMOTE Balancing. *Neural Computing and Applications* **2024**. doi: 10.1007/s00521-024-09857-x.
66. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Networks. *arXiv preprint arXiv:1406.2661*, 2014. Available online: https://arxiv.org/abs/1406.2661 (accessed on 15 December 2024).
67. Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein Generative Adversarial Networks. *arXiv preprint* **2017**, arXiv:1704.00028. Available online: https://doi.org/10.48550/arXiv.1704.00028 (accessed on [Access Date]).
68. Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. Improved Training of Wasserstein GANs. *arXiv preprint* **2017**, arXiv:1704.00028. Available online: https://arxiv.org/abs/1704.00028 (accessed on [Access Date]).

69. Becerra-Suarez, F.L.; Tuesta-Monteza, V.A.; Mejia-Cabrera, H.I.; Arcila-Diaz, J. Performance Evaluation of Deep Learning Models for Classifying Cybersecurity Attacks in IoT Networks. *Informatics* **2024**, *11*(2), 32. Available online: https://www.mdpi.com/2227-9709/11/2/32 (accessed on [Access Date]). `https://doi.org/10.3390/informatics11020032`.
70. Hizal, S.; Cavusoglu, U.; Akgun, D. A Novel Deep Learning-Based Intrusion Detection System for IoT DDoS Security. *Internet of Things* **2024**, *28*, 101336. Available online: https://www.sciencedirect.com/science/article/pii/S2542660524002774 (accessed on [Access Date]). `https://doi.org/10.1016/j.iot.2024.101336`.
71. Zhao, X.; Fok, K.W.; Thing, V.L.L. Enhancing Network Intrusion Detection Performance using Generative Adversarial Networks. *arXiv preprint* **2024**, arXiv:2404.07464. Available online: http://arxiv.org/abs/2404.07464 (accessed on [Access Date]).