Future Generation Computer Systems 174 (2026) 107911

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Data distribution aware clustering for parallel split learning in healthcare applications

Md. Tanvir Arafat [a], Md. Abdur Razzaque [a], Abdulhameed Alelaiwi [b], Md. Zia Uddin [c], Mohammad Mehedi Hassan [d,*]

[a] Department of Computer Science and Engineering, University of Dhaka, Dhaka 1000, Bangladesh
[b] Department of Software Engineering, King Saud University, Riyadh, Saudi Arabia
[c] SINTEF Digital, Oslo, Norway
[d] Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

ARTICLE INFO

Keywords:
Parallel split learning
Healthcare applications
Clustering
Proximal Policy Optimization (PPO)
Deep Reinforcement Learning (DRL)

ABSTRACT

Split learning, a promising approach in privacy-preserving machine learning, decentralizes model training by dividing it among client devices and a central server. However, split learning has exhibited a certain level of slowness in its vanilla approach, mainly due to the serial processing of devices. Recent research endeavors have addressed this challenge by introducing parallelism and thus accelerating the split learning process. However, the existing split learning methodologies often overlook the critical aspect of data distribution among client devices.

This paper introduces a Data Distribution Aware Clustering-based Parallel Split Learning (DCSL), a scheme purposefully crafted to address the complexities stemming from non-identically and non-independently distributed (non-IID) data among client devices engaged in the split learning paradigm. In healthcare applications, comprehending the intricacies of data distribution is imperative, particularly given the non-IID nature of medical datasets, to ensure accurate analysis and decision-making. The DCSL leverages a novel clustering technique to create clusters of medical client devices, considering the data distributions of their local datasets, and employs parallel model training within the device clusters. It enhances model convergence and reduces training latency by optimizing the cluster formation. Extensive experiments demonstrate that DCSL outperforms traditional split learning approaches, significantly improving accuracy and reducing training latency across various applications.

1. Introduction

In the modern era, the rapid proliferation of Internet of Things (IoT) devices, alongside sophisticated sensing technologies, generates vast amounts of mobile data daily. This surge in data is effectively managed using advanced artificial intelligence (AI) techniques like deep neural networks (DNNs), enhancing applications from audio recognition to disease classification [1,2]. However, the traditional centralized learning model, which pools raw data from these devices, often faces significant obstacles due to strict privacy laws protecting user data [3,4].

Federated Learning (FL) has emerged as a prominent collaborative learning paradigm to address privacy concerns. FL enables devices to train models locally, sharing only model updates with a central server, thus preserving user data privacy. However, FL has significant limitations, particularly for resource-constrained devices such as wearables and IoT sensors. These devices often lack the computational power and memory to train and update large, complex models like deep neural networks, as it requires substantial resources for both training and communication. Additionally, the process of transmitting full model updates to the server introduces considerable communication overhead, which can be impractical in bandwidth-constrained environments. These challenges hinder FL's scalability and efficiency, especially in applications requiring low-latency, resource efficiency, and robust performance involving resource-constrained devices.

Emerging as a collaborative learning alternative, split learning (SL) addresses the challenges of centralized learning and FL. As depicted in Fig. 1, SL divides an AI model into a device-side model on local devices

* Corresponding author.
E-mail addresses: tanvirarafat.2316.csedu@gmail.com (M.T. Arafat), razzaque@du.ac.bd (M.A. Razzaque), aalelaiwi@ksu.edu.sa (A. Alelaiwi), zia.uddin@sintef.no (M.Z. Uddin), mmhassan@ksu.edu.sa (M.M. Hassan).

https://doi.org/10.1016/j.future.2025.107911
Received 19 September 2024; Received in revised form 9 May 2025; Accepted 11 May 2025
Available online 9 June 2025

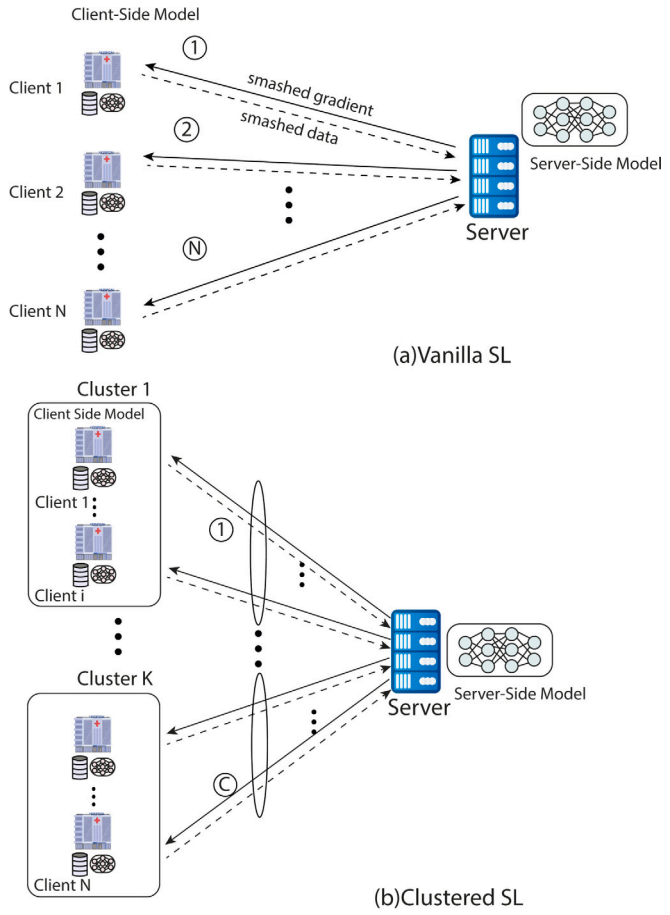


Fig. 1. (a) Vanilla split learning (b) clustered split learning.

and a server-side model on edge server(s). The process begins with the local device running its part of the model, sending the intermediate output (smashed data) from the predefined cut layer to the edge server. The server completes the forward propagation, updates the model, and returns the gradient of the smashed data to the device for backward propagation. This cycle repeats across devices, ensuring each device is trained without compromising data privacy.

In SL, exchanging compressed device-side models, smashed data, and gradients notably cuts communication overhead compared to FL [5], where the full AI model is uploaded. This approach also lightens device computational load since each only trains a model segment. However, employing standard SL [6] in multi-device settings introduces challenges, such as increased training latency resulting from the serialization of the training process, especially with many devices involved.

To mitigate latency in SL, more efficient training strategies are essential. Approaches like SplitFed [7] and Cluster-based Parallel Split Learning (CPSL) [8] have been developed. SplitFed combines SL and FL to reduce latency, while CPSL introduces a low-latency method that clusters devices for parallel training. In CPSL, devices within a cluster train their models in parallel based on local data, while the edge server simultaneously trains the server-side model using concatenated smashed data from all devices in the cluster. This approach reduces latency by enabling concurrent training across clusters. However, existing SL approaches often assume uniform data distributions, neglecting the challenges posed by non-IID data, which remain a critical bottleneck in ensuring robust and accurate model performance. This limitation highlights a clear research gap, motivating the need for enhanced SL frameworks that address both latency and data distribution diversity.

The quality and distribution of data are critical in machine learning (ML), with data ideally being Identically and Independently Distributed (IID) to ensure accuracy. Deviations from IID can lead to skewed results, biases, and overfitting, which are particularly problematic in deep learning models like CNNs, RNNs, and GANs. In healthcare, especially in disease classification, non-IID data is common due to variations in patient demographics, geographic locations, and medical practices. For example, health data for skin disease detection like melanoma varies significantly between regions with different sun exposure levels, affecting the assumptions about data uniformity. Such diversity challenges the effectiveness of ML models trained on data from specific regions, as they may not perform well universally. Managing non-IID data is crucial for the robustness and reliability of healthcare ML models, ensuring they are adaptable and accurate across varied conditions.

Motivated by the challenges of non-IID data, we introduce the Data Distribution-Aware Clustering-Based Split Learning (DCSL) framework. Building on the low-latency techniques of CPSL, DCSL starts by grouping devices (e.g., hospitals, clinics and other healthcare entities) into clusters based on their data distribution characteristics. The edge server then trains using aggregated 'smashed data' from all devices within a cluster, ensuring parallelism. This clustering strategy aims to mimic IID conditions within the aggregated datasets of different clusters, enhancing model accuracy and stability despite the inherent non-IID nature of the data among participating devices. Our main contributions are summarized as follows:

- We propose a clustering approach for grouping participating devices in clustered parallel split learning, with the goal of mitigating heterogeneity in the distribution of data across these clusters.
- We formulate the aforementioned clustering problem as a binary integer nonlinear programming (BINLP) problem
- To address the NP-hardness of the optimization problem, we develop a PPO (Proximal Policy Optimization)-based DRL (Deep Reinforcement Learning) method to realize an approximate solution to the clustering problem while enhancing scalability and minimizing complexity.
- A numerical assessment of the proposed mechanism through simulation is conducted and benchmarked against state-of-the-art methods to evaluate performance in terms of training latency and accuracy.

The rest of this paper is organized as follows. Section 2 reviews the related work. The system model is described in Section 3. In Section 4, the clustering problem in the SL scenario is formulated as BINLP problem. The PPO based DRL method is described in Section 5. In Section 6, the performance of the proposed DCSL scheme is compared with other benchmarks. Section 7 provides a discussion on the results and their implications. Finally, Section 8 concludes the paper with future directions.

2. Related work

Federated Learning (FL), introduced by Google [9], is a decentralized learning framework designed to address privacy concerns by enabling model training without exposing clients' raw data. While FL offers strong privacy guarantees, it faces substantial challenges when dealing with non-IID client data, which can significantly degrade model performance. As noted in [9], mitigating non-IID effects is feasible but comes at the cost of increased training time and bandwidth consumption.

To address these challenges, various strategies have been proposed. Zhao et al. [10] have introduced a booster model pretrained on aggregated client data to balance distributions, though this approach increases privacy risks and computational overhead due to data duplication. Briggs et al. [11] have proposed hierarchical clustering, grouping clients by model similarity to reduce heterogeneity; however,

this transforms FL into multiple independent sub-networks, limiting scalability. Similarly, Chen et al. [12] have explored sequential clustering based on geographic or other parameters, training clients within clusters before aggregation, yet risking biased updates due to a lack of data shuffling, which negatively impacts global convergence.

Beyond clustering, client selection strategies have been investigated to address non-IID challenges. Luping et al. [13] have developed a relevance-based selection approach, weighting client contributions by alignment with the global model's update direction. Kang et al. [14] have introduced an incentive-driven framework rewarding clients with larger datasets or greater computational power, though this prioritizes high-resource clients at the expense of data diversity. Duan et al. [15] have proposed mediation-based selection, balancing class distributions via up-sampling or down-sampling, but this raises privacy concerns as it requires direct access to client data distributions. Yang et al. [16] have addressed this by estimating class distributions using model weights and Kullback–Leibler (KL) divergence, combined with a combinatorial multi-armed bandit (CMAB) algorithm to optimize client selection while preserving dataset diversity.

In parallel, regularization techniques have been investigated to improve model stability in non-IID settings. Li et al. [17] have proposed FedProx, which integrates a proximal term into local training to constrain deviations from the global model. Reddi et al. [18] have introduced adaptive server-side learning rates to mitigate the impact of noisy updates. Although these methods enhance model robustness, they introduce additional computational demands, making them less suitable for IoT and resource-constrained environments.

While FL has been widely studied, Split Learning (SL) has emerged as an alternative privacy-preserving approach, particularly for applications requiring strict data confidentiality, such as healthcare. Gupta et al. [6] have introduced SL as a framework where models are partitioned between client and server, allowing collaborative training without raw data exchange. Due to its sequential architecture, SL can maintain higher accuracy under non-IID data compared to FL but suffers from increased latency due to synchronization requirements.

To overcome the respective limitations of FL and SL, hybrid FL–SL frameworks have been developed. For example, Thapa et al. [7] have proposed SplitFed, combining parallel FL-style updates with server-side processing of deeper model layers. While this improves communication efficiency, it inherits FL's challenges with non-IID data and straggler effects. Several follow-up works have focused on enhancing computational efficiency: Wu et al. [8] have introduced Cluster-based Parallel SL (CPSL), grouping clients with similar training speeds and dynamically selecting the cut layer; Wu et al. [19] have employed reinforcement learning to determine optimal cut layers for heterogeneous clients; Samikwa et al. [20] have developed ARES, an adaptive split-point mechanism based on network and resource conditions; Lin et al. [21] have presented Efficient Parallel Split Learning (EPSL) tailored for edge computing; and Tirana et al. [22] have proposed Multihop Parallel SL (MP-SL), a modular framework facilitating learning across low-resource devices. While these innovations have reduced training latency, they have largely overlooked the impact of non-IID data in SL architectures.

Recent studies have begun addressing non-IID issues specifically in SL. Cai et al. [23] have proposed an incentive mechanism that rewards users based on dataset size and uniformity, though such schemes may be impractical in domains like healthcare where data is inherently heterogeneous. Arafeh et al. [24] have introduced a two-stage clustering scheme, first grouping clients into IID clusters for sequential training and then grouping by computational capacity to reduce straggler effects. However, their method relies on multiple split and main servers, leading to high operational costs and substantial communication overhead due to frequent weight sharing.

In contrast to these approaches, our work focuses on an SL framework specifically designed to address data distribution challenges. Recognizing the inherent heterogeneity across clients, we propose a clustering algorithm that strategically groups participating devices to improve the IID properties and uniformity of the combined datasets within each cluster. This enhances the effectiveness of clustered parallel SL without compromising training latency or introducing excessive communication overhead. By addressing both data heterogeneity and computational efficiency, our framework offers a balanced and scalable solution for distributed learning in resource-constrained environments.

Table 1
Notations.

Symbol	Definition
D, C, $\mathcal{L}$	Set of devices, set of clusters, set of class labels
D_c	Set of devices in cluster c
w^i, W	Device-side model of device i, Server-side model
$\mathcal{W}$	Complete Deep Learning model
d_i	Dataset at device i
δ^i, δ'^i	The dataset information and data distribution of d_i
n_m^i	Number of data samples in d_i of class label m
n_m^c	Number of data samples in cluster c of class label m
$\mathcal{R}$, $\mathcal{E}$	Set of training rounds, Set of local Epochs
$w_i^{\rho,e}$	Device-side model of device i at epoch e of round ρ
$\mathcal{W}^{\rho,e}$	Server-side model at epoch e of round ρ
α_s, α_i	Learning rate of server, Learning rate of device i
$\nabla L^{\rho,e}(.)$	Gradient of the loss $L(.)$ at epoch e of round ρ
$\nabla l_i^{\rho,e}$	Smashed gradient of device i at epoch e of round ρ
$\tilde{w}_c^{\rho}$	Average of device-side models in cluster c at round ρ
$b_i^{\rho,e}$	Minibatch of data from d_i, at epoch e of round ρ
$\sigma_i^{\rho,e}$	Smashed data from device i, at epoch e of round ρ
$S_c^{\rho,e}$	Smashed data for cluster c, at epoch e of round ρ
P, Q^c	Population dataset, Dataset of a cluster $c \in C$
p, q^c	Distribution of P and Q^c
$D_{KL}(p\|\|q^c)$	KL divergence between distribution p and q^c
μ	Summation of KL divergence of all clusters $c \in C$
θ, ω	Actor network, Critic network

3. System model and assumptions

In this section, we provide a concise overview of the entities participating in our system and state the fundamental assumptions associated with them (see Table 1).

3.1. Entities involved

As shown in Fig. 2, we examine a common scenario in a wireless network involving a server and multiple medical client devices, such as hospitals, clinics, and research centers, which store local medical data. The edge server, equipped with computational and communication resources, conducts server-side model training using a model denoted as W. The server also gathers information about the data distribution of client devices to facilitate cluster formation.

The ensemble of devices is denoted by D, where $i \in D$ represents a single device. Each device has computational capabilities for local model training and is equipped with a device-specific model, w^i. The complete deep learning model is denoted as $\mathcal{W} = \{w^i; W\}$. The ensemble of device clusters is denoted by C, with $c \in C$ representing a single cluster. The set of class labels is denoted by $\mathcal{L}$, where $m \in \mathcal{L}$ represents a particular class label. The set of devices in cluster c is denoted by D_c. The dataset information δ^i for dataset d_i on device i is expressed as a tuple,

$$\delta^i = < n_1^i, n_2^i, \ldots, n_m^i, \ldots, n_{|\mathcal{L}|}^i >, \tag{1}$$

where, n_m^i denotes the number of data samples in dataset d_i with class label m. The data distribution of dataset d_i is represented as a

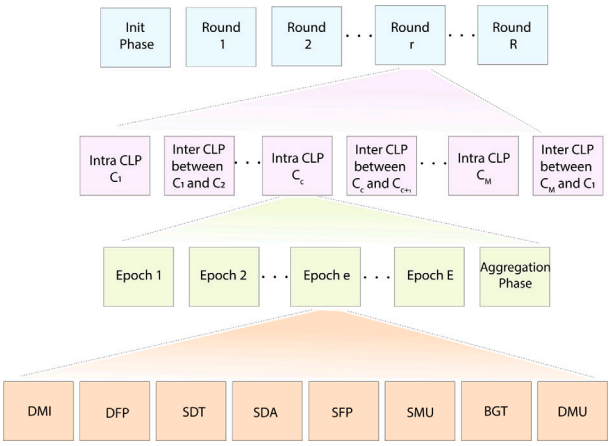


Fig. 2. Model training process in DCSL.

probability distribution over class labels $m \in \mathcal{L}$, denoted by $\delta'^i = < n'^i_1, n'^i_2, \ldots, n'^i_m, \ldots, n'^i_{|\mathcal{L}|} >$, where n'^i_m is the probability that a randomly selected data sample from dataset d_i belongs to class m. This probability is calculated as $n'^i_m = \frac{n^i_m}{|d_i|}$, where $|d_i|$ is the total number of data samples in dataset d_i. With a description of the participating entities, we now illustrate the interactions among these entities within the DCSL framework.

3.2. Interaction among entities in DCSL scheme

The Model Training Process in DCSL, as depicted in Fig. 2, follows a structured hierarchical framework comprising R training rounds. It begins with an Initialization Phase, where devices are systematically clustered based on dataset characteristics using Algorithm 2. Each training round consists of both the Intra-Cluster Learning Phase (Intra CLP) and the Inter-Cluster Learning Phase (Inter CLP). Within Intra CLP, devices in each cluster train their local models over E local epochs, with each epoch comprising eight distinct steps — DMI, DFP, SDT, SDA, SFP, SMU, BGT, and DMU — as elaborated in Section 3.2.2. Upon completion of the E local epochs, device-side models are transmitted to the server and aggregated. This is followed by the Inter CLP, wherein the aggregated device-side models are sequentially transferred across clusters to further enhance global learning. This parallel-sequential training methodology optimizes computational efficiency, minimizes communication overhead, and enhances model generalization. The subsequent sections provide a comprehensive analysis of each phase and its underlying processes.

3.2.1. Initialization phase

In the initialization phase, the server gathers the dataset information δ^i from all participating devices $i \in D$ and uses a clustering approach (discussed later in Section 4) to group these devices. This information is obtained through privacy-preserving schemes to ensure confidentiality [25]. After clustering, the server communicates the clustering decisions to the client devices.

After the initialization phase, the DCSL framework proceeds through multiple training rounds until convergence, aiming to find the optimal model parameters. It is assumed that all devices maintain stable connections and consistent computational capabilities throughout each training round, denoted by $\rho \in \mathcal{R} = \{1, 2, \ldots, R\}$. During each round, both intra-cluster and inter-cluster learning stages are executed.

3.2.2. Intra-cluster learning phase

During the intra-cluster learning phases, all devices within a cluster simultaneously update their device-side models while interacting with the server. The steps of an intra-cluster learning phase are detailed below.

Device-side model initialization (DMI). In this step, the server disseminates the initial device-side model, represented as $w_i^{\rho,1}$, to all devices i within cluster c. The model undergoes training for E local epochs, indexed by $e \in \mathcal{E} = \{1, 2, \ldots, E\}$. The device-side model parameters for device i at epoch e during training round ρ are denoted as $w_i^{\rho,e}$

$$w_i^{\rho,1} \leftarrow \hat{w}_c^{\rho}, \quad \forall i \in D_c \tag{2}$$

In the initial local epoch ($e = 1$), all devices i in cluster c synchronize their device-side models with the broadcasted model $\hat{w}_c^{\rho}$ as shown in Eq. (2).

Device-side forward propagation (DFP) and smashed data transmission (SDT). This step involves forward propagation (FP) and smashed data transmission at each local epoch $e \in \mathcal{E}$. Initially, each device within the cluster performs forward propagation by drawing a mini-batch of data samples, $b_i^{\rho,e}$, from its local dataset d_i at epoch e of round ρ. The model processes this data until the cut layer, producing smashed data as shown in Eq. (3).

$$\sigma_i^{\rho,e} = f(b_i^{\rho,e} | w_i^{\rho,e}), \quad \forall i \in D_c. \tag{3}$$

Here, $\sigma_i^{\rho,e}$ represents the smashed data at device i at epoch e of round ρ, and $f(.)$ is the forward propagation function. Thus, given the device-side model $w_i^{\rho,e}$, the smashed data $\sigma_i^{\rho,e}$ is generated by feeding the dataset $b_i^{\rho,e}$ into $f(.)$. This smashed data is then transmitted to the server, where it is aggregated.

Smashed data aggregation (SDA) and server-side forward propagation (SFP). This step includes the aggregation of smashed data and forward propagation (FP) on the server side. The server collects smashed data from all devices i within cluster c and concatenates it as shown in Eq. (4).

$$S_c^{\rho,e} = \cup_{i \in D_c} \sigma_i^{\rho,e} \tag{4}$$

Here, $S_c^{\rho,e}$ represents the aggregated smashed data on the server at epoch e of round ρ for cluster c. The concatenated smashed data is then used for the server-side model's forward propagation as shown in Eq. (5).

$$\tilde{y}_c^{\rho,e} = f(S_c^{\rho,e} | \mathcal{W}^{\rho,e}), \tag{5}$$

where $\tilde{y}_c^{\rho,e}$ is the predicted outcome or class label at the server at epoch e of round ρ, and $\mathcal{W}^{\rho,e}$ is the server-side model. One careful observation here is, that the loss is calculated on the concatenated smashed data, i.e. $S_c^{\rho,e}$. So even if batches $b_i^{\rho,e}$ from individual clients are non-IID, we can cluster the devices in such a way that there is IID property maintained as much among $S_c^{\rho,e}, \forall c \in C$.

Server-side model update (SMU) and backward gradients transmission (BGT). This stage involves updating the server-side model through backpropagation and transmitting backward gradients to client devices. Given the predicted outcomes and their actual labels, the gradient of the loss function $L(\tilde{y}_c^{\rho,e} | y_c^{\rho,e})$ is computed. The server-side model is then updated using the gradient descent (GD) method as shown in Eq. (6).

$$W^{\rho,e+1} = W^{\rho,e} - \alpha_s \times \nabla L(\tilde{y}_c^{\rho,e} | y_c^{\rho,e}) \tag{6}$$

Here, α_s is the server's learning rate, and $\nabla L(\tilde{y}_c^{\rho,e} | y_c^{\rho,e})$ represents the gradient of the loss function $L(.)$. The loss function $L(.)$ is calculated from the predicted outcomes $\tilde{y}_c^{\rho,e}$ for cluster c at epoch e of round ρ, given the actual outcomes $y_c^{\rho,e}$.

The model parameters are updated layer by layer, starting from the final layer and moving towards the cut layer, following the chain rule for gradient computation. When the gradient calculation reaches the cut layer, the gradient of the data samples, known as the smashed data's gradient, is transmitted back to the corresponding device.

Device-side model update (DMU). This step involves receiving the gradient of the smashed data and updating the client-side model through backpropagation. Upon receiving the gradient from the server, each device updates its model using the gradient descent method, starting from the cut layer and moving towards the input layer, as shown in Eq. (7).

$$w_i^{\rho,e+1} = w_i^{\rho,e} - \alpha_i \times \nabla l_i^{\rho,e}, \quad \forall i \in D_c \tag{7}$$

Here, α_i is the learning rate of device i, and $\nabla l_i^{\rho,e}$ represents the gradient of the loss function $l(.)$ for device i at epoch e of round ρ. This completes one full forward–backward propagation cycle across the entire model.

Aggregation of device-side models. After E epochs, the device-side models are aggregated across all participating devices within a cluster. Once every device has updated its model, the trained device-side models are sent to the server and combined using the Federated Averaging (FedAvg) [9] algorithm, as shown in Eq. (8).

$$\tilde{w}_c^{\rho} = \frac{\sum_{i \in D_c} |d_i| \times w_i^{\rho,E+1}}{\sum_{i \in D_c} |d_i|} \tag{8}$$

The aggregated device-side model for cluster c at round ρ is $\tilde{w}_c^{\rho}$, a weighted average of the device-side parameters at the end of round ρ, $w_i^{\rho,E+1}$, for all $i \in D_c$. This aggregation step is essential because each device-side model is updated based only on its own gradient of the smashed data, resulting in differences among the models. The FedAvg algorithm consolidates these models, accounting for the number of data samples $|d_i|$ each device i has, to create a unified model that reflects contributions from all devices within the cluster.

3.2.3. Inter-cluster learning phase

This phase involves transferring the amalgamated device-side model from one cluster to the next to continue the training process. After creating the combined device-side model $\tilde{w}_c^{\rho}$ for cluster c, the server disseminates the updated model to the devices in the next cluster $(c+1)$, as shown in Eq. (9)

$$\hat{w}_{c+1}^{\rho} \leftarrow \tilde{w}_c^{\rho} \tag{9}$$

This $\hat{w}_{c+1}^{\rho}$ initializes the device-side parameters for all devices in the next cluster $(c+1)$, according to Eq. (10).

$$w_i^{\rho,1} \leftarrow \hat{w}_{c+1}^{\rho}, \quad \forall i \in D_{c+1} \tag{10}$$

The inter-cluster learning stage then sequentially progresses across clusters, similar to vanilla SL. After completing the intra-cluster and inter-cluster learning steps for all clusters, one complete training round is finished.

It is important to note that, unlike the conventional SL scheme that operates sequentially, the proposed DCSL adopts a "first-parallel-then-sequential" approach. In DCSL, devices within each cluster train in parallel, while clusters train sequentially. This method streamlines the training process and reduces latency. The steps of the DCSL scheme are summarized in Algorithm 1.

In the first step, the server uses Algorithm 2 to partition devices into clusters (line 1) and broadcasts the clustering decision to all devices $i \in D$ (line 2). During lines 6–10, devices within cluster c, denoted as $i \in D_c$, concurrently draw data samples $b_i^{\rho,e}$ from their datasets d_i. Each device processes its data using Eq. (3) to obtain smashed data $\sigma_i^{\rho,e}$ and sends it to the server during epoch e of round ρ. The server concatenates the smashed data $\sigma_i^{\rho,e}$ from devices $i \in D_c$ (line 11) to generate aggregated smashed data $S_c^{\rho,e}$ for cluster c. In line 12, the server makes predictions $\tilde{y}_c^{\rho,e}$ via Eq. (5) and updates the server-side model $\mathcal{W}^{\rho,e+1}$ using backpropagation as per Eq. (6) (line 13). The server then distributes the gradient $\nabla L_i^{\rho,e}$ to devices in cluster c (line 14). Lines 15–17 involve parallel updates of device-side models $w_i^{\rho,e+1}$ for all devices $i \in D_c$ using Eq. (7), repeated for E epochs. In lines 19–21, device-side models $w_i^{\rho,E+1}$ are uploaded to the server, which aggregates them using the FedAvg algorithm (Eq. (8)) at line 22. The parameters $\hat{w}_{c+1}^{\rho}$ are used to initialize the devices $i \in D_{c+1}$ as per Eq. (9). The process from lines 5 to 23 is repeated for each cluster $c \in C$ and for R training rounds, as shown in lines 4–24.

Algorithm 1 Data Distribution Aware Clustered SL: DCSL

```
1:  Server partitions devices into clusters by Alg. 2
2:  Server broadcasts clustering decision to ∀i ∈ D
3:  for training round ρ = 1, 2, ..., R do
4:      for cluster c ∈ C do
5:          for epoch e = 1, 2, ..., E do
6:              for ∀i ∈ D_c in parallel do
7:                  Draw b_i^{ρ,e} from the dataset d_i randomly
8:                  Get smashed data σ_i^{ρ,e} by Eq. (3)
9:                  Transmit smashed data σ_i^{ρ,e} to server
10:             end for
11:             Get S_c^{ρ,e} by concatenating σ_i^{ρ,e}, ∀i ∈ D_c
12:             Get predictions ỹ_c^{ρ,e} by Eq. (5)
13:             Update Server-side model W^{ρ,e+1} by Eq. (6)
14:             Server disseminates ∇l_i^{ρ,e} to ∀i ∈ D_c
15:             for ∀i ∈ D_c in parallel do
16:                 Update Device-side model w_i^{ρ,e+1} by Eq. (7)
17:             end for
18:         end for
19:         for ∀i ∈ D_c in parallel do
20:             Upload w_i^{ρ,E+1} to server
21:         end for
22:         Server Gets w̃_c^ρ by Eq. (8)
23:         Server Gets ŵ_{c+1}^ρ by Eq. (9)
24:     end for
25: end for
```

4. Design of data distribution aware clustering

In machine learning, the goal is to minimize the average training loss over the entire dataset P, which is the union of all local datasets d_i from clients $i \in D$, expressed as $P = \cup_{i \in D} d_i$. The optimization problem is:

$$\underset{\mathcal{W}}{minimize} \quad \{L(P) = \frac{1}{|P|} \sum_{\xi \in P} L(\xi|\mathcal{W})\} \tag{11}$$

where $L(.)$ is the loss function, $L(P)$ is the average training loss over P, $\mathcal{W}$ represents the model parameters, and ξ is a data sample from P. In distributed learning, such as SL or FL, accessing the entire data population P is restricted due to privacy concerns. The training loss $L(d_i)$ on the local dataset d_i of device $i \in D$ is:

$$L(d_i) := \mathbb{E}_{\xi_i \sim d_i}[L(\xi_i|\mathcal{W})] = \frac{1}{|d_i|} \sum_{\xi \in d_i} L(\xi|\mathcal{W}), \quad \forall i \in D \tag{12}$$

where ξ_i is a sample from d_i, and $\mathbb{E}_{\xi_i \sim d_i}[L(\xi_i|\mathcal{W})]$ is the expected loss.

Training a model through gradient descent, such as in a DNN, involves computing the gradient of the training loss with respect to the model parameters $\mathcal{W}$. When the data distribution among participating devices is IID, meaning each device's dataset d_i closely resembles the population dataset P, the stochastic gradient $\Delta[L(\xi_i|\mathcal{W})]$ at any device i serves as a conditionally unbiased estimator of the gradient of the training loss $L(P)$ over the entire population P. This relationship is expressed as:

$$\mathbb{E}[\Delta L(\xi_i|\mathcal{W})] = \Delta L(d_i) = \Delta L(P) \tag{13}$$

However, in non-IID scenarios, where each device's data distribution differs from the population dataset P, the stochastic gradient $\Delta[L(\xi_i|\mathcal{W})]$ at any device i becomes a biased estimator of the gradient of $L(P)$:

$$\mathbb{E}[\Delta L(\xi_i|\mathcal{W})] = \Delta L(d_i) \neq \Delta L(P) \tag{14}$$

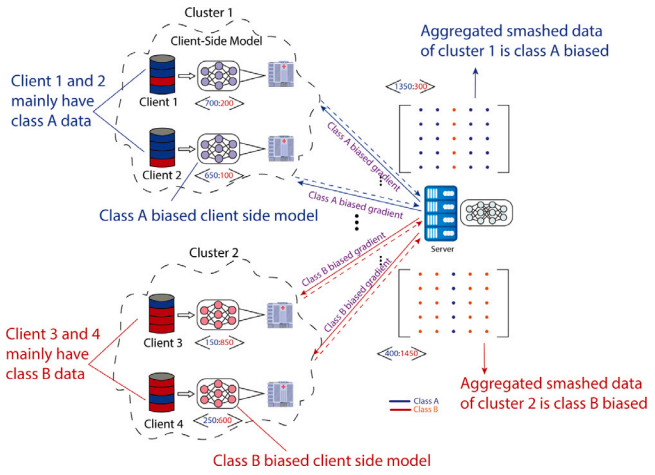


Fig. 3. Data distribution unware clustering.

For effective gradient descent across multiple training iterations, the stochastic gradient $\Delta L(\xi_i|\mathcal{W})$ must accurately reflect the true gradient over the entire population. This requires the minibatch to be sampled from the population distribution, not just any identical distribution, to ensure reliable optimization. In the DCSL framework, during intra-cluster learning, each device extracts a minibatch from its local dataset, performs forward propagation, and sends the smashed data to the server. The server aggregates this data and computes the loss on the combined dataset Q^c of cluster $c \in C$.

The dataset of a cluster $c \in C$ is represented by Q^c, formed by combining the datasets d_i of all devices within the cluster: $Q^c = \cup_{i \in D_c} d_i$. The training loss $L(Q^c)$ associated with the aggregated dataset Q^c for cluster c is:

$$L(Q^c) := \mathbb{E}_{\xi_c \sim Q^c}[L(\xi_c|\mathcal{W})] = \frac{1}{|Q^c|} \sum_{\xi \in Q^c} L(\xi|\mathcal{W}), \quad \forall c \in C. \tag{15}$$

Here, ξ_c is a randomly chosen data sample from Q^c, and $\mathbb{E}_{\xi_c \sim Q^c}[L(\xi_c|\mathcal{W})]$ denotes the expected loss on ξ_c. The goal of clustering in the DCSL framework is to ensure that the data distribution of Q^c for each cluster $c \in C$ closely mirrors the population dataset P. When this condition is met, the stochastic gradient $\Delta[L(\xi_c|\mathcal{W})]$ for any cluster $c \in C$ serves as an unbiased estimator of the gradient of the training loss $L(P)$ across the entire population:

$$\mathbb{E}[\Delta L(\xi_c|\mathcal{W})] = \Delta L(Q^c) = \Delta L(P), \quad \forall c \in C. \tag{16}$$

Thus, even with non-IID data across devices, clustering to satisfy Eq. (16) ensures that the gradient computed over a minibatch accurately represents the true gradient over the entire population. This keeps the learned model unbiased and unaffected by data heterogeneity. In the DCSL framework, the goal is to optimally cluster devices so that the aggregated datasets of different clusters approximate the IID property. Additionally, label skewness within a cluster's aggregated dataset should be minimized to avoid bias towards particular class labels.

As illustrated in Fig. 3, the data distribution across client devices in a hospital environment varies significantly, leading to a non-IID setting. Each client holds a local dataset with an imbalanced representation of different classes. For example, client 1 has 700 samples from class A and 200 from Class B, resulting in a distribution of $< 700 : 200 >$, with selection probabilities of 0.77 for class A and 0.23 for class B. Similarly, clients 2, 3, and 4 have distributions of $< 650 : 100 >$, $< 150 : 850 >$, and $< 250 : 600 >$, with corresponding probabilities of (0.87, 0.13), (0.15, 0.85), and (0.29, 0.71), respectively. Although the overall dataset, when aggregated across all clients, results in a balanced distribution of $< 1750 : 1750 >$ (0.5 probability for both classes), the local variations introduce bias at the cluster level.

Consider the clustering structure depicted in Fig. 2. If clients 1 and 2 are grouped into cluster 1, their combined dataset forms a skewed distribution of $< 1350 : 300 >$, predominantly favoring class A. Similarly, cluster 2, composed of clients 3 and 4, exhibits a bias towards class B with a distribution of $< 400 : 1450 >$. Consequently, the smashed data $S_1^{\rho,e}$ and $S_2^{\rho,e}$, which are aggregated at the server, inherit these class biases. This bias in smashed data leads to biased loss estimations during server-side training, as the class proportions deviate significantly from the overall population distribution. As a result, the backward gradients become skewed, reinforcing the biases in device-side models—cluster 1's models are biased towards class A, while cluster 2's models favor class B. These local models deviate substantially from those trained on a globally balanced dataset, thus limiting generalization.

To mitigate this issue, an alternative clustering approach can be employed. By pairing clients 1 and 3 in one cluster and clients 2 and 4 in another, the resulting distributions better approximate the global data distribution. This restructuring reduces bias in smashed data, leading to more representative gradient updates and, consequently, model parameters that closely resemble those trained on the entire dataset.

Thus, the primary objective is to design a clustering strategy that minimizes the discrepancy between cluster-wise data distributions and the overall population distribution, ensuring that training remains unbiased and representative of the entire dataset.

At this point, we formulate the optimal clustering problem as an Integer Non-Linear Programming (INLP) problem. The decision variable x_c^i represents a binary decision, where $x_c^i \in \{0, 1\}$. If $x_c^i = 1$, device $i \in D$ is assigned to cluster $c \in C$, and vice versa.

As described earlier, the dataset of a cluster $c \in C$ is denoted as Q^c. Additionally, η_m^c denotes the total number of data samples within cluster c that belong to class label m:

$$\eta_m^c = \sum_{i \in D} n_m^i \times x_c^i, \quad \forall m \in \mathcal{L}, \forall c \in C. \tag{17}$$

Here, n_m^i represents the total number of data samples in the dataset d_i of device i with class label m. The total number of records N^c for all class labels $m \in \mathcal{L}$ in the aggregated dataset Q^c of cluster c is given by:

$$N^c = \sum_{m \in \mathcal{L}} \eta_m^c, \quad \forall c \in C. \tag{18}$$

The data distribution of the aggregated dataset Q^c of cluster c is represented by the tuple $q^c =< q_1^c, q_2^c, \ldots, q_m^c, \ldots, q_{|\mathcal{L}|}^c >$, where q_m^c is the probability of a randomly selected record from Q^c belonging to class m:

$$q_m^c = \frac{\eta_m^c}{N^c}, \quad \forall m \in \mathcal{L}, \forall c \in C. \tag{19}$$

The data distribution of the population dataset P is represented by the tuple $p =< p_1, p_2, \ldots, p_m, \ldots, p_{|\mathcal{L}|} >$, where p_m is the probability that a record randomly selected from P belongs to class m:

$$p_m = \frac{\eta_m}{N}, \quad \forall m \in \mathcal{L} \tag{20}$$

where η_m is the number of data samples in P with class label m, given by $\eta_m = \sum_{c \in C} \eta_m^c$, and N is the total number of data samples in P, given by $N = \sum_{c \in C} N^c$.

As discussed earlier, to mitigate the adverse effects of non-IID data on training in the DCSL scheme, it is desirable for the data distribution q^c of individual clusters $c \in C$ to closely approximate the data distribution p of the entire population dataset P. An effective statistical measure for calculating the deviation between q^c and p is the Kullback–Leibler (KL) divergence, denoted as $D_{KL}(a \parallel b)$ [26]. KL divergence quantifies how one probability distribution b diverges from another reference probability distribution a. Essentially, it measures the expected additional deviation when b is incorrectly assumed to be the model for the actual distribution a.

In DCSL, the reference distribution a is the population data distribution p, and the sample distribution b is the data distribution q^c of individual clusters $c \in C$. The KL divergence $D_{KL}(p \parallel q^c)$ between the cluster data distribution q^c and the population data distribution p is given by the difference between the cross-entropy of p and q^c and the entropy of p:

$$D_{KL}(p \parallel q^c) = H(p, q^c) - H(p), \quad \forall c \in C. \tag{21}$$

where, $H(p, q^c)$ is the cross entropy between distribution p and q^c, and is given by

$$H(p, q^c) = -[\sum_{m \in \mathcal{L}} p_m \log(q_m^c)], \quad \forall c \in C \tag{22}$$

and, $H(p)$ is the entropy of the distribution p, which is expressed as,

$$H(p) = -[\sum_{m \in \mathcal{L}} p_m \log(p_m)], \forall c \in C. \tag{23}$$

From Eq. 8, (22) and (23), the KL divergence $D_{KL}(p \parallel q^c)$ between p and q^c can be written as,

$$D_{KL}(p \parallel q^c) = -[\sum_{m \in \mathcal{L}} p_m \log(\frac{q_m^c}{p_m})], \forall c \in C. \tag{24}$$

The summation of Kullback–Leibler divergence of all clusters $c \in C$ denoted as μ and calculated as

$$\mu = (\sum_{c \in C} D_{KL}(p, q^c)). \tag{25}$$

Also, the average KL divergence of all clusters $c \in C$ is denoted as $\bar{D}_{KL}$ and is given by

$$\bar{D}_{KL} = \frac{\sum_{c \in C} D_{KL}(p, q^c)}{|C|}. \tag{26}$$

And the variation of KL divergence for all the clusters $c \in C$ is denoted as s^2_{KL},

$$s^2_{KL} = \frac{\sum_{c \in C} (D_{KL}(p, q^c) - \bar{D}_{KL})^2}{|C| - 1}. \tag{27}$$

To this end, we define the optimal device clustering problem within the DCSL framework as minimizing the summation of Kullback–Leibler divergence across all clusters $c \in C$, described as follows::

$$minimize\ (\mu) \tag{28}$$

subject to,

$$\sum_{c \in C} x_c^i = 1, \forall i \in D \tag{29}$$

$$\sum_{i \in D} x_c^i <= \kappa_c, \forall c \in C \tag{30}$$

$$s^2_{KL} <= \kappa_{s^2} \tag{31}$$

Here, Constraint (29) ensures that each device is assigned to only one cluster. Constraint (30) limits the number of devices in each cluster to the prescribed capacity κ_c. This capacity is influenced by several factors, including the server's processing ability to handle concurrent data processing, and channel conditions and bandwidth constraints, which govern the simultaneous data transmission capacity from client devices to the server. Constraint (31) ensures that the variation of KL divergences across all clusters remains within a threshold κ_{s^2}, preventing any cluster from having a significantly higher KL divergence than others, which could negatively impact the inter-cluster learning process.

Theorem 1. *Binary Integer Non-Linear Programming (BINLP) involved problem presented in Eq. (28) is NP-hard*

Proof. We first commence our proof by proving that general BILP (Binary Integer Linear Programming) problems are NP-hard. It is straightforward that if general BILP problems are NP-hard then BINLP Problems are NP-hard too since solving NLPs is as hard as solving LPs.

3-SAT is a subset of the satisfiability problem in which a Boolean statement is split into clauses with the disjunction of precisely three literals in each clause, and the expression is in conjunctive normal form. The objective is to find a value assignment to each variable (TRUE or FALSE) that satisfies the Boolean statement. Given a 3-CNF SAT problem with N variables $\chi_1, \chi_2, \ldots, \chi_N$ and M clauses $C_1, C_2, \ldots, C_M$, we demonstrate how to construct an equivalent BILP instance, thereby showing that BILP is NP-hard.

For each SAT variable χ_u, we introduce a BILP variable υ_u, where $\upsilon_u = 1$ if χ_u is true, and $\upsilon_u = 0$ if χ_u is false.

For each clause C_v in the SAT instance, we introduce a corresponding constraint in the BILP. If $C_v = (\iota_{j1} \vee \iota_{j2} \vee \iota_{j3})$, where ι_{jk} is a literal, the corresponding BILP constraint ensures that at least one of the literals is true. For a literal ι_{jk} that is a positive occurrence of χ_u, the term in the constraint is υ_u; for a literal ι_{jk} that is a negative occurrence of χ_u, the term is $(1 - \upsilon_u)$. Each clause C_v yields an BILP constraint of the form:

$$\upsilon'_{j1} + \upsilon'_{j2} + \upsilon'_{j3} \geq 1 \tag{32}$$

where υ'_{jk} is υ_u if ι_{jk} is χ_u, or $(1 - \upsilon_u)$ if ι_{jk} is $\neg\chi_u$.

In this transformation, we have substituted every negated term $\neg\chi_u$ with $(1 - \upsilon_u)$, replaced each un-negated term χ_u with υ_u, and substituted the bitwise OR $(\vee)$ with addition $(+)$. This replacement strategy is effective because υ_u only takes values of 0 or 1. Consequently, $(1 - \upsilon_u)$ serves the same purpose as bitwise negation, and the summation of several terms yields 0 only if all of them are 0, or a positive value if any of them is positive.

The objective function can be arbitrary since we are interested in feasibility (satisfiability) rather than optimization. A simple choice is,

$$minimize \sum_{u=1}^{N} \upsilon_u. \tag{33}$$

This construction shows that any solution to the BILP corresponds to a satisfying assignment of the original SAT problem, and vice versa. Hence, solving a BILP is at least as hard as solving a 3-CNF SAT problem, establishing the NP-hardness of BILP or 0–1 ILP by reduction from 3-CNF SAT, which is known to be NP-complete. Therefore, the NP-hardness of BINLP or 0–1 INLP is also established.

Given the NP-hard nature of the optimal clustering problem, it poses a significant challenge, especially when dealing with a large number of devices, leading to high memory usage and time-intensive solutions. Conventional iterative techniques like alternative optimization and genetic algorithms are exceedingly complex. To address this, we propose an efficient approximate algorithm using Deep Reinforcement Learning (DRL).

5. PPO based DRL algorithm

Traditional iterative algorithms for solving combinatorial optimization problems often depend on heuristics designed by domain experts. These hand-crafted heuristics may not always be optimal due to the complexity of the problems. DRL presents a promising alternative by automating the development of these heuristics. Through training an agent in a supervised or self-supervised manner, DRL can improve solution strategies without the need for manual intervention [27]. This is why we investigate such an approach to approximate a solution to the NP-hard clustering problem. First, we illustrate the essential elements of the MDP (Markov Decision Process) in DRL, then develop a PPO-based (Proximal Policy Optimization) [28] approach to address the clustering problem.

The MDP elements in DRL involve the state s_t, action a_t and reward r_t, presented as follows:

State: In each time slot t, the DRL agent (which is the server) observes the state of the environment, which is presented by a tuple as $s_t = \{\delta^i, c \text{ if } i \in D_c\}, \forall i \in D$. That is, for all the client devices $i \in D$, the agent is observing the data distribution information δ^i and the cluster c to which the device i currently is assigned.

Action: Upon observing the state s_t, the agent takes action $a_t = \{a_t^i\}, \forall i \in D$, where each action a_t^i for a specific device i is drawn from the action space $\mathcal{A}^i$. In this context, $\mathcal{A}^i$ is defined as $\mathcal{A}^i = \{a^{i,1}, a^{i,2}, a^{i,3}\}$. Specifically, action $a^{i,1}$ indicates that if device i is presently assigned to cluster c, it maintains its current assignment. Meanwhile, action $a^{i,2}$ signifies that if device i is currently assigned to cluster c, it will be reassigned to the next cluster, denoted as $c+1$. Lastly, action $a^{i,3}$ implies that if device i is currently assigned to cluster c, it will be reassigned to the previous cluster, represented as $c-1$. This process involves making new clustering decisions for the participating devices. It is important to note that the next cluster after the last cluster is the first cluster, and conversely, the previous cluster of the first cluster is the last one.

Reward: The agent executes the action based on the observed state and obtains an immediate reward r_t from the environment. To reflect the optimization objective in the long run, we design the form of the reward function considering the objective (28) and associated constraints, which is given by,

$$r_t = \frac{1}{\mu} - \zeta_1 \sum_{c \in C}(max(0, (|D_c| - \kappa_c))) - \zeta_2(max(0, (s_{KL}^2 - \kappa_{s^2}))). \tag{34}$$

In this context, the term $\frac{1}{\mu}$ signifies the inverse of the Kullback–Leibler loss, and $\zeta_1 \sum_{c \in C}(max(0, (|D_c| - \kappa_c)))$ represents a linear penalty function associated with the degree of violation of the cluster size constraint (30), where ζ_1 is the coefficient of the penalty term. Similarly, $\zeta_2(max(0, (s_{KL}^2 - \kappa_{s^2})))$ functions as another linear penalty function linked to the extent of violation of the constraint (31), and ζ_2 denotes the coefficient of this penalty term. In conventional DRL, the objective is to maximize episodic rewards. Here, our goal is to minimize the cumulative KL divergence across all clusters, so the reward is defined as the inverse of the summation of KL divergence, μ.

As previously mentioned, clustering devices in the DCSL framework presents a complex INLP problem that traditional optimization methods struggle to solve. Recent advances in machine learning, particularly deep learning, provide innovative solutions to these challenges. Proximal Policy Optimization (PPO) is a well-regarded reinforcement learning algorithm known for its stability and reliability, making it an effective tool for addressing this problem within the DCSL framework.

The PPO operates as an on-policy algorithm, learning from actions within the current policy rather than external data. It improves over the Trust Region Policy Optimization (TRPO) [29] algorithm by introducing a clipping function that constrains policy changes, preventing excessive deviation and ensuring stability. In contrast, algorithms like Q-learning without a trust region constraint are prone to instability.

PPO refines policies iteratively through trial and error. Beginning with an initial policy, the algorithm interacts with the environment, collecting data to update the policy for maximizing expected rewards. PPO operates within the actor–critic framework, where the actor-network serves as the policy generator for actions a_t, and the critic network assesses the state value $V(s_t)$ to fine-tune the current policy. The actor-network θ utilized in this context consists of a 3-layer MLP (Multilayer Perceptron), while the critic network ω also features a 3-layer MLP.

To preserve the old policy during each update iteration, PPO employs the initialization of a network named actor-old, possessing an identical structure and parameters as the actor-network. The actor-old network remains uninvolved in the training process and merely replicates parameters from the actor-network before each update round, thus maintaining the previous strategy for the ongoing round. Subsequently, the actor-network proceeds to explore and refine the new strategy.

The probability ratio between the new and old policies is expressed as $Y_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, where θ and θ_{old} denote the policy parameters associated with the actor-network and the old actor-network, respectively. In PPO, the actor-network θ is designated with the following objective:

$$L_{CLIP}^{actor}(\theta) = \hat{\mathbb{E}}_t[min(Y_t(\theta)\hat{A}_t, clip(Y_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{35}$$

where epsilon is a hyper-parameter, for instance, $\epsilon = 0.2$ and $\hat{A}_t$ is the truncated version of generalized advantage estimation. The rationale behind this objective is explained as follows: The term $clip(Y_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ alters the surrogate objective $Y_t(\theta)\hat{A}_t$ by restricting the probability ratio within the range $[1-\epsilon, 1+\epsilon]$. This step discourages shifting Y_t beyond the interval boundaries. Ultimately, the minimum value between the clipped and unclipped surrogate objectives is selected, establishing the final objective as a conservative lower bound on the unclipped objective.

The actor's objective is fine-tuned through the advantage function, employing a truncated version of the Generalized Advantage Estimator (GAE). This technique effectively mitigates gradient estimation variance, consequently reducing the number of samples required for training.

$$\hat{A}_t = \sum_{z=0}^{T-t-1} (\gamma\lambda)^z (r_{t+z} + \gamma V(s_{t+1+z}) - V(s_{t+z})), \quad \forall t \in [0, T]. \tag{36}$$

Here, γ represents the discount factor, λ denotes the GAE parameter facilitating a balance between bias and variance in estimation, t indicates the time index within the trajectory segment $[0, T]$, and the term $V(s_t) = \sum_{z=0}^{T-t} \gamma^z r_{t+z}$ signifies the cumulative discounted reward, serving as a representation of the state-value function. Expressing $V^\omega(s_t)$ as the state-value function estimated by the critic ω, the critic's loss is formulated as follows:

$$L^{critic}(\omega) = [V^\omega(s_t) - V(s_t)]^2 \tag{37}$$

Hence, the actor and critic-network updates can be performed under Eq. (35) and (37), respectively. The steps of the PPO-based DRL Clustering Algorithm are summarized in Algorithm 2.

Algorithm 2 PPO-based DRL Clustering Algorithm in DCSL

1: Randomly initialize clusters $c \in C$ with devices $i \in D$
2: Initialize number of episodes EPS, number of timesteps T, actor-network θ, critic-network ω, replay buffer β
3: **for** episode $= 1 \dots EPS$ **do**
4: **for** $t = 1 \dots T$ **do**
5: Obtain state s_t form the environment
6: Execute action a_t via actor network θ
7: Compute log-probability ψ_t
8: Compute reward r_t
9: Save transitions $\tau_t =< s_t, a_t, r_t, \psi_t >$ into β
10: **end for**
11: **for** $t = T, T-1 \dots 1$ **do**
12: Calculate $\hat{A}_t$ according to Eq. (36)
13: **end for**
14: Update actor θ w.r.t. optimization in Eq. (35)
15: Update critic ω w.r.t. loss function in Eq. (37)
16: **end for**

In line 1, we are initializing the clusters $c \in C$ randomly such that the cluster assignment does not violate constraint (29) to (31). In line 2 we are setting up the maximum number of episodes EPS, the number of timesteps in each episode T, the actor-network θ, critic network ω and replay buffer β. From lines 5–8, the server running the PPO algorithm observes the state s_t from the environment, takes an action a_t via the policy generated by the actor-network θ, and computes log probability ψ_t and corresponding reward r_t [30]. The state, action, reward and transition probability tuple $\tau_t =< s_t, a_t, r_t, \psi_t >$ is then saved as an experience in the reply buffer β in line 9. Lines 5–9 are

Table 2
Simulation parameters.

Parameter	Value
Server-side processing capacity (approx.)	100 GHz
Device-side processing capacity	2.2 GHz
Number of devices	80
Max size of cluster	5–12
Learning rate of sever-side model	0.01
Learning rate of device-side model	0.05
Batch size for MNIST and Brain Tumor dataset	16
Size of the server-side model on MNIST dataset	1.31 MB
Size of the device-side model on MNIST dataset	0.35 MB
Avg. Smashed data size on MNIST dataset	21 KB
Avg. smashed gradient size on MNIST dataset	45 KB
Size of the server-side model on Brain Tumor dataset	0.76 MB
Size of the device-side model on Brain Tumor dataset	0.19 MB
Avg. Smashed data size on Brain Tumor dataset	11 KB
Avg. smashed gradient size on Brain Tumor dataset	24 KB

repeated in each timestep t. In line 12 the GAE $\hat{A}_t$ is calculated. In lines 14–15, we update the actor-network θ and critic network ω by optimizing corresponding objective functions defined in Eq. (35) and (37) respectively. The lines from 4 to 15 iterate across all episodes until the maximum episode number EPS is reached.

6. Performance evaluation

In the preceding chapters, we detailed our proposed DCSL scheme. In this chapter, we subject DCSL to evaluation alongside contemporary state-of-the-art SL algorithms, specifically vanilla SL [6], CPSL [8] and SplitFed [7]. Our results demonstrate a noteworthy performance advantage for DCSL across various metrics, surpassing the state-of-the-art algorithms by a significant margin. The primary hypothesis driving our evaluation is that while DCSL does not directly optimize for faster convergence time, its design ensures that the loss is calculated on IID-like sampling within each cluster. This property makes the mini-batch loss an unbiased estimator of the population loss, leading to more convergent gradient descent steps. As a result, DCSL achieves faster convergence with higher stability and greater accuracy compared to other schemes. In contrast, state-of-the-art techniques often fail to maintain this unbiased estimation, resulting in biased losses, non-convergent steps, and fluctuations in accuracy.

6.1. Simulation environment

In our simulation, the edge server operates at 100 billion cycles per second (100×10^9 cycles/s) and supports 50 devices. We utilize two datasets: the Brain Tumor dataset [31] and the MNIST dataset [32].

The MNIST dataset, used for image classification, comprises 50,000 training samples and 10,000 test samples across ten classes of handwritten digits (“0” to “9”). The data distribution is non-IID, with each device handling three randomly selected digit classes. The MNIST model employs a 12-layer LeNet architecture, including six convolutional layers, three max-pooling layers, and three fully-connected layers, with the third layer designated as the cut layer. Conversely, the Brain Tumor dataset consists of 7022 MRI images classified into glioma, meningioma, no tumor, and pituitary categories. For this dataset, each device processes two randomly selected categories using a custom 10-layer CNN, with the third layer as the cut layer. These configurations reflect typical non-IID data distributions found in real-world applications. The main simulation parameters are detailed in Table 2.

The dataset selection was motivated by both standardization and domain relevance. The MNIST dataset was chosen as a standard benchmark due to its extensive use in evaluating state-of-the-art algorithms, providing a basis for comparison. Additionally, it has relevance in healthcare applications such as Optical Character Recognition (OCR) for handwritten prescriptions. The Brain Tumor dataset represents medical datasets in healthcare, allowing us to assess DCSL’s effectiveness in a realistic domain scenario.

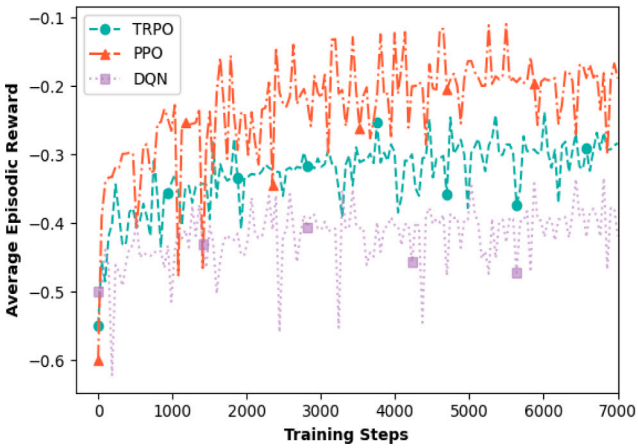


Fig. 4. Comparative rewards achieved by different DRL algorithms.

To evaluate DCSL’s performance under non-IID conditions, we simulated non-IID distributions across devices. For MNIST, each device was assigned data from three randomly selected digit classes, creating significant data skew. Similarly, in the Brain Tumor dataset, each device processed data from two randomly selected categories, introducing non-uniformity. These setups mimic real-world scenarios where data distributions are often non-IID, validating DCSL’s capability to handle such challenges.

The evaluation framework examines DCSL’s performance across several metrics, including accuracy, convergence time, and stability. Detailed results and comparisons with baseline algorithms are presented in the following sections, demonstrating DCSL’s superiority in addressing non-IID data distributions while maintaining efficiency.

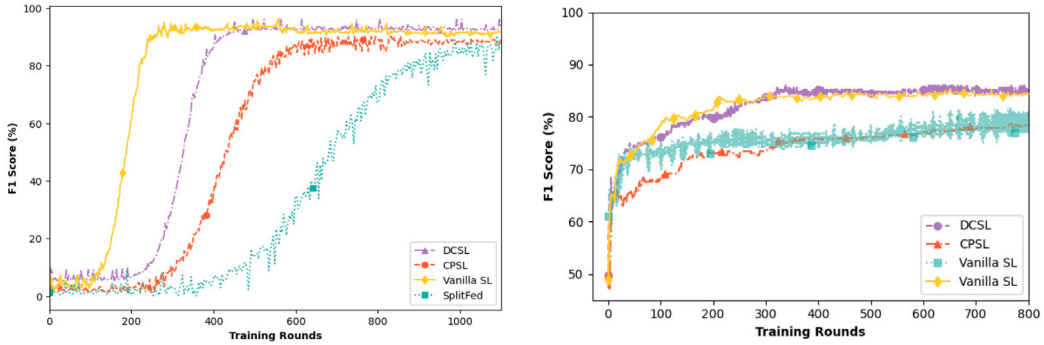
6.2. Simulation results

In this section, we evaluate the performance of DCSL against existing SL algorithms, including vanilla-SL [6], CPSL [8], and Split-Fed [7], focusing on training accuracy across different training rounds, training times, cluster sizes, and KL divergence levels.

6.2.1. Performance of different DRL algorithms

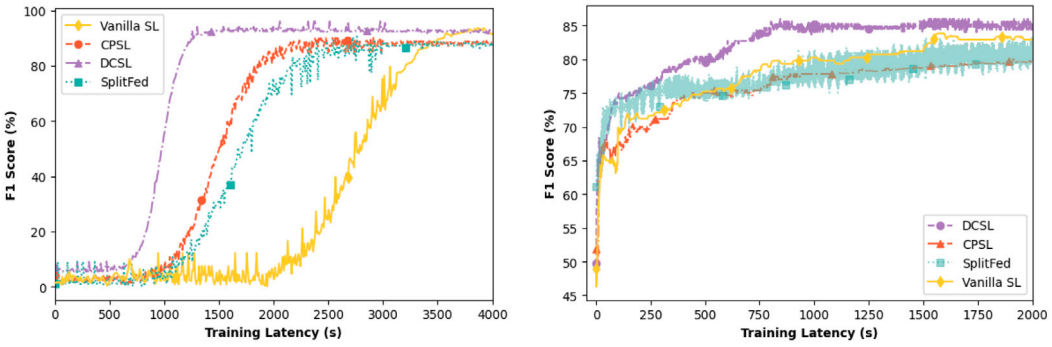
In Fig. 4, we compare reward convergence in DRL training across our PPO-based method, TRPO, and DQN. TRPO, which maintains policy stability by ensuring updates remain within a trust region, provides strong theoretical guarantees for policy improvement but often suffers from slower convergence due to its computational complexity and the constraints imposed by the trust region. In contrast, DQN, combining deep learning with Q-learning to estimate Q-functions, frequently overestimates Q values, especially in large action spaces, complicating the discovery of optimal strategies.

The PPO method outperforms both TRPO and DQN by achieving faster convergence and demonstrating greater stability. PPO improves on TRPO by using a clipped objective function that simplifies the optimization process while preventing large policy updates, leading to more efficient learning. This results in quicker adaptation to the environment and more robust policy performance. Furthermore, PPO’s balance between exploration and exploitation and its ability to handle large state and action spaces more effectively enhance reward acquisition and policy identification, making it superior in various complex scenarios.



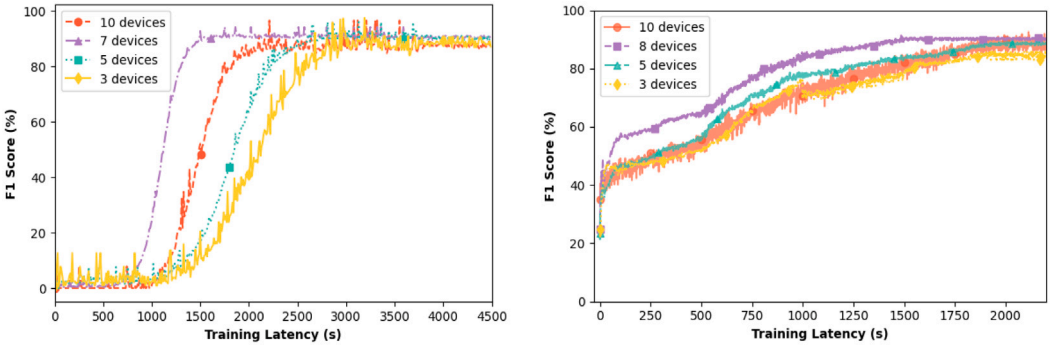
(a) Accuracy on MNIST dataset (b) Accuracy on MRI dataset

Fig. 5. Training performance comparison among different schemes over MNIST and MRI datasets.



(a) Accuracy on MNIST dataset (b) Accuracy on MRI dataset

Fig. 6. Training accuracy vs. latency comparison among different schemes over MNIST and MRI datasets.



(a) Accuracy on MNIST dataset (b) Accuracy on MRI dataset

Fig. 7. Training accuracy vs. latency comparison for different cluster sizes on MNIST and MRI datasets.

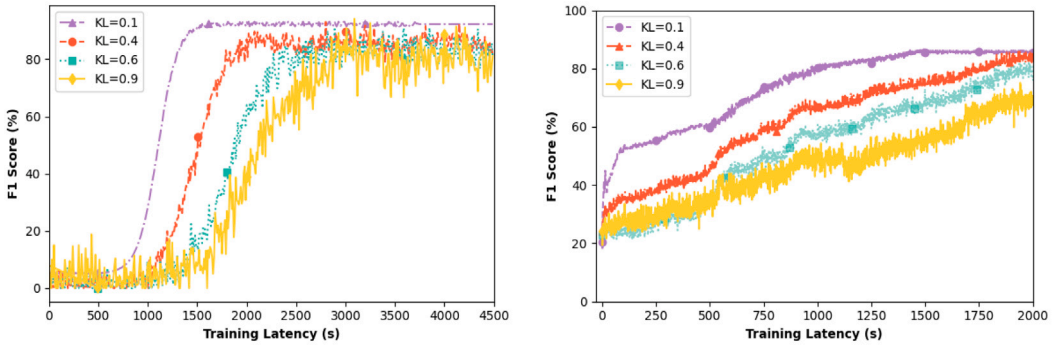
6.2.2. Model performance w.r.t training rounds

Fig. 5 compares the training performance of DCSL against benchmark algorithms in terms of accuracy. For both datasets, DCSL achieves an F1-score comparable to Vanilla SL but requires more training rounds due to intra-cluster model aggregation via FedAvg, which introduces additional synchronization overhead and slows convergence. In contrast, CPSL converges more slowly as it does not account for data heterogeneity within clusters, leading to imbalanced updates and inefficient learning. SplitFed exhibits the slowest convergence, as the extensive device-side model aggregation imposes significant computational and communication overhead.

6.2.3. Model performance w.r.t training time

Considering that the per-round training latency varies across different schemes, we conducted a comprehensive assessment of the overall training latency as depicted in Fig. 6. The overall training latency is calculated as the product of the per-round training latency and the number of training rounds. Notably, DCSL exhibits a shorter training latency in reaching convergence compared to Vanilla SL. This can be attributed to the fact that the per-round training latency in DCSL is considerably smaller than that of SL.

CPSL converges more slowly with a per-round training latency similar to DCSL because it does not account for device data distribution when forming clusters. However, it still surpasses Vanilla SL in training latency. Conversely, SplitFed has the lowest per-round latency but suffers from an unstable F1 Score, affected by its extensive device-side model aggregation. Vanilla SL shows the slowest convergence, hindered by its sequential, non-parallel training approach, where devices are trained one at a time.



(a) Accuracy on MNIST dataset

(b) Accuracy on MRI dataset

Fig. 8. Impacts of KL divergence values on training performances.

6.2.4. Effects of cluster size on accuracy

Fig. 7 provides a comparative analysis of performance based on the number of devices per cluster. A key finding is that the number of devices significantly affects training latency and convergence. Notably, the DCSL scheme achieves the shortest training latency with seven devices per cluster for the MNIST dataset and eight devices per cluster for the MRI dataset, indicating these as the optimal cluster sizes for each respective dataset. However, reducing the cluster size to three slows down training by decreasing parallelism, while increasing it to ten enhances parallelism but leads to instability and fluctuating accuracy due to the complex model aggregation process via the Fed-Avg algorithm involving more devices.

6.2.5. Impacts of KL divergence on accuracy

Fig. 8 illustrates the impact of different average KL divergence levels on performance across clusters. For the MNIST dataset, a high KL divergence of 0.9 leads to a convergence time of 4300 s, while a lower divergence of 0.1 results in faster convergence at 1500 s. Intermediate KL divergences of 0.4 and 0.6 correspond to convergence times of 2900 s and 3300 s, respectively. Similar trends are observed for the MRI dataset.

Higher KL divergence levels in the DCSL framework increase training latency and model instability due to a greater deviation from the IID property across clusters. This effect occurs because batches drawn from a cluster's aggregated dataset may not represent the overall data distribution well. Lower KL divergence suggests a closer alignment with the overall data distribution i.e. the population distribution, allowing batch gradients to more accurately estimate the complete gradient. This leads to faster and more stable model convergence. Conversely, higher divergence indicates misalignment, resulting in biased gradient estimates and slower convergence.

7. Discussion

This study introduced DCSL, a data distribution-aware parallel split learning framework that effectively addresses the challenges of non-IID data in decentralized learning. Unlike prior works that focus primarily on latency reduction, DCSL prioritizes data distribution alignment, ensuring that each cluster's aggregated dataset better represents the global population. This key insight enables faster convergence, improved stability, and higher accuracy, bridging a critical gap in existing split learning methods.

A fundamental finding is that model performance in non-IID settings heavily depends on how loss is computed at the server. If loss is evaluated on a sample that mirrors the global data distribution, it serves as an unbiased estimator of population loss, leading to more effective gradient updates. DCSL leverages this property by intelligently clustering devices to ensure that training data within each cluster is representative, significantly improving generalization. While parallel SL frameworks like CPSL and SplitFed have improved training efficiency, they largely overlook the impact of data heterogeneity, resulting in biased models. DCSL addresses this by optimizing intra- and inter-cluster learning, reducing bias while maintaining training efficiency.

The practical relevance of DCSL lies in its potential applications in IoT and healthcare systems, where data privacy and diversity are critical. DCSL provides a scalable and privacy-preserving solution for collaborative healthcare AI, particularly for rare disease detection, where individual hospitals lack sufficient data, and data-sharing is restricted by regulations. In disease detection, dataset distribution varies significantly across healthcare institutions due to demographic, temporal, and regional differences. For instance, in Idiopathic Pulmonary Fibrosis (IPF) detection, a large hospital may have a diverse dataset with CT scans from various demographics, while smaller clinics primarily contribute localized patient records, resulting in non-IID data. In disease classification, handling non-IID data is particularly crucial, as accuracy across all classes is essential due to the high sensitivity of medical diagnoses—misclassification of a rare condition can lead to severe clinical consequences. While traditional FL and SL frameworks struggles with such disparities, leading to biased models and poor generalization, DCSL addresses this by leveraging data distribution aware clustering. A central server coordinates training while hospitals train device-side models locally, exchanging only smashed data to preserve privacy. Through data distribution aware intra and inter-cluster learning, DCSL refines global models, harmonizing diverse datasets to enhance diagnostic accuracy and generalizability, even when individual institutions have skewed data distributions.

One key limitation of the proposed DCSL framework lies in its assumption of stable and uninterrupted connectivity between client devices and the central server throughout the training process—a condition that may not hold in dynamic, real-world IoT and edge environments. Device dropouts or disconnections can disrupt the intended data distribution within clusters, undermining DCSL's core objective of maintaining cluster-level representativeness relative to the global population. Additionally, while DCSL effectively mitigates non-IID effects through strategic clustering, it currently overlooks device-specific constraints such as computational capacity, communication bandwidth, and energy availability. In low-powered IoT devices (e.g., battery-constrained sensors or wearable health monitors), this omission becomes critical, as straggler devices can bottleneck the entire cluster's training progress by delaying the aggregation phase [33,34]. Moreover, DCSL assumes a uniform cut-layer placement across all devices, which can impose disproportionate computational burdens on resource-limited nodes, particularly when deeper cut layers increase client-side model complexity [8,20]. Beyond system-level challenges, DCSL's reliance on exchanging smashed data and gradients, though privacy-preserving relative to raw data sharing, still leaves it susceptible to potential inference or reconstruction attacks [35], warranting the integration of stronger privacy preserving techniques such as differential privacy or secure multiparty computation in future work.

8. Conclusion

In this paper, DCSL, a parallel split learning framework designed to address the challenges of non-IID data distribution in decentralized learning was introduced. By intelligently clustering client devices based on their dataset characteristics, DCSL enhances model convergence, reduces training latency, and improves overall efficiency. Extensive experiments demonstrated that DCSL outperforms traditional split learning methodologies in training latency, accuracy, and model convergence, reinforcing its potential for privacy-preserving distributed machine learning applications. The explicit consideration of data distribution in DCSL addresses a critical gap in decentralized learning, making machine learning systems more robust, adaptable, and effective in real-world scenarios.

Future work will focus on enhancing DCSL to address current limitations. Incorporating asynchronous training and adaptive clustering can improve robustness against unstable device conditions, ensuring better adaptability to real-world network constraints. Integrating homomorphic encryption and random perturbation techniques will enhance security by protecting gradients and labels from inference attacks. Additionally, addressing device mobility and irregular participation through dynamic cluster reassignment will further improve DCSL's practicality in dynamic environments. Evaluating its performance in diverse domains such as where non-IID data is prevalent such as financial fraud detection, and churn prediction will further validate its scalability and impact. Furthermore, future research can explore integrating advanced privacy-preserving techniques, optimization algorithms, and transfer learning approaches within DCSL to enhance its capabilities. Additionally, incorporating model compression and communication efficiency techniques can make DCSL more scalable for edge and IoT devices.

CRediT authorship contribution statement

Md. Tanvir Arafat: Software, Methodology, Conceptualization. **Md. Abdur Razzaque:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Abdulhameed Alelaiwi:** Writing – review & editing. **Md. Zia Uddin:** Writing – review & editing. **Mohammad Mehedi Hassan:** Writing – original draft, Software, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors are grateful to King Saud University, Riyadh, Saudi Arabia for funding this work through Ongoing Research Funding Program (ORF-2025-18).

Data availability

Data will be made available on request.

References

[1] D. Gündüz, P. de Kerret, N.D. Sidiropoulos, D. Gesbert, C.R. Murthy, M. van der Schaar, Machine learning in the air, IEEE J. Sel. Areas Commun. 37 (10) (2019) 2184–2199.

[2] L. Chen, J. Xu, Seek common while shelving differences: Orchestrating deep neural networks for edge service provisioning, IEEE J. Sel. Areas Commun. 39 (1) (2020) 251–264.

[3] G.D.P. Regulation, General data protection regulation (GDPR), Intersoft Consult. Accessed Oct. 24 (1) (2018).

[4] X.S. Shen, C. Huang, D. Liu, L. Xue, W. Zhuang, R. Sun, B. Ying, Data management for future wireless networks: Architecture, privacy preservation, and regulation, IEEE Netw. 35 (1) (2021) 8–15.

[5] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, Y. Gao, A survey on federated learning, Knowl.-Based Syst. 216 (2021) 106775.

[6] O. Gupta, R. Raskar, Distributed learning of deep neural network over multiple agents, J. Netw. Comput. Appl. 116 (2018) 1–8.

[7] C. Thapa, P.C.M. Arachchige, S. Camtepe, L. Sun, Splitfed: When federated learning meets split learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, 36, (8) 2022, pp. 8485–8493.

[8] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, W. Shi, Split learning over wireless networks: Parallel design and resource management, IEEE J. Sel. Areas Commun. 41 (4) (2023) 1051–1066.

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.

[10] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, Federated learning with non-iid data, 2018, arXiv preprint arXiv:1806.00582.

[11] C. Briggs, Z. Fan, P. Andras, Federated learning with hierarchical clustering of local updates to improve training on non-IID data, in: 2020 International Joint Conference on Neural Networks, IJCNN, IEEE, 2020, pp. 1–9.

[12] Z. Chen, D. Li, R. Ni, J. Zhu, S. Zhang, FedSeq: A hybrid federated learning framework based on sequential in-cluster training, IEEE Syst. J. 17 (3) (2023) 4038–4049.

[13] W. Luping, W. Wei, L. Bo, CMFL: Mitigating communication overhead for federated learning, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2019, pp. 954–964.

[14] J. Kang, Z. Xiong, D. Niyato, H. Yu, Y.-C. Liang, D.I. Kim, Incentive design for efficient federated learning in mobile networks: A contract theory approach, in: 2019 IEEE VTS Asia Pacific Wireless Communications Symposium, APWCS, IEEE, 2019, pp. 1–5.

[15] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, L. Liang, Self-balancing federated learning with global imbalanced data in mobile systems, IEEE Trans. Parallel Distrib. Syst. 32 (1) (2020) 59–71.

[16] M. Yang, X. Wang, H. Qian, Y. Zhu, H. Zhu, M. Guizani, V. Chang, An improved federated learning algorithm for privacy preserving in cybertwin-driven 6G system, IEEE Trans. Ind. Informatics 18 (10) (2022) 6733–6742.

[17] T. Li, A.K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, Proc. Mach. Learn. Syst. 2 (2020) 429–450.

[18] S.J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, H.B. McMahan, Adaptive federated optimization, in: International Conference on Learning Representations, 2021, URL https://openreview.net/forum?id=LkFG3lB13U5.

[19] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, B. Varghese, Fedadapt: Adaptive offloading for iot devices in federated learning, IEEE Internet Things J. 9 (21) (2022) 20889–20901.

[20] E. Samikwa, A. Di Maio, T. Braun, Ares: Adaptive resource-aware split learning for internet of things, Comput. Netw. 218 (2022) 109380.

[21] Z. Lin, G. Zhu, Y. Deng, X. Chen, Y. Gao, K. Huang, Y. Fang, Efficient parallel split learning over resource-constrained wireless edge networks, IEEE Trans. Mob. Comput. (2024).

[22] J. Tirana, S. Lalis, D. Chatzopoulos, MP-sl: Multihop parallel split learning, 2024, arXiv preprint arXiv:2402.00208.

[23] Y. Cai, T. Wei, Efficient split learning with non-iid data, in: 2022 23rd IEEE International Conference on Mobile Data Management, MDM, IEEE, 2022, pp. 128–136.

[24] M. Arafeh, M. Wazzeh, H. Sami, H. Ould-Slimane, C. Talhi, A. Mourad, H. Otrok, Efficient privacy-preserving ML for IoT: Cluster-based split federated learning scheme for non-IID data, J. Netw. Comput. Appl. (2025) 104105.

[25] M. Wu, G. Cheng, D. Ye, J. Kang, R. Yu, Y. Wu, M. Pan, Federated split learning with data and label privacy preservation in vehicular networks, IEEE Trans. Veh. Technol. (2023).

[26] S. Ji, Z. Zhang, S. Ying, L. Wang, X. Zhao, Y. Gao, Kullback–Leibler divergence metric learning, IEEE Trans. Cybern. 52 (4) (2020) 2047–2058.

[27] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, A.A. Cire, Combining reinforcement learning and constraint programming for combinatorial optimization, in: Proceedings of the AAAI Conference on Artificial Intelligence, 35, (5) 2021, pp. 3677–3687.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, arXiv preprint arXiv:1707.06347.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine Learning, PMLR, 2015, pp. 1889–1897.
[30] W. Liu, B. Li, W. Xie, Y. Dai, Z. Fei, Energy efficient computation offloading in aerial edge networks with multi-agent cooperation, IEEE Trans. Wirel. Commun. (2023).
[31] A. Saleh, R. Sukaik, S.S. Abu-Naser, Brain tumor classification using deep learning, in: 2020 International Conference on Assistive and Rehabilitation Technologies (ICareTech), IEEE, 2020, pp. 131–136.
[32] L. Deng, The mnist database of handwritten digit images for machine learning research [best of the web], IEEE Signal Process. Mag. 29 (6) (2012) 141–142.
[33] S. Wang, M. Ji, A unified analysis of federated learning with arbitrary client participation, Adv. Neural Inf. Process. Syst. 35 (2022) 19124–19137.
[34] S. Hossan, F. Mahmud, P. Roy, M.A. Razzaque, M.M. Rahman, Energy and latency-aware computation load distribution of hybrid split and federated learning on IoT devices, in: Proceedings of the 10th International Conference on Networking, Systems and Security, 2023, pp. 61–68.
[35] F. Fu, X. Wang, J. Jiang, H. Xue, B. Cui, ProjPert: Projection-based perturbation for label protection in split learning based vertical federated learning, IEEE Trans. Knowl. Data Eng. (2024).

Md. Tanvir Arafat received the B.Sc. degree in Computer Science and Engineering from the University of Dhaka, Bangladesh, in 2022, and the M.Sc. degree from the same institution in 2024. He is currently a Lecturer at BRAC University and a Research Assistant with the Green Networking Research Group, University of Dhaka. His research interests include privacy-preserving and distributed machine learning, fog robotics, machine learning at the edge, IoT, and cyber–physical systems. He is member of IEEE.

Md. Abdur Razzaque (M'08, SM'12) is a Professor and current chairman of the Dept. of Computer Science and Engineering, University of Dhaka. He received his B.S. in Applied Physics and Electronics and M.S. in Computer Science from University of Dhaka, Bangladesh, in 1995 and 1996, respectively. He obtained a Ph.D. in Computer Engineering from Kyung Hee University, South Korea in 2009. He worked at Green University of Bangladesh for different periods as Pro-Vice-Chancellor, Dean of Faculty of Science and Engineering and Chairperson, Dept of CSE during 2016–2021. He was a research professor at Kyung Hee University, South Korea, during 2010–2011. He worked as a visiting professor at Stratford University, Virginia, USA, in 2017. He is the director of Green Networking Research Group (https://du.ac.bd/body/faculty_details/cse/1768) of the Dept. of CSE, DU. He has been promoting Outcome Based Education for Science and Engineering Faculties of the country's leading universities. He led many national and international-funded research projects. His research interest is in modeling, analysis and optimization of wireless networking protocols and architectures, Mobile Crowdsourcing, Sensor Data Clouds, Internet of Things, Edge Computing, etc. He has published 180+ research papers in peer-reviewed conferences and journals. He is offering editorial services to reputed journals and contributing to numerous international conferences at different capacities including Associate Editor of IEEE Access and Founding General Chair of STI conference 2019-2025. He is a senior member of IEEE, a member of IEEE Computer Society, Internet Society (ISOC), etc.

Abdulhameed Alelaiwi is a Fill Professor of Software Engineering Department, at the College of Computer and Information Sciences, King Saud University. Riyadh, Saudi Arabia. He received his Ph.D. degree in Software Engineering from the College of Engineering, Florida Institute of Technology-Melbourne, USA in 2002. He has authored and co-authored many publications including refereed IEEE/ACM/Springer journals, conference papers, books, and book chapters. His research interest includes software testing analysis and design, cloud computing, and multimedia. He is a member of IEEE.

Dr. Md Zia Uddin obtained his Ph.D. degree in Biomedical Engineering in 2011 from Kyung Hee University of South Korea. He is currently working as a Senior Research Scientist in Human Computer Interaction group, SINTEF Digital, Oslo, Norway. His research fields are mainly focused on data & feature analysis from various sources (sensors and others), physical/mental healthcare, human machine/computer/robot interaction, pattern recognition, deep learning, artificial intelligence etc. He has a good teaching experience with more than 20 computer science-related courses from bachelor's degree to PhD. He has got more than 150 research publications including books, international journals, and conferences. His google scholar citations are more than 5000. His research works received best/outstanding paper awards in several peer reviewed international conferences. Dr. Zia has been working/leading in many work packages of national and international research projects. He has been enlisted in the world's top 2% scientists prepared by the Stanford University of USA and Elsevier BV.

Mohammad Mehedi Hassan received the Ph.D. degree in computer engineering from Kyung Hee University, South Korea, in February 2011. He is currently a Professor with the Information Systems Department, College of Computer and Information Sciences (CCIS), King Saud University (KSU), Riyadh, Saudi Arabia. He has authored and co-authored more than 365+ publications including refereed journals (333 SCI/ISI-Indexed Journal papers, 42 conference papers, 1 book, and 2 book chapters. His research interests include cloud computing, edge computing, the Internet of Things, body sensor networks, big data, deep learning, mobile cloud, smart computing, wireless sensor networks, 5G networks, and social networks. He has served as the Chair and a Technical Program Committee Member in numerous reputed international conferences/workshops, such as IEEE CCNC, ACM BodyNets, and IEEE HPCC. He was a recipient of a number of awards, including 2021 Outstanding Editors Award from Future Generation Computer Systems journal, Distinguished Research Award from College of Computer and Information Sciences, KSU 2020, Best Conference Paper Award from IEEE Int'l Conf on Sustainable Technologies for Industry 4.0 (STI) 2020, Best Journal Paper Award from IEEE Systems Journal in 2018, Best Conference Paper Award from CloudComp in 2014 conference and the Excellence in Research Award from College of Computer and Information Sciences, KSU (2015 and 2016). He is one of the top 2% Scientists of the world in Networking and Telecommunication field. He is one of the top computer scientists in Saudi Arabia as well. Recently, his nine publications have been recognized as the ESI Highly Cited Papers.