Chapter 4

Containerization-based Federated learning for Privacy-Preserving Task Offloading

Patient activity monitoring is a promising application of the Internet of Medical Things (IoMT), revolutionizing clinical diagnosis. An IoMT uses sensory data collected from smart devices to train a model on the server. The trained model recognizes the patient activities on the smart devices. However, training the model on the server has raised privacy concerns and security threats. The sensitive medical data transferred from the smart devices to the Cloud poses different cybersecurity challenges, such as distributed denial of service, phishing, network penetration, side-channel, etc. This chapter proposes a secure patient monitoring system using federated learning, optimized through the use of containerization. The system performs training on the local devices, each encapsulated within a dedicated container and sends only weight matrices to the server for aggregation. By adopting containerization, we ensure a secure, consistent, and isolated environment for the local devices, which enhances security and resource management.

The system intelligently divides the participants into clusters based on the available

resources, utilizing container orchestration for efficient resource allocation. Within each containerized cluster, suitable models are trained, and performance is further enhanced through knowledge distillation. The model of high-performing clusters shares its knowledge with the models in smaller containerized clusters to improve their performance.

The experimental results clearly demonstrate that the proposed system effectively operates even in the presence of unequal resources. Containerization not only enhances security but also simplifies deployment, management, and scaling of the patient monitoring system in the IoMT ecosystem.

4.1 Introduction

The IoMT facilitates autonomous monitoring of patient activities using a variety of wearables and environment sensors [**?**]. It helps in accelerating the clinical diagnosis and enables real-time tracking of the user's activities. Smartphones strengthen the rapid growth of IoMT by curtailing the cost of purchasing additional infrastructure. Smartphones provide the convenience of monitoring activities and constructive guidelines to the users (or patients) [**?**]. For example, the actions (or activities of daily living) of older adults living in isolation can be easily monitored using smartphone sensors from a different location [**?**]. IoMT can be exploited to monitor a large number of people living in isolation during pandemics like COVID-19 [**?**].

IoMT applications use various smart devices like smartphones, smart wearables, and smart bands. The onboard sensors of such devices generate a large amount of data to perform a given task of real-time monitoring and detection [**?**]. Researchers have exploited this colossal data to train machine or deep learning models for various applications. Moreover, current scenarios rely upon distributed architectures, where models are trained on the server using data collected by physical devices (or edge devices). However, it introduces the inherent challenges of data privacy compromise and security threats. The attackers may compromise the sensitive healthcare data

transmitted from the smart devices to the Cloud. The attacks on the healthcare system include distributed denial of service (DDoS), malware, ransomware attacks, *etc.*

Federated Learning (FL) is an emerging paradigm that facilitates collaborative learning among multiple devices (or participants) without compromising data privacy [**?**, **?**]. Thus, FL proves to be an essential enabler technique to preserve data privacy and minimize communication delay via local training and global aggregation, as shown in Fig. 4.1. In the FL settings, it is assumed that all the participants can train the shared global model using available network bandwidth and dataset [**?**]. However, assuming homogeneous dataset availability and network connectivity on all the participants is unrealistic in real-world scenarios. Therefore, effectively utilizing data and resources on the heterogeneous participant is still a research challenge in FL [**?**]. In this work, we consider the participants (edge devices) that possess non-identical resources due to heterogeneity in their software and hardware components. The resources include processing, bandwidth, and memory. Therefore, the participants or edge devices are termed heterogeneous participants. Incorporating containerization into this context, each participant in the FL system operates within a containerized environment. Containerization ensures that the software, models, and dependencies required for the FL process are encapsulated in isolated containers, enhancing security and resource management.

This chapter aims to design a secure and intelligent patient monitoring system using FL in IoMT. The system can handle the heterogeneity of available resources on the different participants in FL for patient monitoring. We investigate the following problem in this chapter: *how to develop a secure and intelligent patient monitoring system that can ensure data privacy and reduce communication delay among heterogeneous participants in FL?* Here, the intelligent system refers to the ability to handle heterogeneous participants without compromising accuracy. We assume a given large-size model that can recognize different activities of the patients with higher accuracy. However,

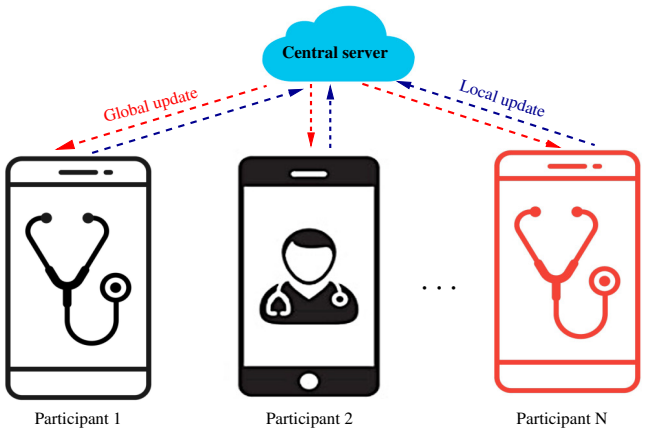


Figure 4.1: An example scenario of a smartphone-based patient monitoring system using federated learning.

it requires a considerable amount of data and resources during training. Therefore, we divide the participants into several clusters and train the optimal model on each cluster without using private participants' data. We use the Knowledge Distillation (KD) technique [**?**] to improve the performance of the lightweight model in the cluster.

Our contributions: To the best of our knowledge, this is the first work to address the problem of patient monitoring by incorporating FL in IoMT. The first contribution is to determine the optimal number of clusters such that each participant can run a suitable model. We have jointly considered the parameters such as bandwidth, dataset size, and freshness of data at the participants during clustering. In the next contribution, we generate a large-size Deep Neural Network (DNN) (suitable for patient activities monitoring) at the server that satisfies the parameters of participants in cluster 1, arranged in decreasing order of their available parameters. Further, the model on the server is iteratively compressed using KD and deployed on the participants of different clusters to perform FL. Finally, we obtain a robust model on each participant trained on local datasets of multiple participants without sharing actual data.

The rest of the chapter is organized as follows. In the next section, we present the preliminary and system model. Section 4.3 elaborates on the proposed patient monitoring system using FL in IoMT. Further, we evaluate the system in Section 4.4.

Finally, the chapter concludes in Section 4.5.

4.2 Preliminary and System Model

This work considers a scenario of the patient monitoring system that comprises a set $\mathcal{P}$ of N participants, where $\mathcal{P} = \{p_1, p_2, \cdots, p_N\}$, and a server S. Each participant p_i is willing to share the data of the patient activities to participate in the federation. Let D_i denote the dataset at the participant p_i having Q_i instances. Let $\{\mathbf{x}_i^j, y_i^j\}$ represent the j^{th} instance of $\mathcal{D}_i$, where $\mathbf{x}_i^j$ and y_i^j are input vector and class label, respectively. The class label can be an activity of a patient, such as taking medicine, eating, or sleeping. The operation of monitoring using FL starts with the transmission of randomly initialized model M from S to all the participants. Next, the participants train the model M on their local dataset D_i to obtain a weight parameter matrix W_i. Further, W_i of all the participants are transferred to the server that performs the weight parameter matrices aggregation. The result updated parameter matrix from the server is transferred to all the participants for another round of training. This local training and global aggregations continue for R global iteration to complete the training of M. Fig. 4.1 illustrates an overview of the proposed patient monitoring system.

Definition 4.1 (Federated Learning) *Federated learning refers to the concept of alleviating the privacy compromise and minimizing the communication overhead while training a shared model using data from multiple decentralized participants [?]. The participants transfer trained model parameter matrices despite data to the server for aggregation and receive the global parameter matrix for further training.*

Definition 4.2 (Freshness of data) *The freshness ($\mathcal{F}_i$) of patients monitoring data ($\mathcal{D}_i$) at participant p_i is inversely proportional to the time elapsed (ΔT_i) between data collection and its incorporation into the training, i.e., $\mathcal{F}_i \propto \frac{1}{\Delta T_i}$, where $1 \leq i \leq N$.*

Let B_i, $|\mathcal{D}_i|$, and $\mathcal{F}_i$ denote the available bandwidth between S and p_i, size of $\mathcal{D}_i$, and freshness of data, respectively. Since B_i, $|\mathcal{D}_i|$, and $\mathcal{F}_i$ vary from one participant to other. Thus, assuming successful training of a common model M from S on the participants is unrealistic. Therefore, the proposed patient monitoring system divides the participants into a set $\mathcal{C}$ of k different clusters, where $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$, and a cluster C_j has N_j participants. Further, the set $\mathcal{C}$ covers all the participants, *i.e.,* $\sum_{j=1}^{k} N_j = N$, where $1 \leq j \leq k$. The clustering criteria to determine k different cluster is discussed in Section 4.3.1. Later, the server generates different versions of M, *i.e.,* $M_1, M_2, \cdots, M_k$ for clusters $C_1, C_2, \cdots, C_k$, respectively (detailed in Section 4.3.2). Thereafter, each participant in cluster C_k train model M_k using KD technique, as discussed in Section 4.3.3.

4.3 Patient Monitoring System using FL in IoMT

This section proposes a patient monitoring system to recognize the different activities of patients using FL. The objective of the monitoring system is to train a global shared model for each participant using FL in IoMT. It ensures the privacy-preserving training of the patient activities recognition model in a distributed manner without sharing sensitive data. The significant components underlying the patient monitoring system are A) clustering of participants in the federation, B) generation of different models at the server, C) KD-based training of lightweight model at the participants, and D) clustering-wise aggregation.

4.3.1 Clustering of Participants in Federation

The diversity of B_i, $|\mathcal{D}_i|$, and $\mathcal{F}_i$ restrict the successful and quality training of global shared model M on all the participants. In other words, some participants could have higher or lower values of B_i, $|\mathcal{D}_i|$, and $\mathcal{F}_i$ $(1 \leq i \leq N)$. Thus, the training time and quality of training for some participants are superior, while some participants incur

huge training delays and provide poor quality training. The simple aggregation of weight parameter matrices from such diverse participants results in poor quality of aggregated weight parameter matrix and requires higher training time. The obtained matrix results in severe performance compromise for some participants. Therefore, the proposed patient monitoring system via FL divides N participants into a set $\mathcal{C}$ of k clusters, where $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$. Later, each cluster C_j have N_j participants, where $1 \leq j \leq k$ and $\sum_{j=1}^{k} N_j = N$.

Clustering criteria: This work uses three parameters, including B_i between participant p_i and server, $|\mathcal{D}_i|$, and $\mathcal{F}_i$, where $1 \leq i \leq N$. As the contribution of all three parameters are different; therefore, we have considered three variables, *i.e.,* λ_1, λ_2, and λ_3 for B_i, $|\mathcal{D}_i|$, and $\mathcal{F}_i$, respectively. Further, we estimate strength $\mathcal{S}(p_i)$ of participant p_i in terms of B_i, $|\mathcal{D}_i|$, and $\mathcal{F}_i$, is estimated as follows:

$$\mathcal{S}(p_i) = \alpha\lambda_1 B_i + \lambda_2|\mathcal{D}_i| + \beta\lambda_3\mathcal{F}_i, \tag{4.1}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$. The value of λ_1, λ_2, and λ_3 varies from one scenario to other. For example, if the bandwidth is at the high priority then $\lambda_1 > \lambda_2$ and λ_3. Similarly, if the size of the dataset at the participant is at high priority then $\lambda_2 > \lambda_1$ and λ_3. α and β are the constants that make B_i and $\mathcal{F}_i$ comparable and dimension-less as $|\mathcal{D}_i|$, respectively; thus, making $\mathcal{S}(p_i)$ a dimension-less quantity.

We divide N participants into k clusters based on their strength $\mathcal{S}(\cdot)$. The target is to reduce the intra-cluster strength and increase the inter-cluster distance. The server performs the clustering of N participants into k clusters. Initially, all the participants expressed their willingness to request to participate in the federation. The server accepts the request and extracts the information about bandwidth B_i. Later, the server requested information about the size of the local dataset $|\mathcal{D}_i|$ at p_i and freshness of data $\mathcal{F}_i$ (*i.e.,* time elapsed between collection of data to the time of request) from all the participants. Upon receiving the information from all the participants, the server

estimates $\mathcal{S}(\cdot)$ for all the participants and generates k clusters. Further, the server arranges the clusters in descending order of their strength, *i.e.*, a participant of C_j has higher strength than the participant of C_{j+1}. Using the compression technique, the server generates different models for different clusters, discussed in the next section.

4.3.2 Generation of Different Models at the Server

This subsection describes the details of the generation of different models $(M_1, M_2, \cdots, M_k)$ by the server for each cluster in $\mathcal{C}$. Initially, the server builds a large-size model or Deep Neural Networks (DNN) M that can not be successfully executed on all the participants. Therefore, the server performs DNN compression to obtain a suitable model for each cluster. The steps involved in the DNN compression are 1) application of dropout on large-size DNN followed by 2) reducing resource requirements of dropout DNN.

4.3.2.1 Application of Dropout on Large-size DNN

The application of dropout over large-size DNN to curtail down unimportant or inferior connections. The resultant dropout DNN is equivalent to a lightweight DNN with its weights scaled with a given dropout rate. We estimate the optimal dropout that best suits our resources and accuracy requirements in this chapter. To initialize the selection of optimal dropout, we select a dropout rate (denoted by d) preferably with a higher value like $d = 0.5$ for hidden units and $d = 0.8$ for input units [**?**]. Let Q_b and Q_a denote the number of connections of a DNN, before and after dropout, respectively. Let $\max_{iteration}$, and c are the maximum iteration runs for the dropout and a hyper-parameter, respectively. The updated dropout rate is given as follows: $d' \leftarrow d \times \max\left\{\sqrt{\frac{Q_b}{Q_a}}, \left(1 - \frac{iteration}{c \times \max_{iteration}}\right)\right\}$.

4.3.2.2 Reducing Resource Requirements of Dropout DNN

The section discusses the methodology for reducing the resource requirements of dropout DNN obtained in Section 4.3.2.1. Apart from the prior work on reducing resources of either convolutional or recurrent layers. We introduce the mechanism for shrinking the resource requirements of DNN layers, including convolutional, fully connected, and recurrent (Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU)).

• **Weight factorization:** In this chapter, we are using the weight factorization technique [**?**] for reducing the parameters and FLOPs involved in convolutional and fully connected layers. The weight factorization technique introduces an intermediate multiplexing layer between two layers of the network. This factorization of layers reduces the computation requirement: if the size of the intermediate layer $\leq \frac{I_i \times O_i}{I_i + O_i}$ [**?**]. Here I_i and O_i are input and output dimensions of layer i, respectively.

• **Minimal Gated Unit for LSTM and GRU:** The parameters and FLOPs involved in the LSTM and GRU directly depend upon the gated operations. Therefore, to reduce the resource requirements of DNN, we use the concept of Minimal Gated Unit (MGU) inspired from [**?**]. We replace LSTM with coupled LSTM and GRU with MGU, respectively.

4.3.3 KD-based Training of Lightweight Model on the Participants

We assume that the model M_1 for cluster C_1 is the same as the large-size model developed at the server, *i.e.,* M. In other words, the participants in cluster C_1 can successfully execute the training of M with given bandwidth and data size. Initially, the server transfers M (or M_1) to the participants of cluster C_1. The participants train M_1 on their local datasets. It generates local weight parameter matrices for each participant.

This work uses standard cross entropy loss during training of M_1. The standard cross-entropy loss estimates the discrepancy between the predicted and true label of

DNN [**?**]. Let $\mathbf{X} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(n)}\}$ represents the set of input feature vectors in the training dataset. The cross-entropy loss function $\mathcal{L}_{CE}(\cdot)$ of DNN is given as follows:

$$\mathcal{L}_{CE}(\mathbf{Y}, \hat{\mathbf{Y}}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{l}\mathbf{1}(y^{(i)} = j)\log p(y^{(i)} = j \mid \mathbf{x}^{(i)}), \tag{4.2}$$

where, $\mathbf{Y} = \{y^{(1)}, \cdots, y^{(n)}\}$ and $\hat{\mathbf{Y}} = \{\hat{y}^{(1)}, \cdots, \hat{y}^{(n)}\}$ correspond to set of true and predicted labels, respectively. n and l denote number of data samples and classes in dataset $\mathcal{D}$, respectively.

Further, the availability of bandwidth, size of the dataset, and freshness of data on consecutive clusters from C_1 to C_k decreases, *i.e.,* strength of $C_1 > C_2 > \cdots > C_k$. In other words, size of $M_1 > M_2 > \cdots > M_k$. It implies the performance of M_1 and M_k is highest and lowest, respectively. If we directly train the model on participants in clusters C_2 to C_k, it leads to performance compromise and inferior quality of weight parameter matrix. Therefore, we use knowledge distillation [**?**] from the model trained in the previous cluster (teacher) to improve the performance of the model in the present cluster (student). KD from teacher to student, while training of students on raw data improves its generalization ability. This improvement helps in enhancing the performance of students.

The training of model $M_2 - M_7$ uses the following loss functions: standard cross-entropy loss function (given in Eq. 4.2) and distillation loss (denoted as $\mathcal{L}_{DL}(\cdot)$). The distillation loss between student and teacher, during their student training, improves the generalization ability of M^s in recognizing the class labels. Let s_{ij} and z_{ij} denote elements of logits generated by student and teacher, respectively. The distillation loss $\mathcal{L}_{DL}(\cdot)$ is given as

$$\mathcal{L}_{DL}(\mathbf{s}, \mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\left\|s_{ij} - z_{ij}\right\|_2^2. \tag{4.3}$$

Upon completion of the local training on all the participants in a cluster, the participants send their weight parameter matrices to the server for cluster-wise aggregation, detailed in Section 4.3.4. This step is repeated for all k clusters.

• **Estimating Total Transmission Time:** Let B_i, ϕ_i, and θ_i are the bandwidth, channel coefficient, and path loss frequency between p_i and S, respectively. Let p_j be a participant of cluster C_1, where $1 \leq j \leq N_1$. Here, the target is to find weight parameter matrix W_j for p_j to minimize a loss function $\mathcal{L}_j(\cdot)$. Let R_j denote the number of rounds needed for transferring model parameters to saturate FL between p_j and S. At r^{th} ($r \in R_j$) round of update the participant p_j minimizes the function $\mathbf{\Pi}_j(W_j)$ iteratively, until desired local accuracy is achieved. The function $\mathbf{\Pi}_j(W_j)$ is defined as [**?**];

$$W_j^r = \arg\min_{W_j} \mathbf{\Pi}_j(W_j|W^{(r-1)}, \nabla\mathcal{L}^{(r-1)}(\cdot)). \tag{4.4}$$

After, r^{th} round of update, all the participants of first cluster ($1 \leq i \leq N_1$) transfer W_j^r to S using wireless link. At S, the parameter and gradient are estimated as $W^{(r+1)} = \frac{1}{N_1}\sum_{j=1}^{N_1} W_j^r$, and $\nabla\mathcal{L}^{(r+1)}(\cdot) = \frac{1}{N_1}\sum_{j=1}^{N_1} \nabla\mathcal{L}_j^r(\cdot)$, respectively. The parameters transmission between p_i and S depend upon the wireless channel's transmission rate. Let $|W_1|$ represents bits in W_1. By using Shannon channel capacity [**?**], the total time transmission time ($\mathcal{T}_1$) to transfer the weight parameters of M_1 to S, is

$$\mathcal{T}_1 = \frac{|W_1|}{B_1 \log_2\left(1 + \frac{\theta_1\phi_1^2}{\sigma^2}\right)} \times R_1. \tag{4.5}$$

The patient monitoring system transfers the trained M_1 from S to the participants of G_2 and an untrained lightweight version model of M_1 denoted as M_2. The size of M_2 is sufficient for given strength at C_2. The participants use KD to train M_2 by using trained M_1 and local data, as discussed previously. Further, S compresses M_1 by α to generates a compressed M_2. Similar as Eq. 4.5, the total time transmission time ($\mathcal{T}_2$) to transfer the weight parameters of M_2 to S, is

$$\mathcal{T}_2 = \frac{|W_2|}{B_2 \log_2\left(1 + \frac{\theta_2 \phi_2^2}{\sigma^2}\right)} \times R_2. \tag{4.6}$$

$\mathcal{T}_1$ and $\mathcal{T}_2$ are the maximum time taken by a participants in clusters C_1 and C_2, respectively. Similarly, we repeat the steps for remaining clusters $\{C_3, \cdots, C_k\}$ and train the models $\{M_3, \cdots, M_k\}$. Such models are suitable for the bandwidths $\{B_3, \cdots, B_k\}$, size of data $\{|\mathcal{D}|_3, \cdots, |\mathcal{D}|_k\}$, and freshness of data $\{|\mathcal{F}|_3, \cdots, |\mathcal{F}|_k\}$ for the participants of remaining clusters.

Lemma 4.1 *The total time requires to transfer weight parameters from all the participants in $\mathcal{P}$ is given by*

$$\mathcal{T} = \sum_{i=1}^{k} \alpha^{(i-1)} \frac{|W_1|}{B_1 \log_2\left(1 + \frac{\theta_1 \phi_1^2}{\sigma^2}\right)} \times R_i. \tag{4.7}$$

Proof: From Eq. 4.5 and Eq. 4.6, we have value of value of $\mathcal{T}_1$ and $\mathcal{T}_2$. Similarly, we can obtain,

$$\mathcal{T}_3 = \frac{|W_3|}{B_3 \log_2\left(1 + \frac{\theta_3 \phi_3^2}{\sigma^2}\right)} \times R_3. \tag{4.8}$$

Further, the size of M_2 is α times M_1; thus, we can obtain $\frac{\mathcal{T}_2}{R_2} = \alpha \frac{\mathcal{T}_1}{R_1}$. Similarly, we have $\frac{\mathcal{T}_3}{R_3} = \alpha \frac{\mathcal{T}_2}{R_2} \implies \frac{\mathcal{T}_3}{R_3} = \alpha^2 \frac{\mathcal{T}_1}{R_1}$. From this, we can easily conclude that $\frac{\mathcal{T}_k}{R_k} = \alpha^{(k-1)} \frac{\mathcal{T}_1}{R_1}$. Using, this we can obtain

$$\mathcal{T} = \sum_{i=1}^{k} \alpha^{(i-1)} \frac{|W_1|}{B_1 \log_2\left(1 + \frac{\theta_1 \phi_1^2}{\sigma^2}\right)} \times R_i. \tag{4.9}$$

Hence proved. □

4.3.4 Cluster-wise Aggregation on the Server

All participants in $\mathcal{P}$ finish local training and send their updated weight parameter matrices to the server. The server performs aggregation to update the global model weight parameter. It broadcasts the weight parameter to the participants for the next round of local training. However, the weight parameter matrices of different clusters are different; thus, the direct application of FedAvg [**?**] proves incompetent in dealing with the varying number of class labels in the local dataset of participants. To tackle such an issue, we formulate cluster-wise aggregation as

$$\mathbf{\Omega}\left(W_j\right) = \sum_{i=1}^{N_j} \frac{Q_i}{\left(Q_1 + Q_2 + \cdots + Q_k\right)} \cdot W_j^i, \tag{4.10}$$

where W_j is the aggregated weight parameter matrix for cluster C_j $(1 \leq j \leq k)$ and Q_i is the number of instances of participants' local dataset in cluster C_j.

Algorithm 4.1 illustrates different steps involved in the proposed patient monitoring system. Later, the clusters with limited strength apply knowledge distillation techniques to improve their performance. We performed cluster-wise aggregation as the size of models is non-identical in different clusters. Whenever the members of the clusters change, *i.e.,* the participants are shifted from one cluster to another then the model of the shifted participant is initialized with the WPM of the assigned cluster. The computational complexity of Algorithm 1 directly resembles the convergence of FedAvg [**?**], *i.e.,* $\frac{1}{\mathcal{T}}$, where $\mathcal{T}$ is the total number of SGD operations required in R communication rounds [**?**] . Let E denote the number of local epochs running per communication round on each participant then $\mathcal{T} = RE$. Additionally, the proposed system's complexity also relies on cluster computation. However, the cluster-wise aggregation is performed in parallel and with a limited number of participants; thus, the convergence time is reduced, compensating for the computational complexity. Thus, the proposed system converges at the rate of $O(\frac{1}{\mathcal{T}})$.

Example 1 *Fig. 4.2 illustrates an example scenario of the proposed patient monitoring system.* ① *indicates the transfer of* M_1 *from* S *to participants of cluster* C_1. ② *depicts training of* M_1 *on participants of* C_1 *using FL.* ③ *represents aggregation of parameters and compression of* M_1 *to* M_2 *at* S. *The size of* M_2 *is* α *times* M_1 *as shown in Eq. 4.6.* ④ *indicates transfer of* M_2 *to the participants of* C_2. ⑤ *denotes training of* M_2 *on participants of* C_2. *We continue this approach for the participants of* C_k. *Algorithm 4.1 illustrates the steps involved in the proposed patient monitoring system.*

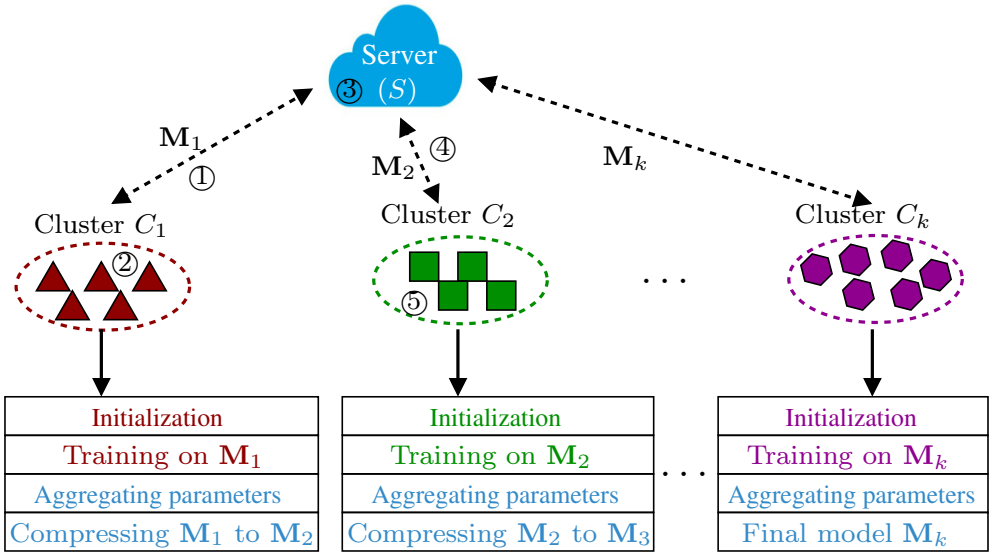


Figure 4.2: Illustration of an example scenario of the proposed patient monitoring system.

4.4 Experimental Evaluation and Results

During the empirical evaluation, we have considered existing DNN, *i.e.,* DeepZero [**?**]. We elaborate on CASAS smart home testbeds at Washington State University [**?**] used during experiments. Further, this section presents the parameter settings, followed by the experimental results.

Algorithm 4.1: Patient Monitoring System.

Input: Large-size model M on S and network bandwidth of participants in $\mathcal{P}$.
Output: Trained models $M_1, \cdots, M_k$.

Obtain k clusters $\{C_1, C_2, \cdots, C_k\}$;
for $j \leftarrow 1$ *to* k **do**
    S sends model M_j on the participants of cluster C_j;
    Initialize parameters of participants in C_j for M_j;
    while *not converge* **do**
        Train M_j on local dataset of participants in C_j;
        Participants in C_j transfers weight parameter matrices of M_j to S;
        S perform cluster-wise aggregation using Eq. 4.10;
        S returns aggregated matrix to the participants;
    Server compresses M_j to obtain M_{j+1};
        Apply dropout on M_j;
        Perform weight factorization on dropout M_j;
        Apply minimal gated unit operations on dropout M_j;
    $j \leftarrow j + 1$.
return Trained $M_1, \cdots, M_k$ on participants in clusters $C_1, \cdots, C_k$, respectively;

4.4.1 Existing Datasets

During the experimental evaluation, we use publicly available real datasets collected in CASAS smart home testbeds [**?**]. It assumes the different types of sensors are placed at a distinct location inside the apartment. We used six datasets collected from 6 different single-resident CASAS smart home apartments during the experimental analysis. We denote these datasets as, D_1, D_2, D_3, D_4, D_5, and D_6.

4.4.1.1 Layout for collecting data

All six datasets used in the experimental analysis exhibit a similar layout and sensor placements. Each dataset has a single bedroom, kitchen, dining area, and at least one bathroom. The inhabitants in each apartment were old age person who performed normal activities. During data collection, three sensors are used, namely, infrared motion sensors, wide-area motion sensors, and magnetic sensors denoted by s_1, s_2, and s_3, respectively.

4.4.1.2 Embedded sensors in the apartment

The embedded sensors (*i.e.,* s_1, s_2, and s_3) continuously and unobtrusively monitor the daily life activities of inhabitants in the apartments. All these sensors generate events only if significant changes occur in the physical surroundings of the inhabitants. The sensor data stored in the datasets are collected for a longer duration, and thousands of sensor events are generated while inhabitants perform activities of daily life.

4.4.1.3 Activities of Daily Living

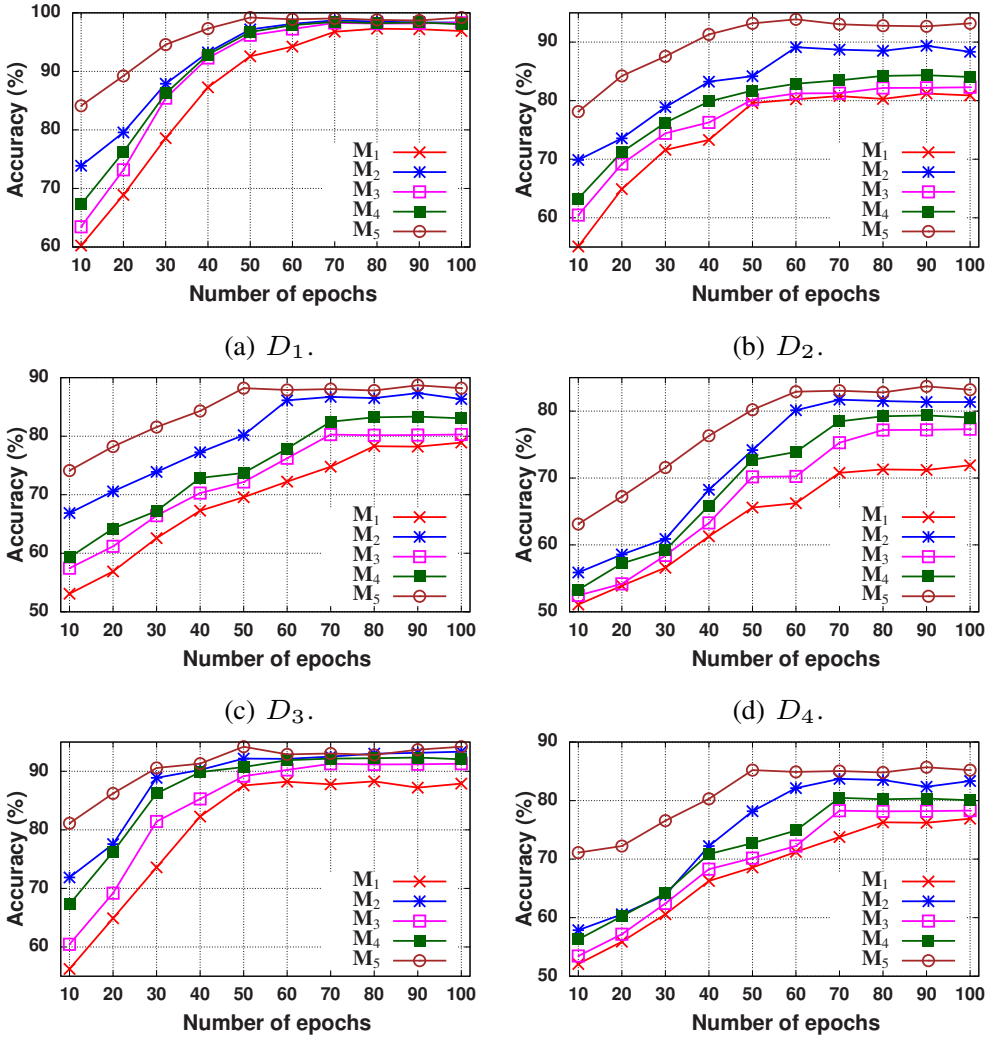
In the experimental analysis, we have considered 11 ADL in set $\mathcal{A}$ ($\mathcal{A} \in$\{bathing, bed toilet transition, eating, enter the home, housekeeping, leave home, meal preparation, personal hygiene, resting on the couch, sleeping in a bed, Taking medicine\}), which are denoted as $\{a_1, \cdots, a_{11}\}$. The dataset includes the date, time, sensor ID, sensor message linked with each sensor event and label.

4.4.2 Parameter Setting During Implementation

We consider the following five models during evaluation on five different clusters: $\mathbf{M}_1$ uses 40% compressed DeepZero model, $\mathbf{M}_2$ is 30% compressed, $\mathbf{M}_3$ is 20% compressed, $\mathbf{M}_4$ is 20% compressed, and $\mathbf{M}_5$ is original DeepZero model. These models are selected as per the criteria discussed in Section 4.3.1. For implementing the models $\mathbf{M}_1$-$\mathbf{M}_5$, we have incorporated the sequential model and functional API of deep learning library Keras in Python language. In the experimental analysis, we randomly divide the datasets into two sub-datasets, *i.e.,* training and testing with 70% and 30% data instances, respectively.

4.4.3 Experimental Results

This section first discusses several experiments to study the impact of datasets on performance, energy consumption, memory requirements and inference time.



(a) D_1. (b) D_2.

(c) D_3. (d) D_4.

(e) D_5. (f) D_6.

Figure 4.3: Performance results of different models ($\mathbf{M}_1 - \mathbf{M}_5$) on D_1, D_2, D_3, D_4, D_5, and D_6 datasets during training on all five clusters.

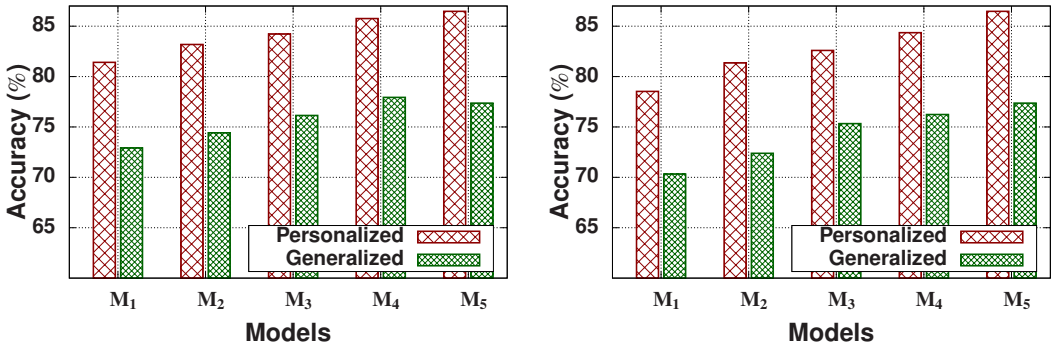
4.4.3.1 Number of epochs

At first, this work carried out experiments to determine the suitable number of epochs for evaluating different models including, $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$, and $\mathbf{M}_5$. In the experiment, the performance of the different models are reported on all the six considered datasets (*i.e.*, D_1, D_2, D_3, D_4, D_5, and D_6) during training. Fig. 4.3 illustrates the variation in the performance of different models on different datasets. The training performance of D_1 is higher, as shown in part (a) of Fig. 4.3 due to nearly equal training instances for

each class label. However, for D_4 the resultant training performance is worst and reaches maximum for model $\mathbf{M}_5$, *i.e.,* 84.23% at 60 epochs as shown in part(d) of Fig 4.3. We can observe from the result that the stability of $\mathbf{M}_5$ is achieved at the lowest epochs count (50). For $\mathbf{M}_1$ is highest of 80, as spatial and temporal dependencies are highly captured in $\mathbf{M}_5$ that stabilizes the model at lower epochs. Similarly, other models achieve stabilization in between $\mathbf{M}_5$ and $\mathbf{M}_1$ as they capture spatial and temporal dependencies, as shown in Fig. 4.3.

4.4.3.2 Personalized and generalized accuracy

Next, we illustrate the impact of different models ($\mathbf{M}_1 - \mathbf{M}_5$) on personalized and generalized accuracy. Personalized accuracy refers to the measure of the performance achieved by the participant during local inference. On the other hand, generalization refers to the measure of the performance achieved by the model during global inference. We consider scenarios of using KD and not using KD during the training of lightweight models, as illustrated in Fig. 4.4. The result depicts the positive effect of using KD during training of ($\mathbf{M}_1 - \mathbf{M}_4$). It improves both personalized and generalized accuracy. The personalized accuracy is on the local testing dataset, and generalized accuracy is on the global testing dataset, a randomly merged testing dataset of all the clients.



(a) With KD. (b) Without KD.

Figure 4.4: Personalized and generalized accuracy of different models ($\mathbf{M}_1 - \mathbf{M}_5$) on D_1 datasets with and without KD.

4.4.3.3 Memory consumption

Further, we estimate the memory consumption for storing task prediction models on a device and temporary memory for inference on different models. Since model $\mathbf{M}_1$ consumes the least memory in storage and inference; therefore, we use it as a baseline model. We estimate Memory Consumption Ratio (MCR) of i^{th} model with respect to $\mathbf{M}_1$ by using following relationship $\text{MCR}_i = \frac{mem\_cons_i}{mem\_cons_{\mathbf{M}_1}} \times 100, \forall i \in \{\mathbf{M}_2, \mathbf{M}_3, \mathbf{M}_4, \mathbf{M}_5\}$, where, mem_cons_i and $mem\_cons_{\mathbf{M}_1}$ denote the memory consumed by model i and model $\mathbf{M}_1$, respectively. Fig. 4.5 illustrates the MCR of $\mathbf{M}_5$ is highest due to the incorporation of the original DeepZero model. $\mathbf{M}_2$ can run in the least MCR (21%) because it holds 30% compress version of the DeepZero model. Similar patterns are observed for inference memory requirements as sharing reduces temporary memory during inference.

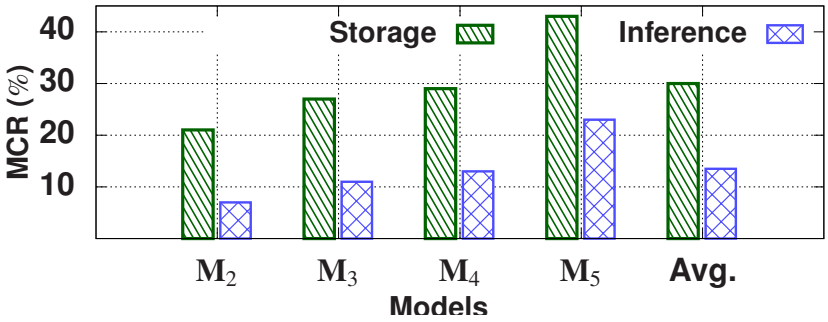


Figure 4.5: Memory consumption ratio of different models ($\mathbf{M}_1 - \mathbf{M}_5$).

4.5 Conclusions and Future Work

In this chapter, we have presented a secure and intelligent patient monitoring system that has used federated learning in IoMT. Apart from the existing work, we have partitioned the participants as per the availability of bandwidth, size of the dataset, and freshness of the dataset. Next, we have generated a large-size DNN suitable for patient activities monitoring at the server. It satisfied the parameters of participants in cluster

number one, arranged in decreasing order of their available parameters. Further, the model on the server is iteratively compressed using KD and deployed on the participants of different clusters to perform FL. Finally, the experimental results have depicted the proposed patient monitoring system's effectiveness on publicly available and collected datasets. In the future, we can also handle different issues such as noisy labels in the training data for patient activities monitoring. We also plan to consider different types of challenges in datasets, such as imbalanced, noisy, and unseen classes.