Intelligent Automation & Soft Computing

Tech Science Press

DOI: 10.32604/iasc.2024.056792

ARTICLE

Distributed Federated Split Learning Based Intrusion Detection System

Rasha Almarshdi[1,2,*], Etimad Fadel[1], Nahed Alowidi[1] and Laila Nassef[1]

[1]Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, 21589, Saudi Arabia

[2]Department of Computer Science, Faculty of Computer Science and Engineering, University of Hail, Hail, 55476, Saudi Arabia

*Corresponding Author: Rasha Almarshdi. Email: rabdullahalmarshdi@stu.kau.edu.sa

Received: 31 July 2024 Accepted: 23 September 2024 Published: 31 October 2024

ABSTRACT

The Internet of Medical Things (IoMT) is one of the critical emerging applications of the Internet of Things (IoT). The huge increases in data generation and transmission across distributed networks make security one of the most important challenges facing IoMT networks. Distributed Denial of Service (DDoS) attacks impact the availability of services of legitimate users. Intrusion Detection Systems (IDSs) that are based on Centralized Learning (CL) suffer from high training time and communication overhead. IDS that are based on distributed learning, such as Federated Learning (FL) or Split Learning (SL), are recently used for intrusion detection. FL preserves data privacy while enabling collaborative model development. However, FL suffers from high training time and communication overhead. On the other hand, SL offers advantages in terms of computational resources, but it faces challenges such as communication overhead and potential security vulnerabilities at the split point. Federated Split Learning (FSL) has proposed overcoming the problems of both FL and SL and offering more secure, efficient, and scalable distribution systems. This paper proposes a novel distributed FSL (DFSL) system to detect DDoS attacks. The proposed DFSL enhances detection accuracy and reduces training time by designing an adaptive aggregation method based on the early stopping strategy. However, the increased number of clients leads to increasing communication overheads. We further propose a Multi-Node Selection (MNS) based Best Channel-Best l_2-Norm (BC-BN2) selection scheme to reduce communication overhead. Two DL models are used to test the effectiveness of the proposed system, including a Convolutional Neural Network (CNN) and CNN with Long Short-Term Memory (LSTM) on two modern datasets. The performance of the proposed system is compared with three baseline distributed approaches such as FedAvg, Vanilla SL, and SplitFed algorithms. The proposed system outperforms the baseline algorithms with an accuracy of 99.70% and 99.87% in CICDDoS2019 and LITNET-2020 datasets, respectively. The proposed system's training time and communication overhead are 30% and 20% less than the baseline algorithms.

KEYWORDS

IDS; DFSL; DDoS attacks; CNN; CNN+LSTM

1 Introduction

The Internet of Things (IoT) is the most important technology, and it has grown exponentially. IoT is a set of technologies that connect a wide range of IoT devices and generate a large volume of

data with features of larger size, higher velocity, and heterogeneity [1]. At the same time, the evolution of 5G networks has given strength to the growth of the IoT. 5G is characterized by higher capacity and lower latency to facilitate communication of billions of devices over the Internet. Therefore, the 5G network should be integrated with modern technologies to provide exceptional services [2]. One of the important 5G-enabling technologies is Multiple-Input-Multiple-Output (MIMO), which has been used in recent years to provide high spectral efficiency and high throughput [3].

The growth of 5G-based IoT (5G-IoT) comes with challenges related to IoT data, such as security and privacy, and they also have an impact on computational complexity and cost in data storage and data processing. An important IoT domain that needs 5G features in transmission and channel utilization is the Internet of Medical Things (IoMT) [4]. The IoMT is a latency-sensitive application expected to be deployed based on 5G technologies. It will have a great role in maintaining the sustainability of smart cities [5]. Fig. 1 describes the 5G-IoMT architecture, including three layers: sensing layer, fog computing layer, and cloud computing layer. First, the data is collected using various sensors or medical devices in the sensing layer. These devices are resource-constrained and have low computation power. The data is passed to the fog computing layer via communication protocols, where more powerful computing fog servers process it to avoid latency. The data is transmitted in the 5G structure through multiple Base Stations (BSs) that employ fog computing. Finally, the data is transmitted from the fog computing layer to the cloud computing layer to store and update the patient's data [6].

The significant increase in transmitting a large volume of data across distributed networks makes security one of the most important challenges in IoMT. Recently, Distributed Denial of Service (DDoS) attacks have increased dramatically, leading to significant consequences [7]. One of the most popular real examples of DDoS against medical institutions was in 2016 when the attacker used malware to breach the Banner Health network in Arizona and access the patient's information, including patient's social numbers, dates of services, and information related to the insurance. The attack went undetected for a month, with 3.7 million patients affected. It cost $1.25 million to resolve the data breach issue [8].

Intrusion Detection Systems (IDSs) based on the Centralized Learning (CL) approach are widely used to detect security attacks. CL is a traditional learning approach where the data is uploaded from each connected client device to the cloud server to train the entire model and distribute it to all devices [9]. IDSs using Machine Learning (ML) methods are impractical in the new network environment, and they are not effective in detecting attacks in massive and distributed environments. Although centralized systems based on Deep Learning (DL) achieve acceptable accuracy, they include limitations such as connectivity, bandwidth, latency, communication overhead, computation power, and distributed data security. In addition, it is not scalable to the size of IoMT [10].

IDSs based on distributed learning, such as Federated Learning (FL) or Split Learning (SL), are used in distributed environments to overcome the limitations of CL. FL is an efficient approach that trains the model with its local data among multiple clients and shares the model update with the server. FL preserves data privacy while enabling collaborative model development. However, FL suffers from high training time and communication overhead [11]. On the other hand, SL splits the model into two parts, one for clients and one for servers, which reduces the computational cost and can help reduce training time [12]. SL offers advantages in terms of computational resources, but it faces challenges such as communication overhead and potential security vulnerabilities at the split point. Both FL and SL are not efficient when the number of clients increases, having heterogeneous datasets, and with constraints on communication resources. Federated Split Learning (FSL) is emerging research

combining FL and SL that could solve problems of both FL and SL and offer more secure, efficient, and scalable distribution systems [13]. Most existing IDSs are not scalable to the network size. It suffers from low detection accuracy, training time, and communication overhead. These issues have negatively affected the throughput of real-time applications such as IoMT, and we need to solve them urgently.

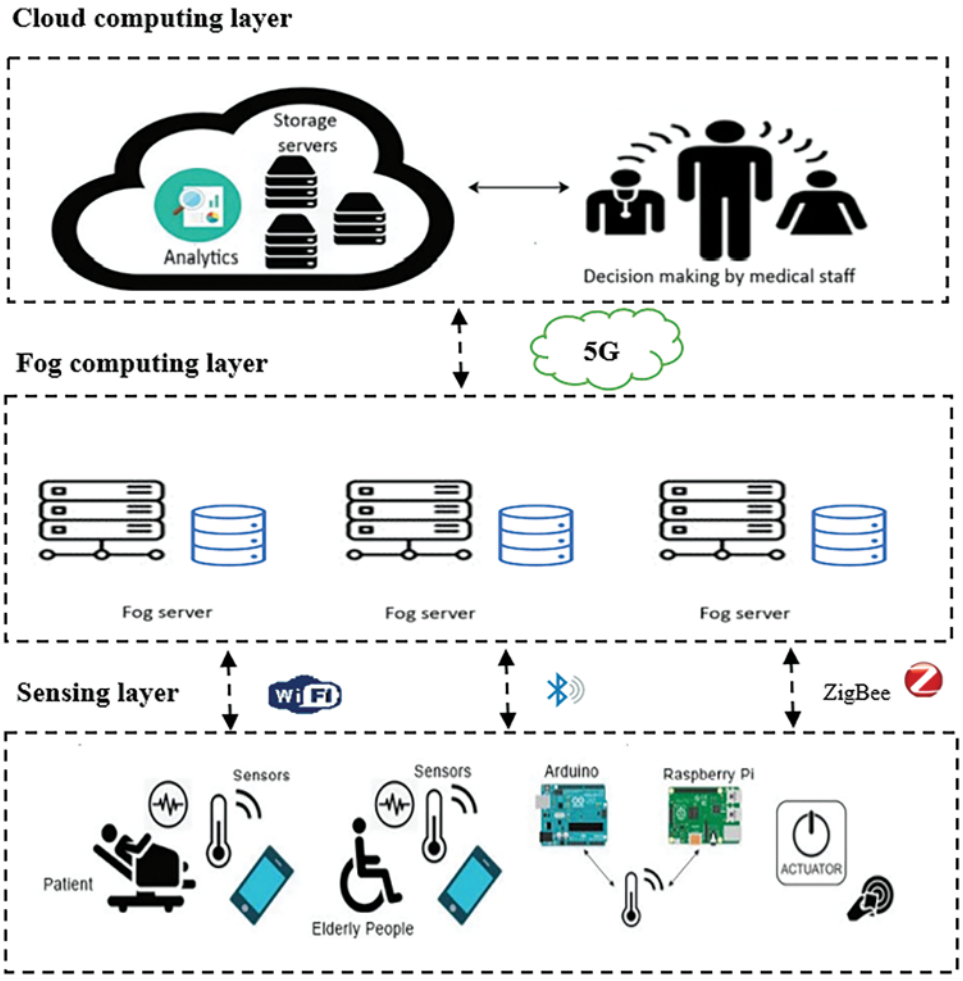

Figure 1: 5G-IoMT system architecture

Recently, a user selection strategy has been used to reduce communication overheads. In FL, different selection schemes have been proposed in [14], including four schemes. The first scheme is Best Channel (BC), where the user is selected based only on the channel condition. The second scheme is Best l_2-Norm (BN2), where the user is selected according to the significance of the model update. The third scheme is Best Channel-Best l_2-Norm (BC-BN2), which considers the channel condition and the significance of the model update. The last scheme is Best l_2-Norm-Channel (BN2-C), where each user performs a Decentralized Stochastic Gradient Descent (D-SGD) quantization scheme assuming all bandwidth availability to itself and finds the resultant quantized vector and the significant importance of the local update is sent to the server. In [15], a user equipment UE selection scheme named Multi-Arm Bound (MAB) is based on BC-BN2 (MAB-BC-BN2). The proposed scheme uses the discounted Upper Confidence Bound (UCB) strategy to make the selection decision based on the UCB score.

Nevertheless, the above schemes, including channel quality and the importance of local model update, are popular selection strategies in FL. In practice, it is not practical because it is difficult to get an accurate selection before the learning process is executed, which leads to more computation and communication consumption.

Motivated by the above, this paper proposed a distributed FSL (DFSL) system to detect DDoS attacks, enhancing detection accuracy and reducing training time and communication overhead. Three research questions were formulated as follows:

Q1: How do we distribute the detection model with high detection accuracy and low training time?

Q2: How can the communication overhead of the system be reduced?

Q3: How can we test the effectiveness of the proposed system?

To address the research questions of this paper, the following contributions were identified:

1. Propose a novel distributed FSL (DFSL) system to detect DDoS attacks in IoMT. An adaptive aggregation method is designed based on the early stopping strategy to reduce training time and enhance detection accuracy.
2. Propose a Multi-Node Selection (MNS) based Best Channel-Best l_2-Norm (BC-BN2) selection scheme to reduce communication overhead. The nodes are selected based on the importance of local update, channel quality, and the nearest distance from the server. The MNS dynamically identifies the irrelevant local updates during the training process to avoid computation and communication resources consuming.
3. Test the system performance using Convolutional Neural Network (CNN) and a hybrid model that combines the CNN and long Short-Term Memory (LSTM), on two modern datasets such as CICDDoS2019 and LITNET-2020, and compare the performance with baseline distributed approaches such as FedAvg, Vanilla SL, and SplitFed algorithms.

The paper is organized as follows. A brief overview of FL, SL, FSL, and related works is provided in Section 2. Section 3 describes the problem formulation. The proposed work of the DFSL system and MNS scheme are described in Section 4. Section 5 describes the dataset description and data preprocessing. The performance results and comparative analysis are presented in Section 6. The discussion is presented in Section 7. Finally, the conclusion and future work is described in Section 8.

2 Literature Review

This section described a background for the FL, SL, and FSL approaches. Then, related works that are based on ML, CL, and distributed learning IDS in IoMT networks are presented.

2.1 Federated Learning

In Federated Learning (FL) [16], the model is trained at each client in parallel with its local data for its local epochs. After that, the local model updates are sent to the server for model aggregation. The server aggregates the received local updates, generates the global model update, and sends it back to the clients so they can train for the next round. After receiving the global model update, each client trains the global parameters on its local data and sends its local update to the server. This process continues until the model converges. Federated Averaging (FedAvg) algorithm [17] is the popular centralized aggregation algorithm in FL. FedAvg algorithm considers a weighted average of the gradients for the model updates. Based on the FedAvg procedure, as shown in Fig. 2, the server initializes the global model parameter w_0 and broadcasts the initial global parameter to the clients. Then, each client n

receives the initial parameter and starts the local training with its local data D_n. The local loss function L for each client n with D_n is calculated as

$$L_n(w) = \frac{1}{|D_n|} \sum_{i \in D_n} l(w, s_n^i) \tag{1}$$

where $|D_n|$ is the dataset size and $l(w, s_n^i)$ is the loss function of the model at data sample s_n^i. The global loss function $L(w)$ over the data for the clients is calculated as

$$L(w) = \frac{\sum_{n=1}^{N} D_n L_n(w)}{D} \tag{2}$$

where D is the global data, and N is the number of clients. The local training updates w_{t+1}^n is calculated as

$$w_{t+1}^n = w_t^n - \alpha_t \nabla g(w_t^n) \tag{3}$$

where α_t is the learning rate, $\nabla g(w_t^n)$ is the gradients for w_t^n of loss function L. The global model parameter w_t is updated at the server by aggregating each local update w_{t+1}^n from each client and sending it back to the clients; this process is repeated until the loss function reaches the optimal value for the global model. The global model update is represented as

$$w_t = \frac{\sum_{n=1}^{N} D_n w_{t+1}^n}{D} \tag{4}$$

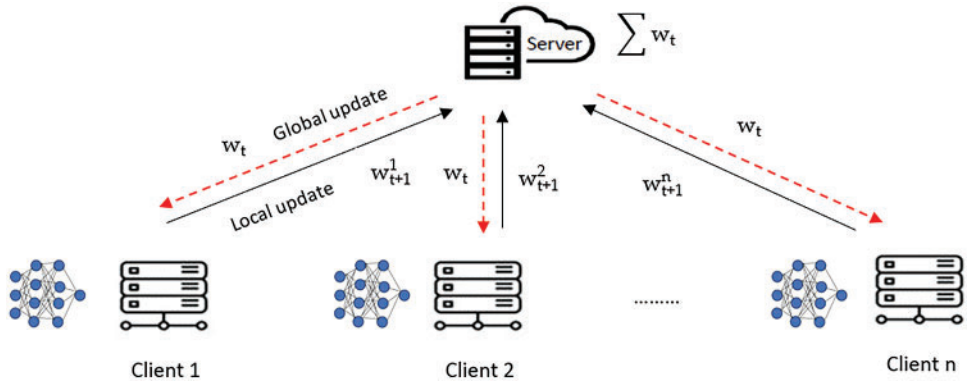

Figure 2: FedAvg architecture

2.2 Split Learning

In Split Learning (SL), the model is trained sequentially with its local data. The training raw data is stored in the client, and the server cannot access the raw data. Four SL configuration strategies include Vanilla SL, U-shaped SL, Vertically SL, and two-part SL configurations. Vanilla SL [18] is the most popular SL algorithm where the model is partitioned into at least two parts and shares labels. The first part runs on the client w_{t+1}^n, and the second part runs on the server w_t^s; it trains its smashed data up to the cut layer C, as shown in Fig. 3. Each client needs to update its local model parameters before starting the next round of training. In the sequential training, the client performs forward propagation w_{t+1}^n on its local data D_n and sends the activations a_t^n up to the cut layer to the server. Then, the server receives the client's activation a_t^n and performs the forward propagation until the last layer of the model w_t^s. After that, the server performs the backpropagation, and the gradient g_t^n of the client cut layer is

sent back to the client. Then, the client updates the model on its side and sends it back to the server for updates.

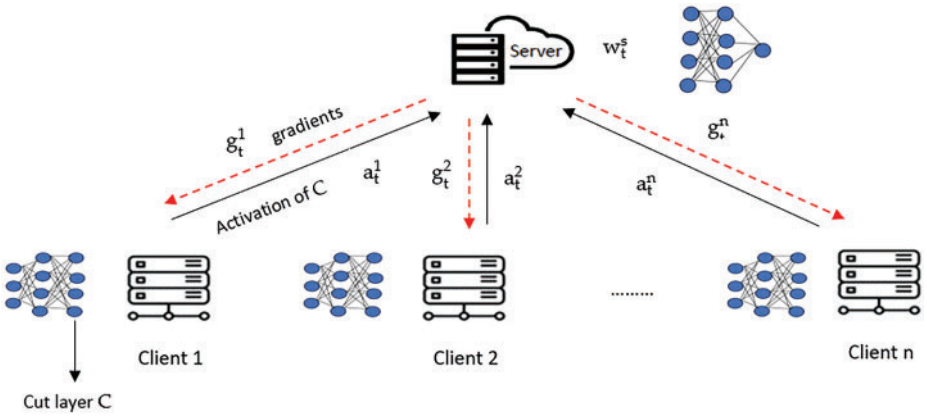

Figure 3: Vanilla SL architecture

U-shaped SL is another split configuration similar to Vanilla SL but without label sharing. Because the client does not share the label, the client completes learning. In U-shaped SL, the client has one or more bottom layers and one or two top layers, whereas the middle layers are allocated by the server. Vertically SL is a way to train a split model with different data without sharing data. Each client trains a part of the model up to the cut layer C. After that, the data is sent to the server to complete the rest of the learning process. Finally, two-part SL is a strategy to split the model into only two parts between clients and server [19]. The SL helps reduce training time where the model is portioned among multiple clients and server, but most of the SL algorithms suffer from high communication overhead.

2.3 Federated Split Learning

Federated Split Learning (FSL) is a promising distributed learning that combines FL and SL approaches and takes advantage of the two approaches. It trains the model in parallel among multiple distributed clients in FL and splits the entire model between the clients and server in SL. SplitFed is the first method in FSL that eliminates the drawbacks of FL and SL. Unlike FL, the model is portioned and trained among the clients and the server to reduce training time. On the other hand, it does their training in parallel to reduce computation time and enhance learning performance, as in FL. It works similarly to the classic SL setting, except the existing fed server on the client side performs federated aggregation [20]. MiniBatch-SFL is proposed to incorporate MiniBatch SGD to FSL, where the clients train the client-side model in an FL fashion, and the server trains the server side and analyzes the convergence of the data [21]. A Communication and Storage Efficient FSL (CSE-FSL) strategy is proposed using the auxiliary network to update the local model at the client while keeping only a single model at the server, which helps to avoid the communication of gradients from the server and reduces the server resource requirement [22]. Finally, a comparison of the FL and SL settings regarding communication efficiency is explained in [23].

2.4 Related Works

The following presents the most current IDS for detecting attacks in the IoMT environment.

Saheed et al. [24] developed an efficient IDS in IoMT based on Deep Recurrent Neural Network (DRNN) and ML models such as Random Forest (RF), Decision Tree (DT), K-Nearest Neighbors

(KNN), and ridge classifiers. The goal is to classify and forecast unexpected cyber-attacks. They evaluated the developed IDS with the NSL-KDD dataset, and it outperformed the existing approaches with an accuracy of 99.76%. Manimurugan et al. [25] proposed a DL-based method Deep Belief Network (DBN) algorithm for IDS in IoMT. The CICIDS2017 dataset is used to test the proposed algorithm. The results achieved 96.67% accuracy for the DoS/DDoS class, 95.21% precision, 97.34% recall, 0.97 F-measure, and 97.31% detection rate. Usman et al. [26] proposed a framework that divides an underlying wireless multimedia sensor network into many clusters in IoMT. Each cluster has a cluster head, which is responsible for protecting the privacy data via data and location coordinates aggregation. Then, the aggregated data are processed on the cloud using Artificial Neural Network (ANN). The proposed P2DCA framework is implemented with the CICIDS2017 dataset and compared with the existing schemes. It achieved 150 min training time, 350 MB communication overhead, and 95.56% accuracy. Ren et al. [27] proposed Data Optimization IDS (DO-IDS) to identify the anomaly behaviors in the network. The proposed DO-IDS used hybrid data optimization. They used Isolation Forest (iForest) in data sampling and Genetic Algorithm (GA) to optimize the sampling ratio. The RF and GA are used in feature selection to obtain the optimal subset. The DO-IDS is tested using the RF classifier with the UNSW-NB15 dataset; it scored an accuracy of 92.8% and a false alarm rate of 0.330.

Some of the studies used the CL approach to detect attacks. Gudla et al. [28] proposed a Deep Intelligent DDoS Detection Scheme (DI-ADS) for IoT applications. The model is implemented in the fog layer to predict the behavior of the fog devices. The scheme was implemented using Multilayer Perceptron (MLP) on a dataset for Software Defined System called DDoS-SDN that consists of three types of DDoS attacks. The results showed the accuracy of 97.84% and 1000 min training time. Priyadarshini et al. [29] proposed a source-based DDoS defense mechanism in fog and cloud environments. The defender module is deployed at the SDN controller to detect abnormal behavior in the network and transport layer. The detection model is implemented using LSTM on CTU-13 Botnet and ISCX 2012 IDS datasets. The accuracy of the model was 87.67% and 87.96% for CTU-13 Botnet and ISCX 2012 IDS, respectively. Zhang et al. [30] proposed a DDoS detection model based on Bidirectional LSTM (BiLSTM) at edge computing. The model achieves information extraction using the BiLSTM network and automatically learns the characteristics of the attack traffic in the original data traffic. The proposed model, called power IoT, was implemented in their data and achieved an accuracy of 95%, 250 min training time, and 520 MB communication overhead.

For distributed learning, little works addresses the security in IoMT using FL, SL, and FSL approaches. Verma et al. [31] proposed an FL Deep Intrusion Detection (FLDID) framework in smart industries. The proposed framework builds a collaborative model to detect DDoS attacks. They used CNN+LSTM+MLP in the detection model with the X-IIoTID dataset. The proposed framework outperforms the other approaches with 99.22% accuracy and 3042 s training time. Rey et al. [32] proposed an FL framework to detect malware in IoT devices. The N-BaIoT dataset is used and is designed for network traffic of multiple IoT devices. Two different DL models, MLP and Autoencoders (EA), are used for detection. The performance of the models is compared with the traditional approach, which shares its data with the server, and with the traditional approach, which does not share its data. The performance of the proposed framework scored an accuracy of 95% and 88 MB communication overhead. Alhasawi et al. [33] introduced a federated learning-based approach to detect DDoS attacks known as FL for decentralized DDoS Attack Detection (FL-DAD). The detection model used CNN to effectively identify DDoS attacks on the CICIDS2017 dataset. The model is compared with centralized detection methods, and it achieved 99.3% accuracy, 48.3 s training time, and 390.2 MB communication overhead with the maximum number of nodes.

Yu et al. [34] proposed a Pseudo-Client ATack (PCAT) based on SL to detect DDoS attacks with more challenging situations where the client model is unknown to the server. They investigated the inherent privacy leakage via the server model in SL, where the server model can easily steal the data. The server trains a small part from the dataset, about 0.1%–5%, for the same learning task. The performance is evaluated using the Deep Neural Network (DNN) model on the MINIST dataset with a two-part SL strategy, where the results scored an accuracy of 96.98%. Abuadbba et al. [35] examined the SL on the 1D CNN model to detect DDoS attacks. They tested the model with medical ECG data and added two mitigation techniques. The first technique adds more hidden layers to the client side. The second technique applies various privacy. The model achieved 98% accuracy and 20 min training time.

Khan et al. [36] performed the first empirical analysis of the SplitFed algorithm. They used SplitFed to design a strong detection model. The CIFAR10 and FEMNIST datasets are used with Alexnet and VGG11 models to perform the training process. The results of the Alexnet and VGG11 in the CIFAR10 dataset, the accuracy scored 62.4% and 73%, respectively. For the custom model in the FEMNIST dataset, the accuracy scored 81%. The communication overhead scored 150 and 200 MB for both datasets. Li et al. [37] proposed an attack detection approach FSL in real IoT scenarios. The proposed approach for the construction of global models of various time series in the context of data isolation has the ability to detect attacks. The experiment is applied to the CNN model and MNIST dataset. It achieved an accuracy of 93.56%, precision of 92.78%, recall of 93.19%, and F-measure of 92.98%.

All related works that are mentioned above are summarized in Table 1.

Table 1: Comparison between different detection methods

Ref./year	Method	Dataset	Finding	Limitations
[24] 2021	DRNN and ML	NSL-KDD	ACU: 99.76%	Accuracy alone cannot ensure the effectiveness of the proposed IDS in IoMT detection
[25] 2020	DBN	CICIDS2017	ACU: 96.67% Precision: 95.21% Recall: 97.34% F1: 0.97 DR: 97.31%	No consideration was given to training time and communication overhead
[26] 2019	ANN	CICIDS2017	ACU: 95.56% Training time: 150 min Communication overhead: 350 MB	Accuracy, training time, and communication overhead need to be improved
[27] 2019	DO-IDS with RF	UNSW-NB15	ACU: 92.8% FAR: 0.330	Accuracy needs to be improved
[28] 2022	MLP	DDoS-SDN	ACU: 97.84% Training time: 1000 min	Training time needs to be reduced

(Continued)

Table 1 (continued)

Ref./year	Method	Dataset	Finding	Limitations
[29] 2022	LSTM	CTU-13 Botnet and ISCX 2012 IDS	ACU: 87.67% and 87.96%	Accuracy needs to be improved
[30] 2022	BiLSTM	Power IoT	ACU: 95% Training time: 250 min Communication overhead: 520 MB	Results need to be improved
[31] 2022	FLDID	X-IIoTID	ACU: 99.22% Training time: 3042 s	Communication overhead was not considered
[32] 2022	FL with MLP and EA	N-BaIoT	ACU: 95% Communication overhead: 88 MB	There is still room for improvement
[33] 2024	FL-DAD with CNN	CICIDS2017	ACU: 99.3% Training time: 48.3 s Communication overhead: 390.2 MB	Communication overhead needs to be improved
[34] 2023	SL with DNN	MINIST	ACU: 96.98%	Accuracy alone is not enough to prove the effectiveness of the proposed IDS in IoMT detection
[35] 2020	SL with 1D-CNN	ECG dataset	ACU: 98% Training time: 20 min	Communication overhead was not considered, and accuracy can be improved
[36] 2022	Empirical SplitFed	CIFAR10 and FEMNIST	ACU: 73% and 81%	Accuracy needs to be improved. Training time and communication overhead are not considered
[37] 2023	FSL with CNN	MINIST	ACU: 93.56% Precision: 92.78% Recall: 93.19% F1: 92.98%	No consideration was given to training time and communication overhead

Note: • **(ACU)** for Accuracy in (%), **(F1)** for F-measure in (%), **(DR)** for Detection Rate in (%), and **(FAR)** for False Alarm Rate.

Three important criteria to improve IoMT networks are detection accuracy, latency, and communication overhead. Most existing works that address the security in IoMT based on ML, CL, and distributed learning suffer from low detection accuracy, high training time, and high communication overhead, which is not acceptable for real-time and latency-sensitive applications such as IoMT. In addition, most of the datasets are out of date and related to different types of security attacks; there is a lack of datasets related to DDoS attacks. To solve the problems of existing attack detection, we propose a distributed FSL (DFSL) system to detect DDoS attacks in the IoMT environment, supporting the

existing issues. An adaptive aggregation method based on the early stopping strategy is proposed to enhance detection accuracy and reduce training time. In addition, to improve the communication overhead, we propose a Multi-Node Selection (MNS) based BC-BN2 scheme with three selection metrics. The proposed system is evaluated using two modern DDoS datasets, CICDDoS2019 and LITNET-2020.

3 Problem Formulation

As mentioned above, IDS based on ML and CL has multiple challenges, including learning performance, training time, and communication overhead. The first Optimization Problem (*OP*) considers the learning performance of ML and CL, which needs optimization to maximize the detection accuracy. FL helps to enhance the learning of a statistical model at the server to generate a model with high accuracy with distributed datasets among multiple clients. Therefore, the server needs to minimize the global loss function.

***OP*1:** The first objective is to maximize detection accuracy by minimizing the global loss function $L(w)$ in FL according to Eq. (2) which is subject to massively distributed data constraint where the global data D is stored across a large number of clients N. The objective function can be modeled as

$$J(w) = min_{D_n}(L(w))$$

$$\textbf{Subject to} \sum_{n=1}^{N} D_n = D \tag{5}$$

Although the FL may enhance learning performance, it suffers from high training time, which takes a long time to compute. The term computation refers to the number of training epochs/rounds n_e and the number of model steps/epochs n_s that are assigned for each client n. By multiplying these two values, we get the total number of steps/rounds of the model for each client. The total training time in one communication round t is calculated as

$$T = max\left(T_w^n + (n_e \times n_s) \times T_s^n\right) \tag{6}$$

where T_w^n is the total time for two-way transmission between client n and server s and T_s^n is the total time for the client n to complete the model steps.

***OP*2:** The second objective is to reduce the total training time T with maximum CPU frequency constraint. The objective function can be modeled as

$$z = \min(T)$$

$$\textbf{Subject to } CPU^{min} \leq CPU_n \leq CPU^{max} \tag{7}$$

A distributed FSL (DFSL) system with an adaptive aggregation method based on the early stopping strategy is proposed to solve $OP1$ and O$P2$. Moreover, FSL suffers from high communication overhead, especially in SL training. For each round, the server should collect the smashed data from all clients to train the server-side model, and all clients must wait for gradients from the server to update its local model. We consider the communication overhead of client n at tth communication round can be calculated as

$$CO_t^n = B_t^n log_2\left(1 + \frac{P_t^n}{B_t^n}\right) \tag{8}$$

where B_t^n is the allocated bandwidth of client n at the t communication round, P_t^n is the transmission power of node n at t communication round.

***OP*3:** The last objective is to reduce communication overhead with maximum transmission power constraint. The objective function can be modeled as

$$u = min(CO_t^n)$$

$$\textbf{Subject to } 0 \leq \mathrm{P_t^n} \leq \mathrm{P^{max}} \tag{9}$$

A Multi-Node Selection (MNS) based BC-BN2 selection scheme is proposed to solve *OP*3. The proposed selection scheme selects the nodes with three selection metrics: the importance of the local update, channel quality, and the nearest distance from the server. Only the selected nodes are used for global aggregation to reduce communication overhead.

4 Proposed Distributed FSL System

This section describes the details of the proposed distribution FSL (DFSL) system. First, the channel model is illustrated. Then, the DFSL system procedure is introduced. Then, the MNS scheme is explained. Finally, model convergence and complexity analysis are described.

4.1 Channel Model

The channel model of the proposed distributed FSL (DFSL) is based on uplink massive MIMO technology. The massive MIMO technology has been used in recent years with 5G to achieve high spectral efficiency and throughput, which is needed for IoMT networks. As shown in Fig. 4, the channel model consists of multiple clients, a base station with multiple antennas, and one server. The clients are fog nodes represented as $N = \{n_1, n_2, \ldots, n_i\}$, where each fog node represents a fog server distributed in different areas. We assume that neither the nodes nor the server is attacked. We use a 16 × 16 MIMO antenna configuration where increasing the number of antennas helps avoid interference with the neighbor nodes. We suppose the uplink massive MIMO is equipped with M antennas at the base station and communicates with the set of fog nodes N simultaneously. If the signal is transmitted from the node to estimate the channel is $x \in C^N$, it is received at the base station during uplink transmission as

$$y = Hx + n_{uplink} \tag{10}$$

where $y \in C^M$ is the signal received, H is the channel vector between the node and base station, and elements of $H \in C^{M \times N}$are independent and distributed with unit variance and zero mean $H \sim CN(0, 1)$. The $n_{uplink} \in C^M$ is the additional interference from the multiple transmission and receiver noise. This interference is independent of the node signal x, but it can be dependent on the channel H as

$$n_{uplink} = n_{uplink-interference} + n_{noise} \tag{11}$$

4.2 DFSL Procedure

The proposed DFSL is based on the FSL approach to maximize detection accuracy and reduce training time. Fig. 5 shows the DFSL system framework. Because the IoMT devices are resource-constrained and limited in computations, our DFSL algorithm works on fog nodes which have high computation capabilities and can avoid latency in computations. The proposed system works in three phases including initialization and model splitting, local model training, and adaptive aggregation.

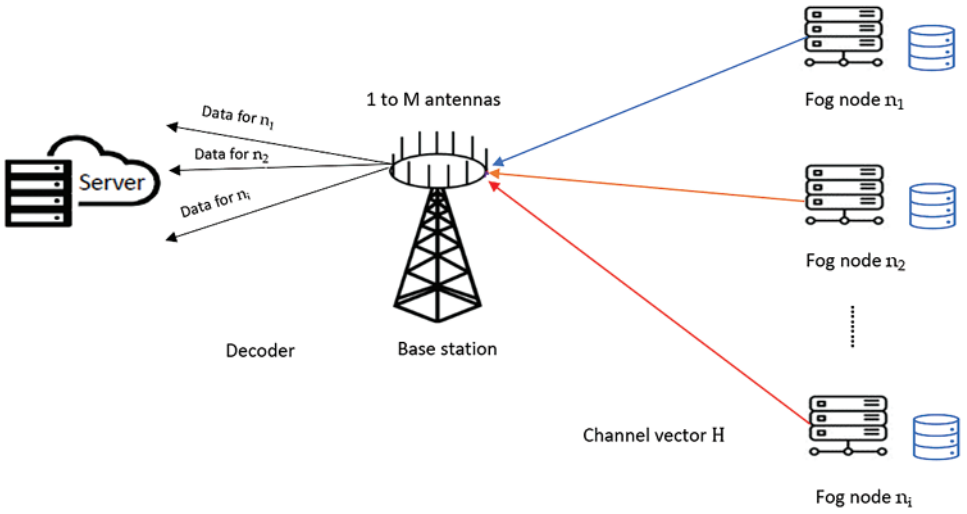

Figure 4: System channel model

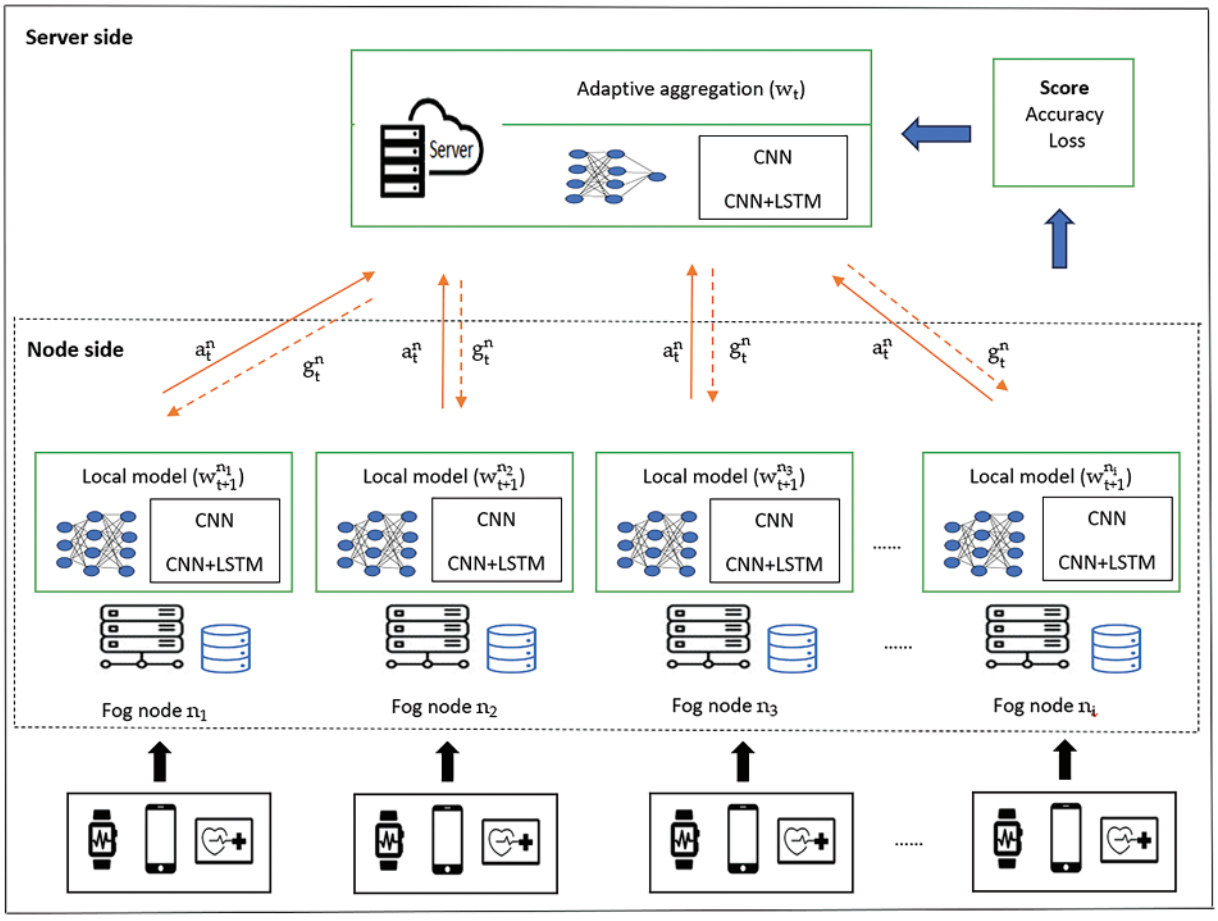

Figure 5: Proposed DFSL framework

4.2.1 Initialization and Model Splitting

The server randomly selects the nodes randomly and broadcasts the initial global parameter w_0 to all selected nodes. After receiving the initial parameter, the model is split between the node and server. Based on the two-part SL strategy, the CNN and CNN+LSTM models are split into two parts, as shown in Fig. 6. The first part runs on the node side, and the second runs on the server side. The CNN model is used with four convolutional layers (Conv), two max-pooling layers (Max-Polling), two dropout layers (Dropout), and three fully connected layers (FC). The first part consists of Conv 1 and Conv 2 operated with (convolution filters = 16, filter size = 3, and output shape = 16 × 196 feature matrix), Max-Pooling 1 with pooling size = 2, (Conv 3 and Conv 4) operated with (convolution filters = 32, filter size = 3, and output shape = 16 × 128 feature matrix), and max-pooling 2 with pooling size = 2. The second part has many layers, including FC 1, Dropout 1, FC 2, Dropout 2, and FC 3. One padding is applied before each convolution operation, and the Rectified Linear Unit (ReLU) is chosen as an activation function of hidden layers. Sigmoid is used to activate the last FC layer. For the CNN+LSTM model, the same layers with the same configuration add an LSTM layer after Max-Pooling 2 in the part that is operated with (filter size = 64 and the output shape = 64 × 128).

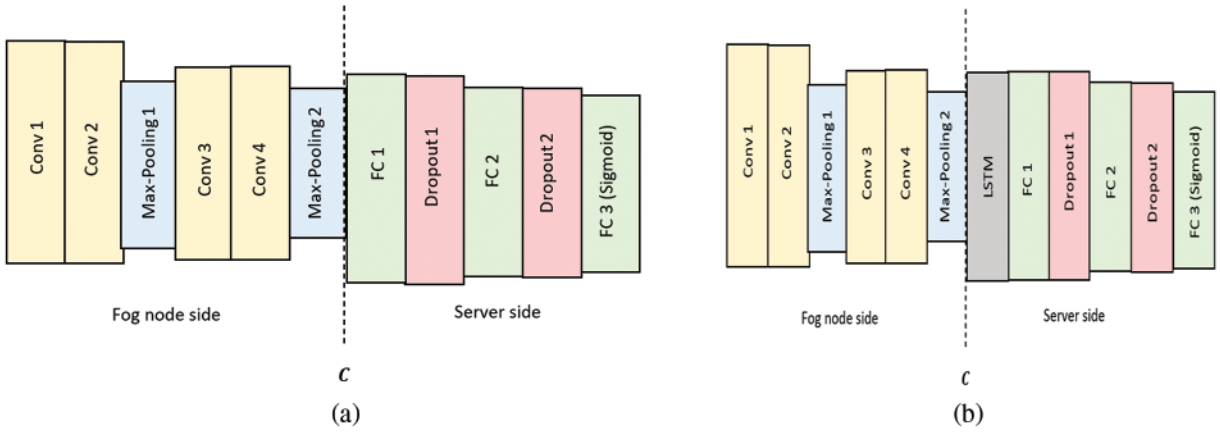

Figure 6: Two-part SL (a) CNN; (b) CNN+LSTM

4.2.2 Local Model Training

After model splitting, each node with its smashed data computes activation a_t^n up to its cut layer C with the server and updates its local model update. Now, the server receives the a_t^n and performs the forward propagation a_t^s from its split layer up to the last layer in the model and calculates the loss function $L_n\left(w_{t+1}^n\right)$. After that, the backpropagation is calculated and sent back to the node to update its model part. When the server receives the final update from the node, it calculates the backpropagation to the last layer and updates the model w_{t+1}^n. The backpropagation (gradient) and model update on the server side are calculated as

$$g_t^s = \nabla L\left(a_t^s,\ w_t^s\right) \tag{12}$$

$$g_t^n = \nabla L\left(a_t^n,\ w_t^n\right) \tag{13}$$

The local model updates on the node side are calculated as

$$w_{t+1}^n = w_t^n - \alpha_t \nabla g(w_t^n) \tag{14}$$

4.2.3 Adaptive Aggregation

An adaptive aggregation method is proposed to reduce training time and enhance detection accuracy. When training a model, the weights are the most important parameter for learning the model. The objective of the nodes' local training process is to find the weight that enhances the global model's accuracy on the local training sets. Therefore, the proposed adaptive aggregation adopts a computation amount controlled strategy that performs a certain number of model training steps per round to each node. This controlled strategy is based on the gap between the current local model's accuracy and loss on the training set, as well as the maximum achievable accuracy and minimum achievable loss.

Unlike FedAvg, which perform fixed computation (n_e and n_s) to the nodes in each communication round. The proposed adaptive aggregation method can assign specific computation amounts to each node based on the early stopping strategy. The early stopping strategy aims to prevent the training at the inflection point where the performance is not improved, reducing the overall training time. The early stopping strategy stops the training based on the classification score to enhance detection accuracy. This classification score includes the accuracy and loss scores. Accuracy and loss are important parameters that determine the model's behavior based on its score value. As a result, nodes with larger accuracy and loss gaps are needed to perform extra training rounds to obtain the optimal score. Conversely, nodes with smaller gaps are assigned fewer steps.

Our idea is that FedAvg's convergence can be improved by using the max score of accuracy and loss in the training set to smartly stop the training at a certain point and to set the values for n_e and n_s for each node and each round. By reducing the computational time, the convergence time will be optimized.

The adaptive aggregation works in three steps. First, the server determines the maximum score $score_{max}$ by computing the training accuracy for the local update at each round. Then, the accuracy value is subtracted from ($\mu \times loss$) as

$$score_{max} = w^n_{t+1}(acc) - w^n_{t+1}(\mu \times loss) \tag{15}$$

The μ is a hyperparameter used to balance accuracy and loss in the overall score; it is assigned to the default value $\mu = 0.5$.

Second, the current $\boldsymbol{score_{max}}$ is compared to the average score $\boldsymbol{score_{avg}}$ of the rest of the local updates. If the current $\boldsymbol{score_{max}}$ exceeds the $\boldsymbol{score_{avg}}$ value, the model is saved as best model $\boldsymbol{w^n_{t+1}}$ and the new score as best score $\boldsymbol{score_{b\_max}}$. Otherwise, it will not be saved as the best model. This can be expressed as

$$\boldsymbol{w^n_{t+1}} = \begin{cases} \overline{\boldsymbol{w}}^n_{t+1} & \boldsymbol{if\ score_{\max} > score_{avg}} \\ \boldsymbol{w^n_{t+1}} & \boldsymbol{if\ score_{\max} < score_{avg}} \end{cases} \tag{16}$$

Finally, the stopping counter $\boldsymbol{sc}$ is set to 0 when assigning the best model. Otherwise, $\boldsymbol{sc}$ is increased by one to record no improvement. When $\boldsymbol{sc} > \boldsymbol{patience}$ ($\boldsymbol{patience}$ is the number of epochs/rounds with no improvements after which training will be stopped; we set the $\boldsymbol{patience} = \mathbf{10}$ rounds), the process stops, and the local update is starting aggregation.

Subsequently, the server aggregates the selected local updates to obtain the global model updates. The global model aggregation is calculated as

$$\boldsymbol{w_t} = \boldsymbol{w_t} - \sum\nolimits_{n=1}^{n} \nabla \boldsymbol{L_n}(\overline{\boldsymbol{w}}^n_t) \tag{17}$$

The workflow of the proposed DFSL system is described in Algorithm 1.

Algorithm 1: Proposed DFSL

Require: Node $n \in N$, initial global parameter w_0, local training data D_n of node n, activation a_t^s up to the last layer of the server at round t, activation a_t^n up to cut layer C of the node n at round t, gradients g_t^s of server s at round t, gradients g_t^n of node n at round t, maximum score $score_{max}$, stopping counter sc, epochs e, training steps s, number of epochs/round assigned to the node n_e, and number of model steps/epochs n_s,

1. $score_{max} = 0$
2. $sc = 0$
3. $n_e = e_{max}$, $n_s = s_{max}$

// Initialization

4. Server initializes w_0
5. **for** each round $t = 0, 1, \ldots, T$ **do**
6. Select a random set N of n nodes
7. Server sends w_0 to all n nodes participating in the round t

// Local model update

8. Each node $n \in N$ containing the smashed data, performs forward propagation up to its cut layer C
9. Each node $n \in N$ sends the activation a_t^n of its cut layer C to the server
10. Server that received the activation a_t^n from node n, performs forward propagation a_t^s from its cut layer to the final layer and compute loss value $L_n\left(w_{t+1}^n\right)$
11. Server performs backward propagation g_t^s in (12) and send the gradient of its loss value to the node n
12. Each node $n \in N$ that received the gradient from the server, performs the backward propagation g_t^n in (13) and updates its local model update w_{t+1}^n in (14)

// adaptive aggregation

13. Server computes $score_{max}\left(w_{t+1}^n\right)$ in (15)
14. **if** $score_{max} > score_{avg}$
15. $w_{t+1}^n = \overline{w}_{t+1}^n$
16. $score_{max} = score_{b\_max}$
17. $sc = 0$
18. **else**
19. $sc = sc + 1$
20. **end if**
21. **if** $sc > patience$ **then**
22. Process stop and server aggregate the local update as $w_t = w_t - \sum_{n=1}^{n} \nabla L_n(\overline{w}_t^n)$
23. **end for**
24. **return** w_t

4.3 Multi-Node Selection

The fog nodes are connected to a wireless network via a radio communication link with constrained bandwidth. Although the DFSL reduces communication overhead by sharing model parameters rather than raw training data, the transmission of complex models from many nodes still generates much traffic. The goal is to reduce the number of nodes participating in each communication round to reduce overall communication overhead. One common technique to reduce the communication overhead in a distributed network is the node selection technique.

Based on the BC-BN2 which selects the nodes based on two metrics including the importance of local update and channel quality, the proposed MNS scheme selects the nodes based on three metrics. These metrics include the importance of local update, channel quality, and the nearest distance from the server. Selecting nodes with important local update helps to improve the model performance and get an accurate model. On the other hand, selecting the node with good channel quality and the nearest distance from the server improves communication and reduces communication time consumption.

For selecting nodes with the importance of local update, the nodes need to compare their local updates with the global update to determine the importance of its update in each communication round. The issue here is we cannot know if the global update is relevant or not until all local updates are aggregated. So, we can use the previous global update for comparison. To ensure the correct relevance between the previous global update and the current local update, we take the difference between two sequential global updates because the model training does not always have constant model convergence in each communication round. The normalized difference between two previous sequential global updates is measured as

$$\Delta w_t = \frac{\|w_{t-1} - w_t\|}{\|w_t\|} \tag{18}$$

where $\|\cdot\|$ is the Euclidean norm for a given vector, w_t and w_{t-1} are two previous sequential global updates generated in rounds t and $t-1$, respectively.

A larger normalized difference means a larger divergence between the two updates. To ensure no large divergence between the two updates, the CNN and CNN+LSTM models are trained and measured for the cumulative distribution of the normalized divergence of the two global updates. As shown in Fig. 7, we see that for CNN, more than 98% of the difference is less than 0.3, where the maximum difference is 0.5. For CNN+LSTM, more than 99% has a difference of less than 0.2, where the maximum difference is 0.4. We can ensure that the previous global update can be used for the current global update. This prediction does not require more communications because each node maintains the global update made in the previous round. The current local update is compared to the previous round's global update. The relevance between the local and global update is determined by computing the number of parameters with the same sign (gradients) in the two updates and normalizing the result by the total number of parameters. The relevance between the two updates is formulated as

$$\mathrm{r}(w^n_{t+1}, \Delta w_t) = \begin{cases} 1 & if\ sign\left(w^n_{t+1}\right) = sign\left(\Delta w_t\right) \\ 0 & Otherwise \end{cases} \tag{19}$$

where w^n_{t+1} and Δw_t are the current local update and relevant global update, respectively. If $r(w^n_{t+1}, \Delta w_t) = 1$, this means the w^n_{t+1} and Δw_t have the same parameters sign. If $r(w^n_{t+1}, \Delta w_t) = 0$, this means the w^n_{t+1} and Δw_t have different parameter signs.

Now, the nodes that have the important local update are denoted as $\overline{N} = \{\overline{n}_1, \overline{n}_2, \ldots, \overline{n}_i\}$. After determining the nodes with the important local update, each relevant node $\overline{n}$ is measured in its channel quality. Unlike the BC-BN2, which uses the channel gain to test the channel quality, our MNS measures the channel quality by its capacity. The channel capacity in IoMT is very important to avoid latency. The channel capacity in MIMO communication is calculated based on the Shannon theorem as

$$Capacity = B\, log_2\,(1 + SNR) \tag{20}$$

where B is the bandwidth of the channel, and SNR is the Signal-to-Noise Ratio.

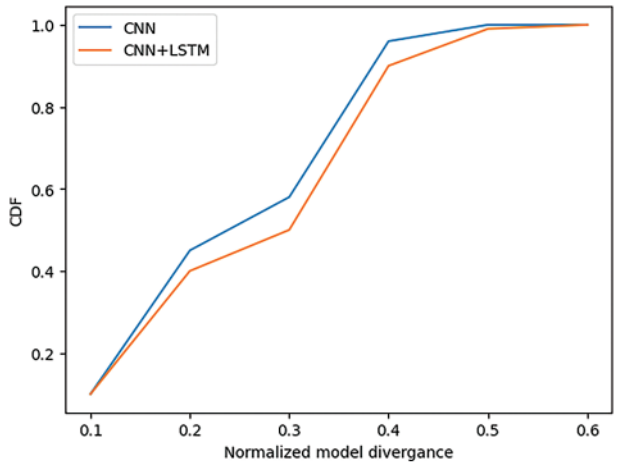

Figure 7: Cumulative distribution of CNN and CNN+LSTM

When the SNR is close to 0, the term $(1 + SNR)$ is close to 1, and the logarithm term $\log_2(1 + SNR)$ is close to 0. As a result, the Capacity is also close to 0 bps, regardless of the available bandwidth B. This means that on an extremely noisy channel with an SNR close to 0, the channel capacity is effectively zero. No meaningful information can be transmitted through the channel due to the overwhelming noise. The channel is essentially unusable for reliable data transmission. Therefore, node $\overline{n}$ that has $Capacity \geq 1$ is considered good channel quality and starts to measure its distance from the server.

The nearest distance between the node $\overline{n}$ and server s is determined using the K-Nearest Neighbor (KNN) algorithm. The KNN algorithm works in three steps. First, select the value of k, which represents the number of nearest neighbors that need to be considered. It differs based on the total number of nodes. Second, calculating the distance using Euclidean distance as

$$dis(\overline{n}, s) = \left(d\left(x_i - x_j\right) + d\left(y_i - y_j\right)\right)^2 \tag{21}$$

where (x_i, y_i) are the coordinates of the node point, (x_j, y_j) are the coordinates of the server point, and d is the distance between (x_i, y_i), and (x_j, y_j) in a two-dimensional plane.

Finally, finding the nearest neighbor nodes, the k points with the smallest distance to the server are the nearest neighbors.

Because the first round doesn't have a previous global update, the nodes are selected randomly. The proposed algorithm starts from the second round, only node $\overline{n}$ with good channel quality and the nearest distance from the server is selected for aggregation. The procedure of the MNS scheme is described in Algorithm 2.

Algorithm 2: MNS algorithm

Require: Node $n \in N$, current local update w_{t+1}^n, and previous two global updates w_t at round t and w_{t-1} at round $t-1$, B, and SNR.

1. **for** each round $t = 1, 2, \ldots, T$ **do**

// Importance local update

2. **for** each node $n \in N$ in parallel **do**

(Continued)

Algorithm 2 (continued)

3. Compute the difference between w_{t-1} and w_t in (18)
4. Check the relevance between w_{t+1}^n and Δw_t
5. **if** $r(w_{t+1}^n, \Delta w_t) = 1$ **then**
6. Define w_{t+1}^n as a relevant update on a relevant node $\overline{n}$
7. **else**
8. Define w_{t+1}^n as an irrelevant update
9. **end if**
10. **end for**

// Channel quality and nearest distance

11. **for** each $\overline{n} \in N$ in parallel **do**
12. Compute capacity in (20)
13. **if** $Capacity(\overline{n}) \geq 1$
14. Compute Euclidean distance $(\overline{n}, s)$ in (21)
15. Select the node and upload the relevant w_{t+1}^n for aggregation
16. **else**
17. Ignore the node
18. **end if**
19. **end for**
20. **end for**

4.4 Convergence Analysis

In this section, the fundamental convergence analysis that affects the node side and server side on its training is performed. First, the technical assumptions are made. Then, the convergence result is presented.

A. Assumptions

First, we make the following four assumptions for convergence analysis.

Assumption 1. (ℓ-*smooth*) Each local loss function $L_n(w)$ is differentiable and ℓ-smooth. For all w and w^*, we have

$$L_n(w) \leq L_n(w^*) + (\nabla L_n(w^*),\ w - w^*) + \frac{l}{2}\|w - w^*\|^2 \tag{22}$$

Assumption 2. (μ-*strongly convex*). Let $L_1, \ldots, L_n$ are μ-strongly convex. For all w and w^*

$$L_n(w) \geq L_n(w^*) + (\nabla L_n(w^*),\ w - w^*) + \frac{\mu}{2}\|w - w^*\|^2 \tag{23}$$

Assumption 3. (*Unbiased and bounded stochastic gradient with bounded variance*). Let ξ_t^n represent the random sample dataset from node n, and $g_n(w_t^n)$ of $L_n\left(w_t^n\right)$ ae unbiased with the variance bounded by σ_n^2 that measures the level of stochasticity

$$E_{\xi_t^n \in D_n}\left\|g_n(w_t^n, \xi_t^n) - \nabla L_n(w_t^n)\right\|^2 \leq \sigma_n^2 \tag{24}$$

Further, the expected squared norm is bounded by G^2

$$E_{\xi_t^n \in D_n}\left\|g_n(w_t^n,\ \xi_t^n)\right\|^2 \leq G^2 \tag{25}$$

Assumption 4. (*Partial participation*). As discussed before, only subset of nodes is selected to participate in model aggregation in each communication round t. Let $\overline{N}$ be the selected subset of nodes. The probability of each node n to be selected for model aggregation is $\mathbb{P} = \dfrac{\overline{\mathrm{N}}}{N}$. Assuming the data is balanced and non-IID, thus the model aggregation at the server is represented as

$$w_t = w_t - \frac{\overline{N}}{N} \sum\nolimits_{n=1}^{n} \nabla L_n(w_t^n) \tag{26}$$

B. Convergence result

Based on the previous assumptions, the following results are obtained.

Theorem 1. Based on **Assumptions 1–4**, we assume $\varrho = \dfrac{2}{\mu}$ with $\iota = \dfrac{4\ell}{\mu}$ and $\mathbb{N} = \dfrac{\ell}{\mu}$, then, the DFSL algorithm with $\overline{N}$ nodes selected for participation satisfies

$$\mathrm{E}[\mathrm{L}(\mathrm{w_T})] - \mathrm{L}^* \leq \frac{\mathrm{N}}{\iota + \mathrm{T} - 1} \text{ł} \left(\frac{2\mathrm{W}}{\mu} + \frac{\mu\iota}{2} \mathrm{E} \left\| \mathrm{w} - \mathrm{w}^* \right\|^2 \right) \tag{27}$$

where $W = \sum_{n=1}^{N} \sigma_n^2 + 6l + 8G^2 \left(1 - \dfrac{\overline{N}}{N} \right)$

Proof. By defining $\nabla L_n = \mathbb{E} \left\| w - w^* \right\|^2$ and setting $\nabla L_n \leq \dfrac{w}{\iota + t}$ with $\varrho \dfrac{2}{\mu}$ and $\iota > 0$, we have

$$w = max \left\{ \frac{\varrho^2 W}{\varrho\mu - 1}, \, (\iota + 1) \mathbb{E} \left\| w_1 - w^* \right\| \right\} \tag{28}$$

$$\leq \frac{\varrho^2 W}{\varrho\mu - 1}, \, (\iota + 1) \, \mathbb{E} \left\| w_1 - w^* \right\|$$

$$\leq \frac{4W}{\mu^2}, \, (\iota + 1) \, E \left\| w_1 - w^* \right\|$$

Then by ℓ-*smooth* of $L(\cdot)$ and let $\mathbb{N} = \frac{\iota}{\mu}$,

$$E\left[L\left(w_t\right)\right] - L^* \leq \frac{\iota}{2} \nabla_t \leq \frac{\iota}{2} \frac{W}{2t + \iota} \leq \frac{N}{t + \iota} \left(\frac{2W}{\mu} + \frac{\mu(\iota + 1)}{2} E \left\| w_1 - w^* \right\| \right) \tag{29}$$

The convergence of the model is stable without extra communication overhead. Moreover, the convergence performance doesn't affect the number of selected nodes, but the speed of convergence increases with the increasing number of selected nodes. The convergence rate achieves an order of $O(1/T)$.

4.5 Complexity Analysis

Suppose the total number of nodes is N, each with a local data size S, model parameter size W, and Q nodes selected for participation for model aggregation in each communication round, with each node conducting R local rounds.

1. Node-side computation complexity: The computation complexity of each local update is based on the number of samples updated in each round and the number of communication rounds. The time complexity of local updates for each node is $O(RS)$.
2. Server-side computation complexity: The computation on the server side is divided into model training, model aggregation, and weight update. For model training, as on the node side, the time complexity of local updates for each node is $O(RS)$. For model aggregation, the complexity is based on the number of nodes participating in aggregation and the size of model parameters. Therefore, the time complexity of model aggregation is $O(QW)$. For weight update, the weight at the server is recalculated depending on the sampling count of all nodes. The time complexity is $O(N)$.
3. Communication overhead analysis: Four communications between the node and server, including forward propagation transmission from the node to the server, backward propagation transmission from server to the node, aggregation from node to the server, and global update from server to the nodes. The time complexity for the first three communications is $O(QW)$, and for the aggregation, it is $O(NW)$. The total communication overhead is $O(QW) + (NW)$.

5 Data Description and Preprocessing

This section describes the datasets used in implementation and data preprocessing, including different techniques.

5.1 Datasets Description

The CICDDoS2019 [38] and LITNET-2020 [39] datasets are used to direct this work. These datasets are designed for the types of DDoS attacks. The details of the CICDDoS2019 and LITET-2020 datasets are as follows.

5.1.1 CICDDoS2019

A new network intrusion dataset has been selected named CICDDoS2019, designed in 2019; the dataset consists of a large number of various types of DDoS attacks that can be carried out using Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) at the application layer. The taxonomy of the DDoS attacks in the dataset is implemented in terms of reflection-based and exploitation-based attacks. The dataset is organized in two days for training and testing evaluation. The training set contains 12 different types, and the testing set contains 7 types of DDoS attacks. The dataset has more than 80 features, which were extracted using the CICFlowMeter tool. The distribution of the CICDDoS2019 data is described in Table 2.

5.1.2 LITNET-2020

The LITNET-2020 NetFlow is a new dataset obtained from a real-world academic network. The network topology of the dataset consists of two parts: senders and collectors. The senders are Cisco routers and Fortige (FG-1500D) firewalls, which were utilized to process NetFlow data and send it to the collectors. The collector is a server with appropriate software that is responsible for receiving, storing, and filtering data. The LITNET-2020 dataset represented real-world examples of normal and attack traffic with 12 types of DDoS attacks. The total flow of the data samples is 45,492,310 flows, categorized into normal data with 45,330,333 flows and attack data with 5,328,934 flows. The LITNET-2020 dataset consists of 85 network features divided into 49 features that are specified to the NetFlow V9 protocol, 15 features are supplemented by the data extender, 19 features are offered to

recognize attack types, and 2 additional features to determine the attack type and normal traffic. The distribution of attacks in the dataset. The distribution of LITNET-2020 data is described in Table 3.

Table 2: Data distribution of CICDDoS2019

Types of attack	No. of flows
Benign	56,863
DNS	5,071,011
LDAP	2,179,930
MSSQL	4,522,492
NetBIOS	4,093,279
NTP	1,202,642
SNMP	5,159,870
SSDP	2,610,611
SYN	1,582,289
TFTP	20,082,580
UDP	3,134,645
UDP-Lag	366,461
WbDDoS	439
Total	911,963,349

Table 3: Data distribution of LITNET-2020

Type of attack	No. of flows	No. of attack flows
Smurf	3,994,426	59,479
ICMP-flood	3,863,655	11,628
UDP-flood	606,814	93,583
TCP SYN-flood	14,608,678	3,725,838
HTTP-flood	3,963,168	22,959
LAND attack	3,569,838	52,417
Blaster worm	2,585,573	24,291
Code red worm	5,082,952	1,255,702
Spam bot's detection	1,153,020	747
Reaper worm	4,377,656	1176
Scanning/Spread	6687	6232
Packet fragmentation	1,244,866	477
Total	45,330,333	5,328,934

5.2 Data Preprocessing

In data preprocessing, we divide the task into five steps, including features mapping, computing the missing value, feature selections, data normalization, and data imbalance.

5.2.1 Features Mapping

The data features in IoMT don't contain only numeric values. So, a mapping technique is required to convert the categorical values to numeric values. One-Hot Encoding (OHE) technique converts the categorical features to integer format to be more expressive. The OHE transforms a single value with n features and d distinct values to d binary values with n features. Each feature indicates the presence of 1 or the absence of 0 of the dichotomous binary value. Using the OHE technique helps the deep learning models deal with the numeric values effectively.

5.2.2 Computing Missing Value

The datasets always consist of many missing values, which come from different reasons such as data failure or corruption during its recording, unreliable data transmission, and system maintenance and storage issues. The missing value is calculated using a linear interpolation method as

$$\mathbb{E}F\left(x_i\right) = \begin{cases} \sum_{k=i-5}^{i+5} P_k U_k \times Average_{local} & if\ x_i \in NaN \\ x_i & if\ x_i \notin NaN \end{cases} \tag{30}$$

The $Average_{local}$ computed as

$$Average_{local} = \frac{1}{10} \times \sum\nolimits_{i-5}^{i+5} f\left(x_i\right) \tag{31}$$

where x_i represents the data value, if x_i is a null or non-numeric value, it is described as NaN. The P_k is the imputed data point between 0 and 1, and it is chosen to be 0.10 because the smallest value gives better performance. The U_k represents a binary value based on the threshold value of k. The threshold value is computed as

$$U_k = \begin{cases} 0 & if \quad x_k < Average_{local} \\ 1 & if \quad x_k \geq Average_{local} \end{cases} \tag{32}$$

5.2.3 Features Selection

Feature selection improves the quality of prediction during the selection of feature inputs. Feature selection is the process of converting the set of features into a subset that contains the features that are important to solve the detection problem and discard the unneeded features. In this paper, a Mutual Information (MI) technique is used for features selection. The criteria of selecting features are the dependency where the features that have high dependencies between them are noted as best features. The MI procedure is shown in the following steps:

Step 1: From the training set sample, the input-output variables can be represented as X and Y where $S = \{X,\ Y\}$.

Step 2: The MI value between input and output variables $MI(X_i,\ Y)$ measures the dependencies between them as

$$MI\left(X,\ Y\right) = \sum\nolimits_{x \in X} \sum\nolimits_{y \in Y} P\left(x,\ y\right) log \frac{P(x,\ y)}{P\left(x\right) P(y)} \tag{33}$$

where $P\left(x\right)$ and $P\left(y\right)$ are the marginal distributions, and $P(x,\ y)$ is the joint probability of X and Y.

Step 3: Remove the values which are less than other values and maintain the remaining values in vector M. The input-output variables represented as S_{new}. The S_{new} is sorting in descending order according to MI value from vector M.

Step 4: Define the number of selected features represented as f. Select the first f input-output variables from S_{new} as the selected feature to form set S_f.

Step 5: Select the variable with the highest MI value as the first variable in S. Select *jth* variables ($1 \leq j \leq f$) from S_f and calculate the $MI(S, X_j)$ values. The threshold is represented as α. If $MI\left(S, X_j\right) \leq \alpha$, $S = S \cup X_j$. Otherwise, X_j will have less dependency and needs to be removed.

Step 6: Repeat Steps 4 and 5 until all variables are chosen.

5.2.4 Normalization

Normalization is a scaling method used to convert all features into a common scale. The minimum and maximum values range of the continuous feature in CICDDoS2019 and LITNET-2020 are different. A Min-Max is a popular method used to facilitate arithmetic processing, and the range of values of each feature is uniformly linearly mapped in the range between 0 and 1. The Min-Max scaling method can be defined as

$$F_{scal} = \frac{F - F_{min}}{F_{max} - F_{min}} \tag{34}$$

where F_{scal} represents the scaling data point, F_{min}, and F_{max} represent the minimum and maximum value of the input feature F.

5.2.5 Data Imbalance

The CICDDoS2019 and LITNET-2020s datasets are not perfectly balanced. The imbalanced dataset is solved using a Synthetic Minority Over-sampling Technique (SMOTE) [40]. SMOTE is an over-sampling technique used to avoid overfitting; it allows the generation of synthetic samples for the minority categories. It is based on the KNN algorithm, where it takes a sample from the dataset and considers its nearest neighbors in the feature map. If (x_1, x_2) is an example of a minority class and if its nearest neighbor is chosen as $(\acute{x_1}, \acute{x_2})$, then the data point is synthesized

$$(X_1, X_2) = (x_1, x_2) + random\,(0, 1) \times \Delta \tag{35}$$

where $\Delta = \left\{\left(\acute{x_1} - x_1\right), \left(\acute{x_2} - x_2\right)\right\}$ and random (0, 1) represent a random value in range [0, 1].

6 Experiments and Results

This section describes the experimental setup and the comparative analysis of the proposed DFSL system with baseline distributed algorithms such as FedAvg, Vanilla SL, and SplitFed with different experiments.

6.1 Experimental Setup

The experiment in this work is conducted with the following practical setting: Python version 3.7.3 and PyTorch library version 1.2 with Anaconda and CUDA version 10.1. We use a Dell laptop with CPU i5-1235U, 64-bit, GPU GTX 1050, and OS (Windows 11). The system consists of 10 fog nodes distributed at different distances from the server. We assume all fog nodes have the same computing capabilities. We use the 80:20 ratio (80% for the training set and 20% for the testing set). Using a high training set helps to ensure that the model is trained on enough datasets to capture the patterns and relationships in the data, and to learn more complex patterns. The tasks are assumed to be independent of each other. The parameter settings that are used in implementation are described in Table 4.

Table 4: Parameter settings

Parameter	Value
Parameters of channel model	
No. of nodes N	10
No. of BS antennas M	256
Transmission power P_n	24 dBm
Bandwidth B	40 MHz
Data rate (each node)	10 Mbps
CPU frequency CPU_n	2.5 GHz
Hyper-parameters of CNN and CNN+LSTM	
Learning rate	0.01
Communication round T	10
Batch size	128
Epochs	100
Dropout	0.1
Optimizer	Adam

6.2 Performance Analysis

The proposed DFSL is compared with a baseline of distributed learning algorithms such as FedAvg, Vanilla SL, and SplitFed. The experiment is implemented with two datasets (CICDDoS2019 and LITNET-2020) on CNN and CNN+LSTM models to test the effectiveness of the proposed DFSL. The learning performance is implemented and compared based on four scenarios. First, the training loss and accuracy are tested with different cut layers. Second, the training time and communication overhead are tested with different numbers of nodes. Third, the adaptive aggregation of the DFSL will be compared with different aggregation methods. Finally, the proposed MNS scheme will be compared with different selection schemes.

6.2.1 Experiment 1: Performance Analysis of the DFSL with Different Cut Layers

The first experiment evaluates the proposed DFSL with different metrics such as loss, accuracy, and detection rate with different cut layers. Changing the cut layer point plays a huge role in the model's effectiveness to get better results. We consider two types of model splitting represented by $C = \{C_1, C_2\}$, where C_2 means more layers assigned to the nodes. A smaller C should lead to a better performance because it corresponds to a larger part of the model server side, which better mitigates the non-IID issue. However, the results show that as C_2 increases (i.e., more layers at the node side), the model performance increases. The impact of the cut layer becomes more significant, under which a larger C is more beneficial. To understand the reason behind this, we considered the evaluation with 10 nodes waiting for *patience* = 10 rounds. We observe that the loss generally decreases, and accuracy and detection rate increase in C_2, meaning a larger C leads to better model training. Also, the performance of CNN+LSTM is better than CNN, where adding LSTM layers positively affects the performance.

As shown in Fig. 8, the loss of DFSL outperforms FedAvg, Vanilla SL, and SplitFed with 0.07 and 0.02 at C_2 in CICDDoS2019, and 0.05 and 0.01 in LITNET-2020 for CNN and CNN+LSTM, respectively. As shown in Fig. 9, the accuracy of DFSL outperforms FedAvg, Vanilla SL, and SplitFed with 98.99% and 99.70% at C_2 in CICDDoS2019 and 99.50% and 99.87% in LITNET-2020 for CNN and CNN+LSTM, respectively. For detection rate, the DFSL outperforms the baseline algorithms with 98.14% and 99.60% at C_2 in CICDDoS2019 and 98.89% and 99.60% in LITNET-2020 for CNN and CNN+LSTM, respectively. This improvement in loss and accuracy can be attributed to the adaptive aggregation method with predetermined the maximum loss and accuracy score.

In contrast, there is no big variation between FedAvg and SplitFed, as both apply the same aggregation strategy. For Vanilla SL, it scored the worst values in terms of loss, accuracy, and detection rate. It is very sensitive to data distribution. In fact, for both FedAvg and Vanilla SL, some knowledge forgetting while learning is evident when trained in non-IID settings.

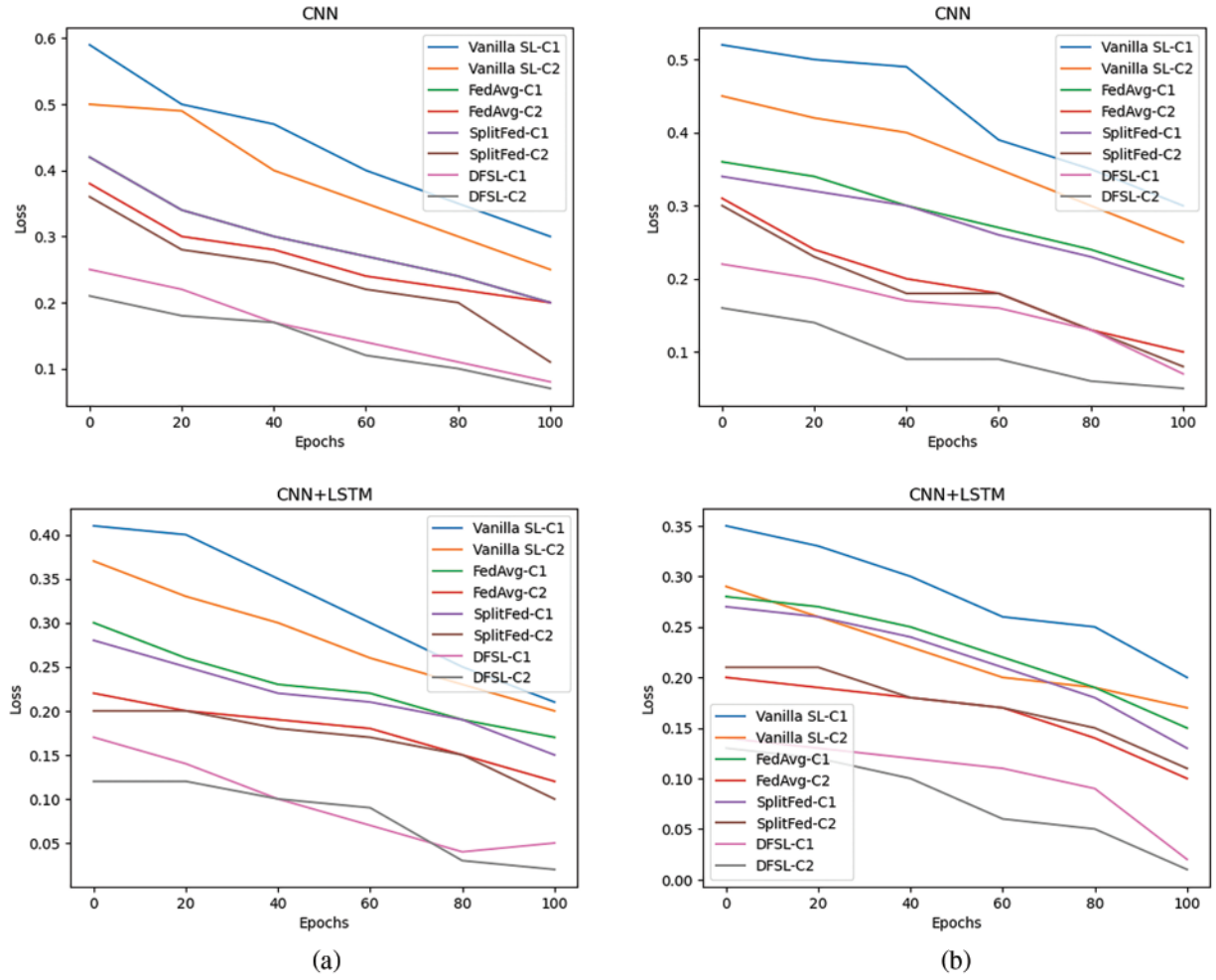

Figure 8: Loss performance of CNN and CNN+LSTM (a) CICDDoS2019; (b) LITNET-2020

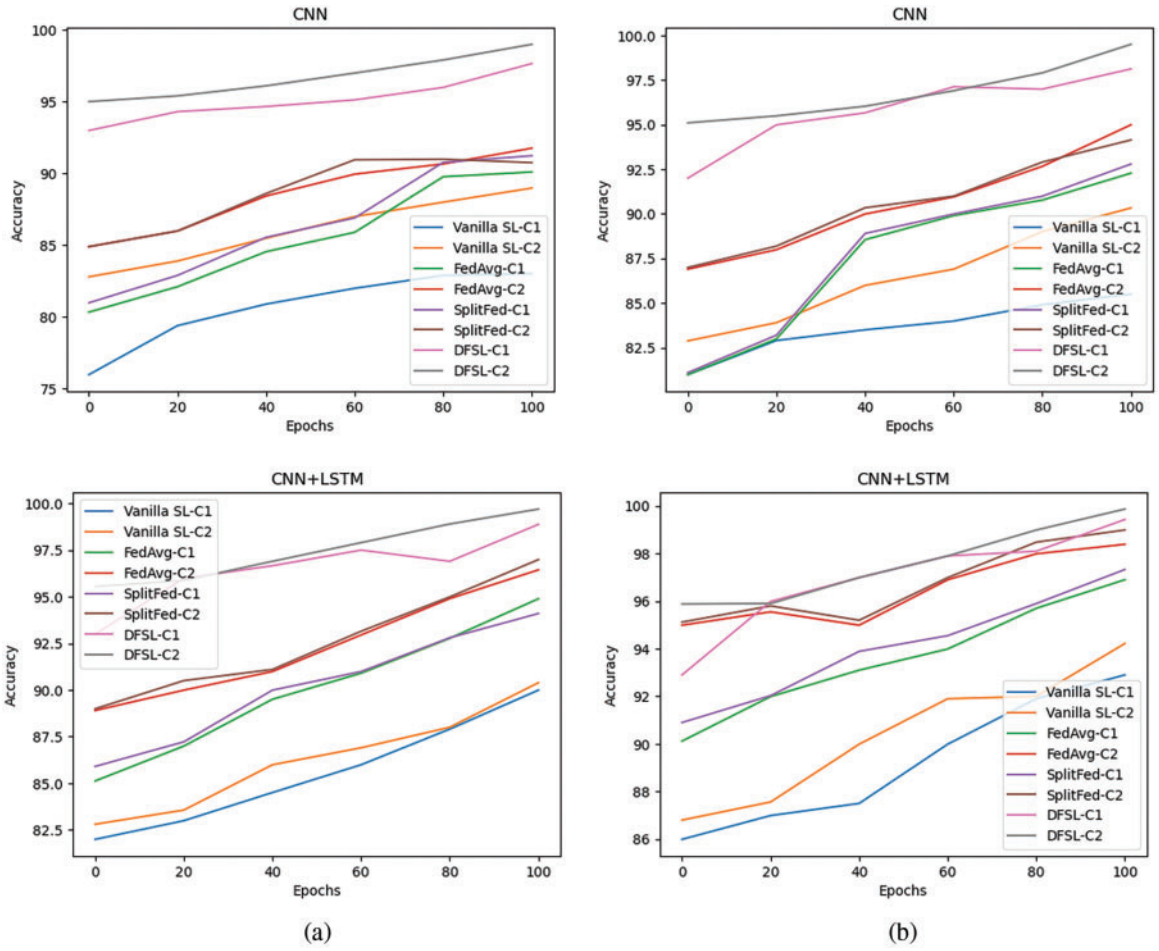

Figure 9: Accuracy performance of CNN and CNN+LSTM (a) CICDDoS2019; (b) LITNET-2020

When comparing the DFSL with the other approaches, we note that our DFSL has more hyper-parameters to tune when the cut layer is large. We conclude that a carefully specified way of determining the maximum accuracy and loss score and comparing it with average scores for each local update can benefit the most when applied in the DFSL system.

As shown in Tables 5 and 6, we observe that the proposed DFSL can detect DDoS attacks effectively enhancing learning performance. Based on these results, using the C_2 version of our model has a better performance for detection.

Table 5: Classification results in CICDDoS2019

Algorithm	CNN (C_1)			CNN (C_2)			CNN+LSTM (C_1)			CNN+LSTM (C_2)		
	Loss	ACU (%)	DR (%)	Loss	ACU (%)	DR (%)	Loss	ACU (%)	DR (%)	Loss	ACU (%)	DR (%)
Vanilla SL	0.30	83.02	82.90	0.25	88.98	88.22	0.22	89.99	89.54	0.20	90.39	89.76
FedAvg	0.22	90.10	89.54	0.12	91.76	90.99	0.17	93.89	93.10	0.12	96.44	96.13
SplitFed	0.20	90.75	89.88	0.11	92.23	91.65	0.15	94.10	92.99	0.10	96.99	96.56
DFSL	0.08	97.65	96.48	0.07	98.99	98.14	0.05	98.88	98.20	0.02	99.70	99.30

Table 6: Classification results in LITNET-2020

Algorithm	CNN (C_1)			CNN (C_2)			CNN+LSTM (C_1)			CNN+LSTM (C_2)		
	Loss	ACU (%)	DR (%)	Loss	ACU (%)	DR (%)	Loss	ACU (%)	DR (%)	Loss	ACU (%)	DR (%)
Vanilla SL	0.29	85.50	84.99	0.24	90.33	89.20	0.20	92.90	92.51	0.17	94.22	93.70
FedAvg	0.22	92.28	90.89	0.10	93.99	93.54	0.15	96.90	95.88	0.11	98.39	96.99
SplitFed	0.19	92.79	92.18	0.08	94.14	93.65	0.13	97.33	96.49	0.10	98.99	98.32
DFSL	0.07	98.12	97.40	0.05	99.50	98.89	0.02	99.43	98.66	0.01	99.87	99.60

6.2.2 Experiment 2: Performance Analysis of the DFSL with Different Number of Nodes

In this experiment, the training time and communication overhead are tested. We test the training time and communication overhead with different numbers of nodes, including N = 10, 30, and 50 nodes. In Vanilla SL, only one node takes the whole bandwidth in each round because of its sequential training. In contrast, FedAvg, SplitFed, and DFSL are trained in parallel, so all selected nodes will share the whole bandwidth in each round. The performance was assessed on testing sets following the final communication round T. We observe that the increasing number of nodes increases the training time and communication overhead. In addition, the performance of CNN is better than CNN+LSTM because the model size negatively affects the training time and communication overhead, where a large model size needs to transmit more model parameters. So, the following results reflect the CNN result values.

Fig. 10 shows the training time performance in CICDDoS2019 and LITNET-2020. The FedAvg has the highest training time compared with Vanilla SL, SplitFed, and DFSL, with 60 and 80 min in CICDDoS2019 and LITNET-2020, respectively, when the number of nodes is 10. The reason behind this is that in FedAvg, the allocated bandwidth to each node is decreasing. SplitFed achieves the highest training time after FedAvg with 60 and 80 min in CICDDoS2019 and LITNET-2020, respectively, when the number of nodes is 10. The Vanilla SL achieves acceptable results with 47 and 55 min in CICDDoS2019 and LITNET-2020, respectively. Compared to FedAvg, Vanilla SL, and SplitFed, DFSL spends less training time. The DFSL dynamically tunes the amount of computation assigned to the selected nodes at each round of training, resulting in a significant reduction in the overall training time per round. It achieves a total training time of 18 and 38 min in both datasets.

Fig. 11 shows the communication overhead in CICDDoS2019 and LITNET-2020. In terms of communication overhead, the performance degrades with increasing the number of nodes. When the nodes are 10, the DFSL has the lowest communication overhead with 25 and 31 MB in CICDDoS2019 and LITNET-2020, respectively. In contrast, FedAvg, Vanilla SL, and SplitFed with random node selection have high communication overhead. The Vanilla SL achieves 64 and 79 MB in CICDDoS2019 and LITNET-2020, respectively. The FedAvg has better performance than Vanilla SL, it achieved 50 and 64 MB in CICDDoS2019 and LITNET-2020, respectively. For SplitFed, it outperforms FedAvg and Vanilla SL with 59 and 70 MB. Finally, the DFSL with the proposed MNS scheme performs effectively on the performance. Selecting only the nodes with important update, good channel quality, and the nearest distance from the server reduces the communication overhead.

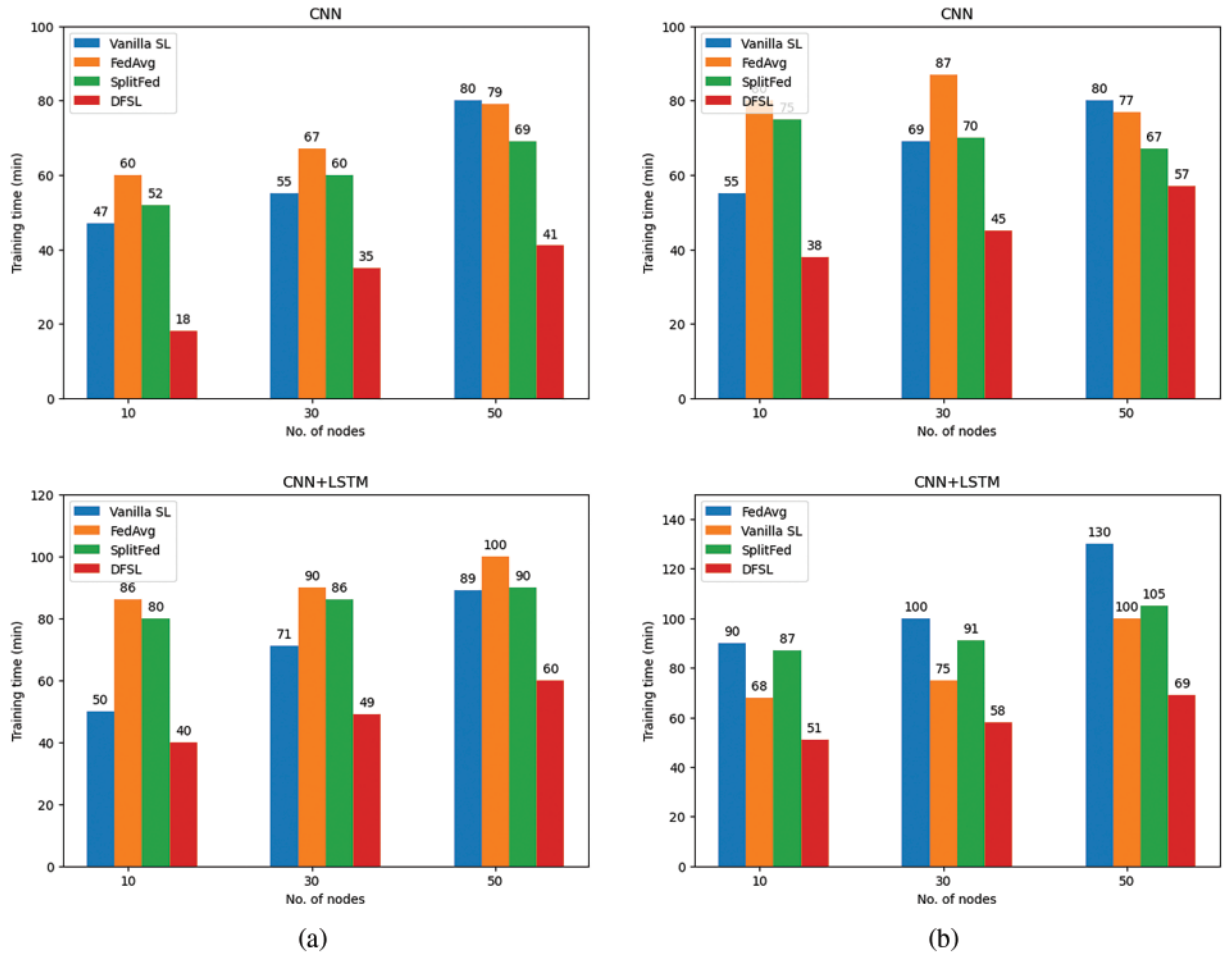

Figure 10: Training time performance (a) CICDDoS2019; (b) LITNET-2020

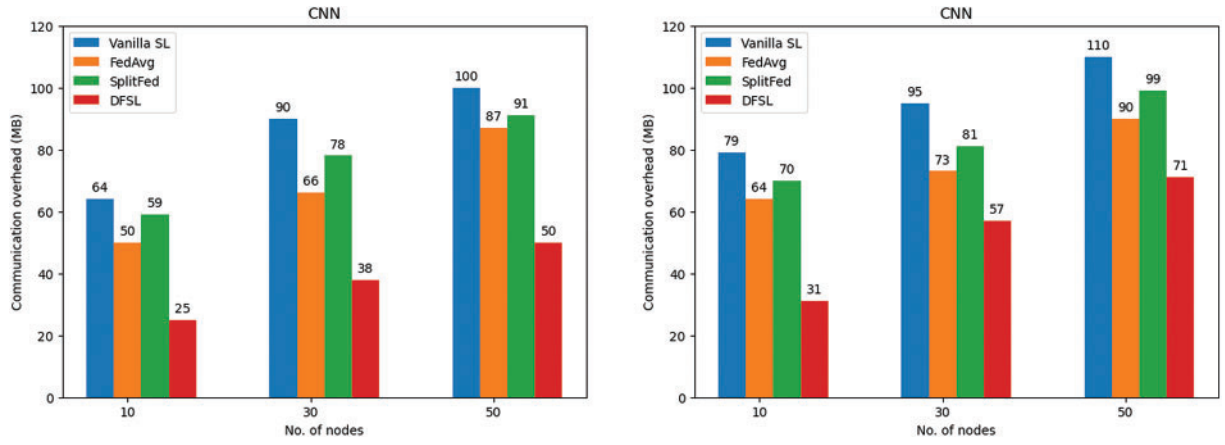

Figure 11: (Continued)

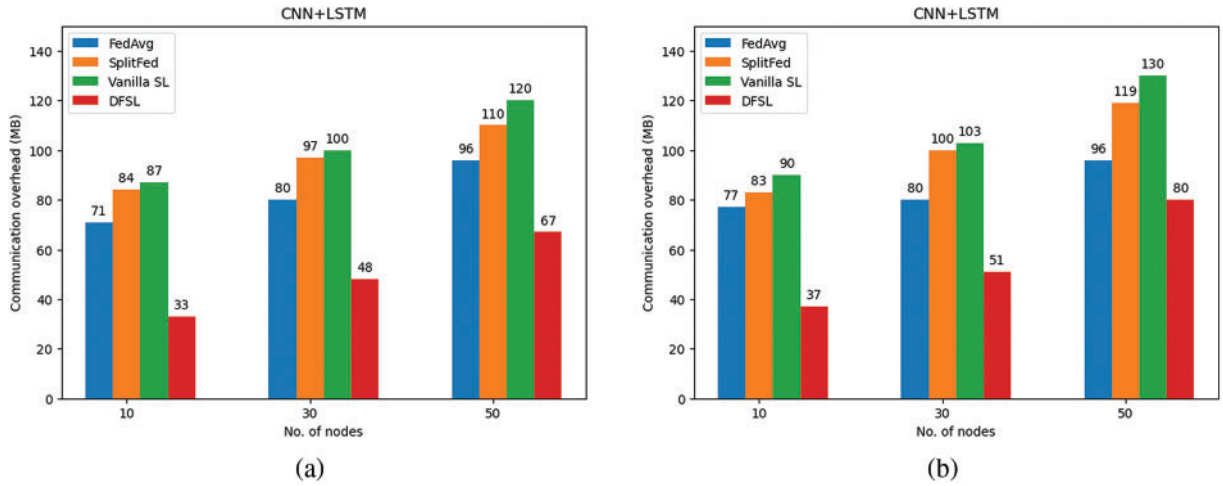

Figure 11: Communication overhead performance (a) CICDDoS2019; (b) LITNET-2020

Tables 7 and 8 summarize the training time and communication overhead results with the increasing number of nodes for CNN and CNN+LSTM in CICDDoS2019 and LITNET-2020.

Table 7: Training time and communication overhead results in the CNN model

Algorithm	No. of nodes	Training time (min)	Communication overhead (MB)	Training time (min)	Communication overhead (MB)
		CICDDoS2019		LITNET-2020	
Vanilla SL	10	47	64	55	79
	30	55	90	69	95
	50	80	100	80	110
FedAvg	10	60	50	80	64
	30	67	66	77	73
	50	79	87	87	90
SplitFed	10	52	59	75	70
	30	67	78	70	81
	50	69	91	67	99
DFSL	10	18	25	38	31
	30	35	38	45	57
	50	41	50	57	71

Table 8: Training time and communication overhead results in the CNN+LSTM model

Algorithm	No. of nodes	Training time (min)	Communication overhead (MB)	Training time (min)	Communication overhead (MB)
		CICDDoS2019		LITNET-2020	
Vanilla SL	10	50	87	68	90
	30	71	100	75	103
	50	89	120	100	130
FedAvg	10	86	71	90	77
	30	90	80	100	80
	50	100	96	130	96
SplitFed	10	80	84	83	37
	30	86	97	100	51
	50	90	110	119	80
DFSL	10	45	33	51	37
	30	50	48	58	51
	50	65	67	69	80

6.2.3 Comparing the Proposed Adaptive Aggregation with Other Aggregation Methods

The proposed adaptive aggregation in DFSL is compared with the three aggregation methods: geometric median [41], Krum aggregation function [42], and Fourier Transform [43], as shown in Table 9. We find that our aggregation method is more effective compared with the other methods. Determining the maximum score for accuracy and loss in each round enhances learning performance and makes the convergence more stable. Finally, we observe that our aggregation method gives more convergence stability without need for extra communications..

Table 9: Comparison of the proposed adaptive aggregation with existing aggregation methods

Method/Ref.	Accuracy (%)
Geometric median [41]	64.30
Krum function [42]	93.50
Fourier transform [43]	80.83
Proposed aggregation	99.87

6.2.4 Comparing the Proposed MNS Scheme with Other Selection Schemes

The proposed MNS scheme is compared with other selection schemes, such as BC, BN2, BS-BN2, and MAB-BC-BN2 schemes, as shown in Table 10. The MNS proves its effectiveness in reducing communication overhead compared with the other selection schemes. The dynamic selection of irrelevant updates and their exclusion during the training process helps to save computation and

communication resources. Testing the performance of the MNS scheme by increasing the number of nodes proves that the MNS can be deployed in real-world IoMT with negligible additional computation overhead and thus can be implemented on a large scale.

Table 10: Comparison of the MNS scheme with other selection schemes

Scheme	Communication overhead (MB)
BC	78
BN2	71
BC-BN2	40
MAB-BC-BN2	65
Proposed MNS	**25**

6.2.5 Scalability Analysis

The above sections explain experiments in scenario with different nodes, including 10, 30, and 50 nodes, to test the training time and communication overhead. We assess DFSL's performance across increasing DFSL sizes, measuring main metrics such as accuracy and convergence time (including ten rounds). To create the DFSL of extra sizes, we have increased the number of nodes up to 90 nodes.

The experiment has been executed 10 times for each DFSL size at each iteration. Fig. 12 shows the average accuracy and convergence time in LITNET-2020 dataset. The Figure shows that DFSL has consistent performance stability when the number of nodes increases. Furthermore, these results validate our proposed DFSL is stable with a large number of nodes.

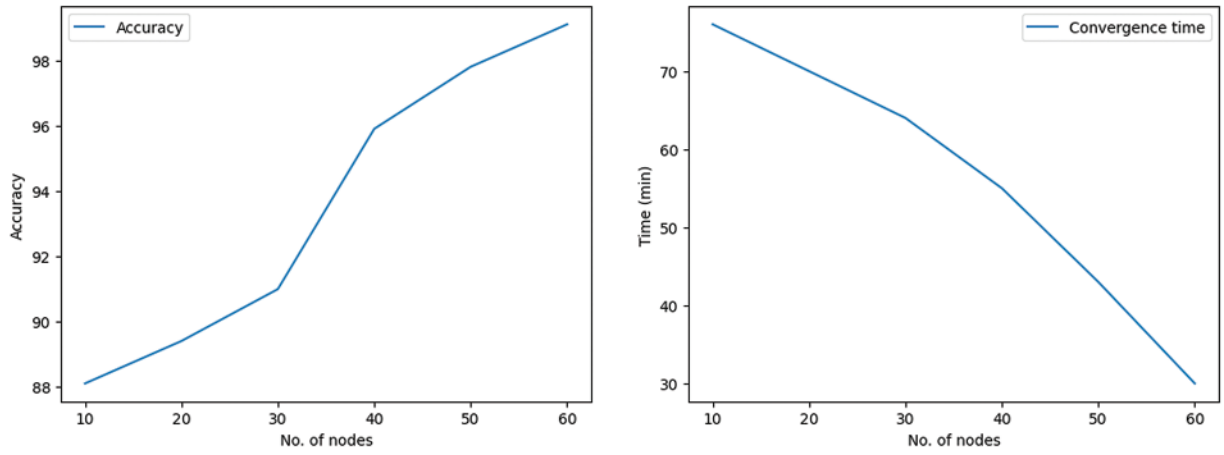

Figure 12: Scalability performance of the DFSL (a) accuracy; (b) convergence time

Therefore, with a few numbers of nodes, a low value on a few training sets can significantly impact on the overall average. In scenarios with more nodes, DFSL learns all attacks more accurately, although it consistently produces an accuracy below 0.93 when the number of nodes is 90.

The global model's ability to perform well greatly impacts the DFSL process's convergence time. By adding new nodes, the global model is learned more quickly, which has proven critical for the process's convergence.

7 Discussion

In summary, the experiments' results prove that the DFSL detects DDoS attacks with low loss value and high detection accuracy compared with the FedAvg, Vanilla SL, and SplitFed. In addition, it achieved low training time and communication overhead compared with FedAvg, Vanilla SL, and SplitFed.

In the first experiment, the DFSL training enhanced the model's performance in terms of loss and accuracy when more layers were trained at the node side. We observe that the model's performance achieves high accuracy and low loss by applying the early stopping strategy. In addition, we observe that the performance in LITNET-2020 is better than CICDDoS2019 performance; it contains more features and data to train. In the second experiment, stopping the model at a certain point when the model stops improvement helps to reduce training time. In contrast, the proposed MNS scheme helps to reduce communication overheads. It aims to decrease the communication rounds where only the important local update with good channel quality and the nearest node are selected for aggregation. It dynamically selects the relevant updates during the training process to avoid consuming computation and communication resources.

The model performance in CICDDoS2019 is better than LITNET-2020 in the second experiment, where more features need more training time and communication to transmit the model parameters.

Even though the proposed framework achieves acceptable results in terms of loss, accuracy, training time, and communication overhead, there are four implementation challenges and limitations of the proposed system, including:

1. **Achieving consistent accuracy:** The data heterogeneity in IoMT across different nodes makes it a challenge to maintain constant accuracy results. Some nodes that have more heterogeneity rates need more training time for convergence.
2. **Anomalies data complexity:** Notwithstanding our preprocessing efforts, some nodes sporadically presented anomalous data patterns. These could be attributed to distinct local network behaviors, which sometimes inject noise through model training.
3. **Real-time data:** Although the proposed system is examined with two non-IID datasets proposed for DDoS attacks, it needs to be examined on real-time data to test its effectiveness.
4. **Real-time application:** Although the proposed system is simulated by PyTorch, it needs to be examined in a real-time IoMT application.

8 Conclusion

A novel distributed FSL (DFSL) system is proposed in this paper to detect Distributed Denial of Service (DDoS) attacks in the IoMT networks while enhancing detection accuracy and reducing training time and communication overhead. The proposed system uses Federated Split Learning (FSL) to take advantage of each approach. The proposed system is tested on two DL models, including the Convolutional Neural Network (CNN) and a hybrid model that combines CNN and Long Short-Term Memory (LSTM) to determine the best model to detect DDoS attacks and test the effectiveness of the system. The procedure of the proposed system starts with initialization and model splitting, local model training, and adaptive aggregation. The adaptive aggregation method is designed based

on the early stopping strategy to enhance learning performance and reduce training time. In addition, a Multi-Node Selection (MNS) based BC-BN2 selection scheme is proposed to reduce communication overhead with three selection metrics: the importance of local update, channel quality, and the nearest distance from the server. The performance results of the proposed system are compared with a baseline of distributed learning such as FedAvg, Vanilla SL, and SplitFed algorithms with different cut layers and increasing number of nodes. The results show the effectiveness of the proposed system where it can detect DDoS with an accuracy of 99.70% and 99.87%, and it achieved 18 and 38 min training time and 25 and 31 MB communication overhead in CICDDoS2019 and LITNET-2020, respectively. In future work, we aim to address the current limitations. We will further explore advanced convergence strategies with robust anomalous data management. In addition, we will deploy the proposed system in real IoMT scenarios to assess its feasibility in resource-constrained devices and networks.

Acknowledgement: The authors would like to extend their sincere thanks and gratitude to the supervisor for his support and direction of this research.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm their contribution to the paper as follows: study conception and design: Rasha Almarshdi; data collection: Rasha Almarshdi; analysis and interpretation of results: Rasha Almarshdi, Etimad Fadel, Nahed Alowidi, Laila Nassef; draft manuscript preparation: Rasha Almarshdi, Etimad Fadel, Nahed Alowidi, Laila Nassef. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data supporting the contributions of this study are open-source and available at https://www.unb.ca/cic/datasets/ddos-2019.html and https://epubl.ktu.edu/object/elaba:61188126/ (accessed on 22 September 2024).

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. A. Chelloug and M. A. El-Zawawy, "Middleware for Internet of Things: Survey and challenges," *Intell. Autom. Soft Comput.*, vol. 24, no. 2, pp. 309–318, 2018. doi: 10.1080/10798587.2017.1290328.

[2] D. Wang, D. Chen, B. Song, N. Guizani, X. Yu and X. Du, "From IoT to 5G I-IoT: The next generation IoT-based intelligent algorithms and 5G technologies," *IEEE Commun. Mag.*, vol. 56, no. 10, pp. 114–120, Oct. 2018. doi: 10.1109/MCOM.2018.1701310.

[3] S. Markkandan, S. Sivasubramanian, J. Mulerikkal, N. Shaik, B. Jackson and L. Naryanan, "Massive MIMO codebook design using gaussian mixture model based clustering," *Intell. Autom. Soft Comput.*, vol. 32, no. 1, pp. 361–375, 2022. doi: 10.32604/iasc.2022.021779.

[4] N. Javaid, A. Sher, H. Nasir, and N. Guizani, "Intelligence in IoT-based 5G networks: Opportunities and challenges," *IEEE Commun. Mag.*, vol. 56, no. 10, pp. 94–100, Oct. 2018. doi: 10.1109/MCOM.2018.1800036.

[5] M. Deepender, U. Shrivastava, and J. K. Verma, "A study on 5G technology and its applications in telecommunications," *IEEE Xplore*, vol. 7, pp. 365–371, 2021. doi: 10.1109/ComPE53109.2021.9752402.

[6] A. Sonavane, A. Khamparia, and D. Gupta, "A systematic review on the internet of medical things: Techniques, open issues, and future directions," *Comput. Model. Eng. Sci.*, vol. 137, no. 2, pp. 1525–1550, 2023. doi: 10.32604/cmes.2023.028203.

[7] H. -C. Chen and S. -S. Kuo, "Active detecting DDoS attack approach based on entropy measurement for the next generation instant messaging app on smartphones," *Intell. Autom. Soft Comput.*, vol. 25, no. 1, pp. 217–228. doi: 10.31209/2018.100000057.

[8] S. Innes, "Banner Health paid $1.25 million to resolve Federal Data Breach Probe," *The Arizona Republic*, 2023. Accessed: Feb. 03, 2023. [Online]. Available: https://www.azcentral.com/story/money/business/health/2023/02/04/banner-health-paid-1-25-million-to-resolve-federal-data-breach-probe/69871530007/.

[9] Y. Ko, K. Choi, H. Jei, D. Lee, and S. Kim, "ALADDIN: Asymmetric centralized training for distributed deep learning," in *Proc. 30th ACM Int. Conf. Inform. Knowl. Manage.*, 2021, pp. 863–872. doi: 10.1145/3459637.3482412.

[10] S. Kamei and S. Taghipour, "A comparison study of centralized and decentralized federated learning approaches utilizing the transformer architecture for estimating remaining useful life," *Reliability Eng. Syst. Saf.*, vol. 233, May 2023, Art. no. 109130. doi: 10.1016/j.ress.2023.109130.

[11] N. N. Thilakarathne *et al.*, "Federated learning for privacy-preserved medical internet of things," *Intell. Autom. Soft Comput.*, vol. 33, no. 1, pp. 157–172, 2022. doi: 10.32604/iasc.2022.023763.

[12] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, Aug. 2018. doi: 10.1016/j.jnca.2018.05.003.

[13] Z. Zhang, A. Pinto, V. Turina, F. Esposito, and I. Matta, "Privacy and efficiency of communications in federated split learning," *IEEE Trans. Big Data*, vol. 9, no. 5, pp. 1380–1391, Oct. 2023. doi: 10.1109/TBDATA.2023.3280405.

[14] M. M. Amiria, D. Gunduzb, S. R. Kulkarni, and H. V. Poor, "Convergence of update aware device scheduling for federated learning at the wireless edge," *IEEE Trans. Wirel. Commun.*, vol. 20, no. 6, pp. 3643–3658, 2021. doi: 10.1109/TWC.2021.3052681.

[15] X. Liu, Y. Deng, and T. Mahmoodi, "Wireless distributed learning: A new hybrid split and federated learning approach," *IEEE Trans. Wirel. Commun.*, vol. 22, no. 4, pp. 2650–2665, 2022. doi: 10.1109/twc.2022.3213411.

[16] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020. doi: 10.1109/MSP.2020.2975749.

[17] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "FedAvg with fine tuning: Local updates lead to representation learning," May 2022. doi: 10.48550/arxiv.2205.13692.

[18] J. Shen, X. Wang, N. Cheng, L. Ma, C. Zhou, and Y. Zhang, "Effectively heterogeneous federated learning: A pairing and split learning based approach," in *GLOBECOM 2023–2023 IEEE Glob. Commun. Conf.*, Kuala Lumpur, Malaysia, Dec. 2023, pp. 5847–5852. doi: 10.1109/GLOBECOM54140.2023.10437666.

[19] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," in *Proc. Int. Conf. Learn. Rep. (ICLR) Workshop AI Social Good*, 2019, pp. 1–7. doi: 10.48550/arXiv.1812.00564.

[20] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "SplitFed: When federated learning meets split learning," *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 8, pp. 8485–8493, Jun. 2022. doi: 10.1609/aaai.v36i8.20825.

[21] C. Huang, G. Tian, and M. Tang, "When minibatch SGB meets splitfed learning: Convergence analysis and performance evaluation," 2023. doi: 10.48550/arXiv.2308.11953.

[22] Y. Mu and C. Shen, "Communication and storage efficient federated split learning," in *Proc. : ICC 2023-IEEE Int. Conf. Commun.*, May 2023, pp. 2976–2981. doi: 10.1109/icc45041.2023.10278891.

[23] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," 2019, *arXiv:1909.09145*.

[24] Y. K. Saheed and M. O. Arowolo, "Efficient cyber attack detection on the internet of medical things-smart environment based on deep recurrent neural network and machine learning algorithms," *IEEE Access*, vol. 9, pp. 161546–161554, 2021. doi: 10.1109/ACCESS.2021.3128837.

[25] M. Manimurugan, S. Al-Mutairi, M. M. Aborokbah, N. Chilamkurti, S. Ganesan and R. Patan, "Effective attack detection in internet of medical things smart environment using a deep belief neural network," *IEEE Access*, vol. 8, pp. 77396–77404, 2020. doi: 10.1109/ACCESS.2020.2986013.
[26] M. Usman, M. A. Jan, X. He, and J. Chen, "P2DCA: A privacy-preserving-based data collection and analysis framework for IoMT applications," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1222–1230, Jun. 2019. doi: 10.1109/JSAC.2019.2904349.
[27] J. Ren, J. Guo, W. Qian, H. Yuan, X. Hao and J. Hu, "Building an effective intrusion detection system by using hybrid data optimization based on machine learning algorithms," *Secur. Commun. Netw.*, vol. 2019, pp. 1–11, Jun. 2019. doi: 10.1155/2019/7130868.
[28] S. P. K. Gudla, S. K. Bhoi, S. R. Nayak, and A. Verma, "DI-ADS: A deep intelligent distributed denial of service attack detection scheme for fog-based IoT applications," *Math. Probl. Eng.*, vol. 2022, pp. 1–17, Aug. 2022. doi: 10.1155/2022/3747302.
[29] R. Priyadarshini and R. K. Barik, "A deep learning based intelligent framework to mitigate DDoS attack in fog environment," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 3, pp. 825–831, 2022. doi: 10.1016/j.jksuci.2019.04.010.
[30] Y. Zhang, Y. Liu, X. Guo, Z. Liu, X. Zhang and K. Liang, "A BiLSTM-based DDoS attack detection method for edge computing," *Energies*, vol. 15, no. 21, Oct. 2022, Art. no. 7882. doi: 10.3390/en15217882.
[31] P. Verma, J. G. Breslin, and D. O'Shea, "FLDID: Federated learning enabled deep intrusion detection in smart manufacturing industries," *Sensors*, vol. 22, no. 22, Nov. 2022, Art. no. 8974. doi: 10.3390/s22228974.
[32] V. Rey, P. M. Sánchez Sánchez, A. Huertas Celdrán, and G. Bovet, "Federated learning for malware detection in IoT devices," *Comput. Netw.*, vol. 204, Feb. 2022, Art. no. 108693. doi: 10.1016/j.comnet.2021.108693.
[33] Y. Alhasawi and S. Alghamdi, "Federated learning for decentralized DDoS attack detection in IoT networks," *IEEE Access*, vol. 12, pp. 42357–42368, Jan. 2024. doi: 10.1109/ACCESS.2024.3378727.
[34] F. Yu, B. Zeng, K. Zhao, Z. Pang, and L. Wang, "Chronic poisoning: Backdoor attack against split learning," *Proc. AAAI Conf. Artif. Intell.*, vol. 38, no. 15, pp. 16531–16538, Mar. 2024. doi: 10.1609/aaai.v38i15.29591.
[35] S. Abuadbba *et al.*, "Can we use split learning on 1D CNN models for privacy preserving training?," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, 2020, pp. 305–318. doi: 10.1145/3320269.3384740.
[36] M. A. Khan, V. Shejwalkar, A. Houmansadr, and F. M. Anwar, "Security analysis of SplitFed learning," in *Proc. ACM Conf. Embed. Netw. Sens. Syst.*, Nov. 2022. doi: 10.1145/3560905.3568302.
[37] F. Li, J. Lin, and H. Han, "FSL: Federated sequential learning-based cyberattack detection for industrial Internet of Things," *Ind. Artif. Intell.*, vol. 1, Mar. 2023, Art. no. 4. doi: 10.1007/s44244-023-00006-2.
[38] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," in *Proc. 2019 Int. Carnahan Conf. Secur. Technol. (ICCST)*, IEEE, Oct. 2019, pp. 1–8. doi: 10.1109/CCST.2019.8888419.
[39] R. Damasevicius *et al.*, "LITNET-2020: An annotated real-world network flow dataset for network intrusion detection," *Electronics*, vol. 9, no. 5, May 2020, Art. no. 800. doi: 10.3390/electronics9050800.
[40] A. Fernandez, S. Garcia, F. Herrera, and N. V. Chawla, "SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary," *J. Artif. Intell. Res.*, vol. 61, pp. 863–905, Apr. 2018. doi: 10.1613/jair.1.11192.
[41] K. Pillutla, S. M. Kakade, and Z. Harchaoui, "Robust aggregation for federated learning," *IEEE Trans. Signal Process.*, vol. 70, pp. 1142–1154, 2022. doi: 10.1109/TSP.2022.3153135.
[42] R. Taheri *et al.*, "Robust aggregation function in federated learning," in *Proc. 6th Int. Conf. Inform. Knowl. Syst.*, Cham, Springer Nature Switzerland, 2023, pp. 168–175. doi: 10.1007/978-3-031-51664-1_12.
[43] E. M. Campos, A. Jose, L. Ramos, and A. Skarmeta, "FedRDF: A robust and dynamic aggregation function against poisoning attacks in federated learning," 2024. doi: 10.48550/arXiv.2402.10082.