HSFL: An Efficient Split Federated Learning Framework via Hierarchical Organization

Tengxi Xia†, Yongheng Deng†, Sheng Yue‡, Junyi He§, Ju Ren†*, Yaoxue Zhang†
†Department of Computer Science and Technology, Tsinghua University, Beijing, China
‡School of Computer Science and Engineering, Central South University, Changsha, China
§School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, China
*Corresponding author
Email: {xtx19,dyh19}@mails.tsinghua.edu.cn, {renju, zhangyx}@tsinghua.edu.cn,
sheng.yue@csu.edu.cn, w

***Abstract*—Federated learning (FL) has emerged as a popular paradigm for distributed machine learning among vast clients. Unfortunately, resource-constrained clients often fail to participate in FL because they cannot pay for the memory resources required for model training due to their limited memory or bandwidth. Split federated learning (SFL) is a novel FL framework in which clients commit intermediate results of model training to a cloud server for client-server collaborative training of models, making resource-constrained clients also eligible for FL. However, existing SFL frameworks mostly require frequent communication with the cloud server to exchange intermediate results and model parameters, which results in significant communication overhead and elongated training time. In particular, this can be exacerbated by the imbalanced data distributions of clients. To tackle this issue, we propose HSFL, a hierarchical split federated learning framework that efficiently trains SFL model through hierarchical organization participants. Under the HSFL framework, we formulate a Cloud Aggregation Time Minimization (CATM) problem to minimize the global training time and design a light-weight client assignment algorithm based on dynamic programming to solve it. Moreover, we develop a self-adaption approach to cope with the dynamic computational resources of clients. Finally, we implement and evaluate HSFL on various real-world training tasks, elaborating on its effectiveness and superiority in terms of efficiency and accuracy compared to baselines.**

***Index Terms*—Federated Learning, Split Learning, Edge Computing.**

I. Introduction

Federated Learning (FL) [1] is a promising distributed machine learning paradigm which enables distributed clients to collaboratively train a shared model without exposing their privacy-sensitive raw data. In the typical FL framework, clients perform model training locally using their raw data, and only the model updates are uploaded to the central cloud server for aggregation. Then the aggregated model updates are sent back to the clients for the next round of training. Despite attractive privacy-preserving benefits, a critical challenge of FL is the significant resource requirements for model training and communication [2]. Model training, especially for complex deep neural networks, results in considerable memory and computation overhead [3]. Besides, for collaborative learning, clients have to iteratively exchange model updates with the central cloud server, which brings significant communication overhead [4], [5]. Such massive resource requirements can be prohibitive for resource-constrained mobile clients, hindering FL's widespread adoption.

To mitigate the resource constraints of clients, Split Federated Learning (SFL) [6] proposed to split the whole model into multiple portions and assign a part of the model to the cloud server for assisted training. Specifically, clients perform forward propagation on their client-side models in parallel and commit the intermediate results to the cloud server to complete the training of the server-side model. Although SFL can relieve clients' memory and computation burden, clients have to frequently commit the intermediate results of model training to the cloud server, which brings additional transmission time and consumes significant communication resources.

To solve the above problems, we propose to exploit the great potential of edge servers, as communicating with the edge server can significantly reduce the communication overhead and transmission delay of clients compared to the central cloud server. To this end, we propose a novel framework HSFL, i.e., Hierarchical Split Federated Learning, where edge servers play the role of training assistants and model aggregators of SFL. In HSFL, clients train the client-side model using their local data and commit the intermediate results to the edge server to train the server-side model. After several client-edge collaborative training, clients commit the client-side model parameters to the edge server, which aggregates the model parameters and sends the aggregated parameters back to clients. Then, after sever edge aggregations, the edge servers send the edge model parameters to the cloud server, which aggregates the edge parameters and returns the aggregated parameters to edge servers for the next round of training. For the HSFL framework, the critical challenge is determining which edge server should cooperate with each client, reducing the model training time and enhancing the model accuracy. To achieve this goal, we aim to minimize the required time per cloud aggregation by solving the formulated Cloud Aggregation Time Minimization (CATM) problem. On the other hand, the data distribution on the client side presents the not identically and independently distribution (Non-IID). Therefore we try to make the data distribution of each edge server more balanced,

978-3-903176-51-5 © 2022 IFIP

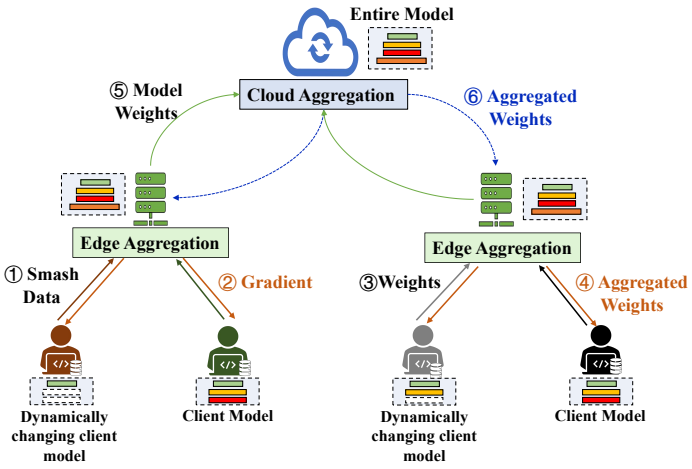

Fig. 1: The framework of our proposed HSFL.

which has been demonstrated to be effective in improving the learning performance [4], [7]. To strike a delicate trade-off between model accuracy and training time, we devise a light-weight assignment mechanism to determine for each client which edge server it should collaborate with. To deal with the dynamic changes of client computing resources, we design an adaptive algorithm that can dynamically adjust the association strategies between clients and edge servers. To evaluate the performance of HSFL, we conduct extensive experiments on various real-world training tasks and compare them with state-of-the-art FL mechanisms. Experimental results demonstrate that HSFL can significantly improve the learning accuracy and reduce the model training time. The main contributions of this paper are summarized as follows:

1) We propose a Hierarchical Split Federated Learning framework (HSFL) that enables collaborative model training between clients and edge servers, which overcomes client resource limitations with reduced time consumption.
2) Under the HSFL framework, we study the Cloud Aggregation Time Minimization (CATM) problem and design a light-weight adaptive assignment algorithm to solve it while considering the dynamic changes in client computing resources.
3) We conduct extensive experiments with real-world datasets. Experimental results demonstrate the efficacy of HSFL in terms of model accuracy and training time compared with state-of-the-art baselines (i.e., FL, SFL and HFL).

II. System Description and Problem Design

In this section, we first describe the Hierarchical Split Federated Learning Framework (HSFL), afterward formally design the Cloud Aggregation Time Minimization (CATM) problem, and finally analyze the rationalization of the problem. The symbols used in this paper are summarized in Table I.

A. Hierarchical Split Federated Learning Framework

This section mainly describes our proposed HSFL framework. We consider an HSFL framework comprising a set of N resource-constrained clients (denoted by $\mathcal{N}$), a set of M edge servers (denoted by $\mathcal{M}$) and one cloud server. In Fig. 1, edge servers and the central cloud server have the entire model structure, and clients own part of the model structure. The dotted line represents the dynamically changing computing resources of clients. To model the dynamic change in clients' computation resources, we let l_i^t denote the number of layers that client i can perform training in the t-th round of cloud aggregation. Specifically, we select some clients from the client set $\mathcal{N}$ and assign them to M edge servers. In each client-edge collaborative training, clients train the client-side model using their local data and transmit the intermediate results to the edge server to train the server-side model. After E_l times client-edge collaborative training, clients send the client-side model parameters to the edge server for parameters aggregate, and the edge server updates the server-side model and returns the aggregated parameters to clients. This process is called edge aggregation. After E_e times edge aggregation, edge servers send the model parameters to the cloud server, which aggregates the edge parameters and returns the aggregated parameters to edge servers for the next round of training. Finally, we describe cloud aggregation time as follows:

Cloud aggregation time:

1) The time of each round cloud aggregation T^A is determined by the last edge server that completes E_e times edge aggregation. The T^A can be expressed as $T^A = \max\{T_j^E\}$.
2) We use T_j^E to denotes the time of the edge server j complete E_e times edge aggregation, which is determined by the last clients that completes E_l times client-edge collaborative training. The T_j^E can be expressed as $T_j^E = \max_{i \in \mathcal{S}_j}\{C_i\} + t_j^C$, where t_j^C is a constant that is the transmission time when edge server j sends model parameters to the cloud server.
3) The time for a client i to complete E_l times client-edge collaborative training $C_i = t_i^a + t_i^S + t_{i,j}^u$, where t_i^a is the time for the client complete client-side model training, t_i^S is the time of edge-server cooperating with the client to finishes the server-side model training; and $t_{i,j}^u$ is the trans time between the client and edge server.

B. Problem Definition

The edge data distribution refers to the distribution of data connected to edge server clients [7]. In the HSFL framework, we aim to solve the cloud aggregation time minimization problem. Furthermore, we refer to the difference in edge data distribution as the edge distribution distance and design an edge distribution distance minimization problem to make the data distribution of each edge server more balanced and thus improve the learning performance. Thus, we describe the above problems as follows.

Cloud Aggregation Time Minimization (CATM):

TABLE I: PRINCIPAL NOTATIONS

Symbol	Definition
N	The number of clients.
M	The number of edge servers.
E_l	The number of local iterations performed by clients in each edge aggregation.
E_e	The number of edge aggregations performed by edge servers in each cloud aggregation.
$\mathcal{B}(t)$	The client assignment scheme in t-th cloud aggregation
H	The number of cloud aggregation times.
L_i	The minimum resources that client i can join in model training.
R_i	The maximum resources that client i can join in model training.
D_i	The data size of client i.
S_j	The set of clients associated with edge server j.
V_j^E	Association limitation at edge server j.
T_t^A	The cloud aggregation time in the t-th round
T_j^E	The time of edge server j completes edge aggregation.
C_i^t	The time for the client i to finish model training in t-th round of cloud aggregation.

We give a set of clients $\mathcal{N}$ and a set of edge servers $\mathcal{M}$. Then, how to design a reasonable client assignment scheme to assign N clients to M servers that minimize the required time per cloud aggregation. Define $b_{i,j}^t \in \{0,1\}$ as a binary variable that indicates in the t-th round cloud aggregation, whether client i is to be associated with edge servers j ($b_{i,j}^t = 1$) or not ($b_{i,j}^t = 0$). The time of per cloud aggregation can be expressed as follows:

$$T_t^A = \max_{j \in \mathcal{M}} \{T_j^E(\mathcal{B}(t))\} \tag{1}$$

Where $T_j^E(\mathcal{B}(t))$ denotes the time of edge server j completes E_e times edge aggregation under the t-th round assignment scheme, and $\mathcal{B}(t)$ is the assignment scheme of the t-th round, $\mathcal{B}(t) = \{b_{i,j}^t\}$. The time to complete E_e times edge aggregation can be expressed as follows:

$$T_j^E(\mathcal{B}(t)) = \max_{i \in \mathcal{S}_j} \{C_i^t \cdot b_{i,j}^t\} + t_j^C \quad \forall j \in \mathcal{M} \tag{2}$$

In addition to cloud aggregation time, it is equally vital to make edge data distribution more balanced. Therefore we define the edge distribution distance to model the balance between edge distributions.

Edge Distribution Distance Minimization (EDDM):

Besides the cloud aggregation time, we also consider making the data distribution of each edge server more balanced, effectively improving learning performance [4], [7]. To quantify the balance between edge data distributions more intuitively, we define the edge distribution distance, which is the KL divergence between the edge server data distribution and the uniform distribution. During each cloud aggregation, the edge distribution distance can be described as follows:

$$D_t^A = \sum_{j=1}^{m} D_{KL}(P_j^t || P_u) \tag{3}$$

In each cloud aggregation, $D_{KL}(\cdot||\cdot)$ is the KL divergence between the data distribution and uniform distribution of the edge server j in the t-th round of cloud aggregation. To strike appropriate balances among cloud aggregation time and edge distribution distance, we formalize the above two problems as Eq.(4). Edge servers usually have limited communication resources, so we constrain the number of connected clients to the edge serve j within V_j^E, shown in constraint (5). The constraint Eq.(6) means that each client can connect to at most one server in each cloud aggregation. Finally, in the constraint Eq.(7), we set $b_{i,j}^t$ as the decision variable, indicating whether client i is associated with edge server j in round t.

$$\min \ F = \sum_{t=1}^{H} (\lambda T_t^A + (1-\lambda) D_t^A) \tag{4}$$

$$s.t. \quad \sum_{i=1}^{N} b_{i,j}^t \leq V_j^E \qquad \forall j \in \mathcal{M} \tag{5}$$

$$\sum_{j=1}^{M} b_{i,j}^t \leq 1 \qquad \forall i \in \mathcal{N} \tag{6}$$

$$b_{i,j}^t \in \{0,1\} \qquad \forall i \in \mathcal{N}, \forall j \in \mathcal{M} \tag{7}$$

C. Problem Analysis

In Eq.(4), we first consider the CATM problem and in order to obtain higher accuracy, the EDDM problem is also taken into account. For each round cloud aggregation $t \in \{1, 2, ..., H\}$, the clients assignment scheme is $\mathcal{B}(t)$. However, they are difficult to process simultaneously for the following reasons. On the one hand, to minimize the time required for each cloud aggregation, we need to consider the physical distance between clients with edge servers and the resources that the client participates in training. On the other hand, physical distance is not equivalent to distribution distance. In other words, we also need to consider the distance between the edge distribution and the uniform distribution to obtain higher accuracy. To strike a better trade-off, we incorporate a parameter λ to tune the cloud aggregation time and edge data distribution.

III. Algorithm Design

To solve our proposed CATM problem, we first prove it is an NP-hard problem, afterward we design a light-weight client assignment algorithm to solve it.

A. Problem transformation

Combining Eq.(1) and Eq.(2) can be expressed as follows:

$$T_t^A = \max_{j \in \mathcal{J}} \{\max_{i \in \mathcal{S}_j} \{C_i^t \cdot b_{i,j}^t\} + t_j^C\} \tag{8}$$

In Eq.(8), the $\max$ operation is complicated for us, so we use the Minkowski distance [8] to transform it. The Minkowski distance as follows:

$$\max_{i=1}^{n} |x_i - y_i| = (\sum_{i=1}^{n} |x_i - y_i|^p))^{\frac{1}{p}} \tag{9}$$

When $p \to \infty$, Minkowski distance means the max element in an array.

$$\lim_{p\to\infty} (\sum_{i=1}^{n} |x_i - y_i|^p))^{\frac{1}{p}} = \max_{i=1,\ldots,n} |x_i - y_i| \tag{10}$$

Substituting Eq.(10) into Eq.(4), then we rewrite the objective function (4) as follows:

$$\min \ F = \sum_{t=1}^{H} (\lambda T_t^A + (1-\lambda) D_t^A) \tag{11}$$

$$s.t. \ \ D_t^A = \sum_{j=1}^{M} D_{KL}(P_j^t || P_u) \tag{12}$$

$$T_t^A = \lim_{p\to\infty} (\sum_{j=1}^{m} | \lim_{k\to\infty} (\sum_{i=1}^{N} b_{i,j}^t \cdot |C_i|^k)^{\frac{1}{k}} + t_j^C)|^p)^{\frac{1}{p}} \tag{13}$$

$$Constraints : (5), (6), (7).$$

Proof. We simplify D_t^A (12), assuming that the data distribution p_i of client i is the uniform distribution. At this point, the data distribution of edge servers is always uniform distribution in any assignment scheme, where $P_j^t = P_u$ and $D_t^A = 0$ the Eq.(11) is converted to Eq.(14).

$$\begin{aligned} \min \ F &= \sum_{t=1}^{H} \lambda T_t^A \\ &\sum_{i=1}^{N} b_{i,j}^t \le V_j^E, && \forall j \in \mathcal{M} \\ &\sum_{j=1}^{M} b_{i,j}^t \le 1, && \forall i \in \mathcal{N} \\ &b_{i,j}^t \in \{0,1\}, && \forall i \in \mathcal{N}, \forall j \in \mathcal{M} \end{aligned} \tag{14}$$

The Eq.(14) can be simplified to the 0/1 multiple knapsack problem [9]. The client is viewed as the good of 1 weight and C_i value. The edge server is viewed as the knapsack, where the capacity of the knapsack is limited to V_j^E. Now we need to put n goods into m knapsacks and minimize the objective function (14). At this point, the special case of our problem transforms into a 0/1 multiple knapsack problem. The 0/1 multiple knapsacks problem is a classic NP-hard problem, and we reduce the problem (14) to the 0/1 multiple knapsacks problem. Eq.(14) is a particular case of Eq.(11), so the original problem (11) can be reduced to a 0/1 multi-knapsack problem. That is, our proposed CATM problem is an NP-hard problem. □

B. Two-Stage Client Assignment (TSCA)

Given M edge servers and N clients, we associate clients with edge servers in each cloud aggregation, thus minimizing the objective function (11). It's worth noting that the number of clients connecting to an edge server cannot exceed its connection limit V_j^E. For this object, we present the Two-Stage Client Assignment algorithm, which is described in Algorithm 1. We use a two-stage approach to design the client assignment scheme and minimize the objective function (11).

Dynamic Stage: We use the s_e to present the set of clients connected to the edge server e, and $|s_e|$ to denote the number of clients connected to the edge server e. We first put the client i to the edge server e, until $|s_e| = V_e^E$. Then, we minimize our object function (11) by exchanging client i and client i_s, where client is i_s means the client has connected to the edge server. **Exchange Stage:** Select one client i in edge server e, and selected client k in the other edge server e', then calculates the value of $\Delta F_{i,k}$. Based on the calculated value $\Delta F_{i,k}$, we find out the association pair of the client i and client k that maximizes the value of $\Delta F_{i,k}$. Then exchange the client i and client k.

$$\begin{aligned} D_{KL}(P_e||P_u) &= -\sum_k P_e(k) \ln \frac{P_u(k)}{P_e(k)} \\ &= -\sum_k P_e(k)(\ln P_u(k) - lnP_e(k)) \\ &= \sum_k P_e(k) \ln P_e(k) - \sum_k P_e(k) \ln P_u(k) \end{aligned} \tag{15}$$

To reduce the TSCA algorithm's time complexity, we design an optimized function CheckEIE($\cdot$,$\cdot$,$\cdot$) based on maximizing information entropy as described in Algorithm 1. We use e denotes an edge server, c_i and c_r to denote whether the client is connected to the edge server e, respectively. The ΔH indicates the variable of change in information entropy of the edge server e caused by exchanging client c_i and client c_k. If $\Delta H > 1\mathrm{e}^{-3}$ exchange client c_i and client c_k, otherwise not.

C. Self Adaption Assignment (SAA)

In this section, we present the self-adaption assignment algorithm to face the clients' resources change, which details in Algorithm 2. At the beginning of each round of cloud aggregation, clients whose computing resources have changed commit model training information l_*^t to the cloud server. Afterward, we calculate the deviation ΔF of the objective function when the assignment scheme $\mathcal{B}(t-1)$ of the $t-1$ round is applied to the next round. If $\Delta F > \mathbf{Z}$, we use $\mathcal{B}(t-1)$ as the t round scheme, otherwise get a new scheme $\mathcal{B}(t)$ through Algorithm 1, where $\mathbf{Z}$ is our acceptable change threshold, we set $\mathbf{Z} = 1\mathrm{e}^{-7}$.

D. Workflow of HSFL

In fig. 2 describes the workflow of HSFL. The workflow of HSFL consists of client assignment, client-side training, edge-side training, client-side backward, edge aggregation, and cloud aggregation. Each part of the workflow describes

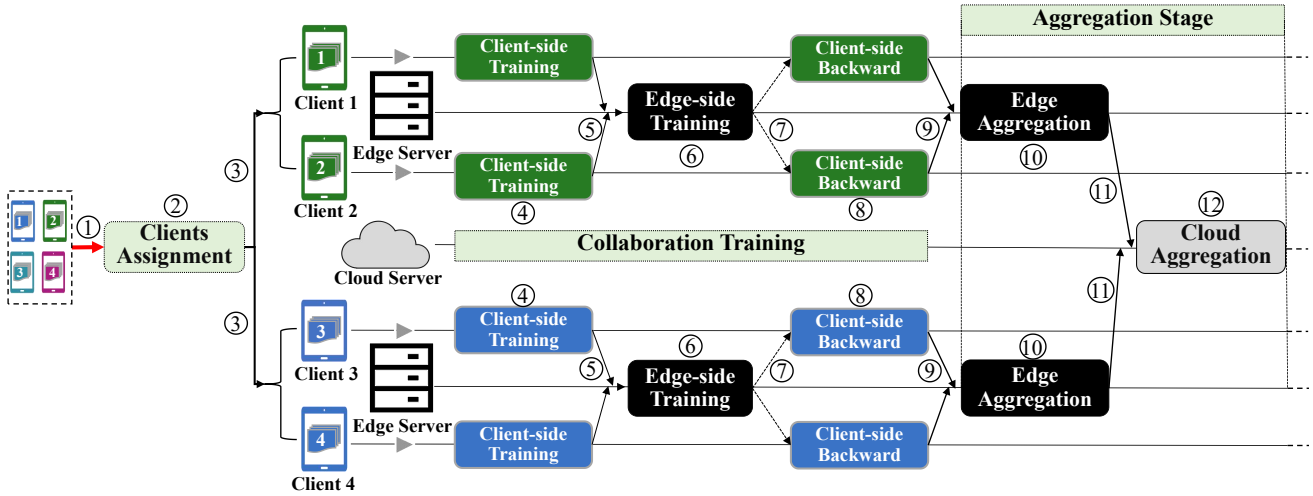

Fig. 2: The workflow of HSFL.

as follows, and details describe in Algorithm 3.

1) Initially, the client sends information to the central cloud server (①). Then, the cloud server utilizes this submitted information to execute Algorithm 1, assigning clients to edge servers (②). Next, the cloud server sends the associate result to clients (③).
2) During the collaboration training stage, the client executes client-side model training in parallel (④). Then, the client sends the intermediate result and labels to the specified edge server (⑤). After that, the edge server assist clients with edge-side training in parallel (⑥) and sends the gradient to the client (⑦), then client backward follow this gradient (⑧). Finally, go back to (④) and repeat the collaboration training stage E_l times.
3) Enter the Aggregation Stage, the client sends the partial model parameters to the edge server (⑨), edge server uses the FedAvg [1] algorithm to aggregate the model parameters (⑩). After E_e times edge aggregations, edge servers commit the model parameters to the cloud server (⑪). The cloud server uses the FedAvg algorithm to aggregate the model parameters (⑫) and then sends the aggregated parameters to edge servers. Afterward, edge servers update the edge model and back it to clients.

IV. Experiments

In this section, we conduct extensive experiments to demonstrate the efficacy of HSFL, which are compared with state-of-the-art baselines on real-world tasks.

A. Experimental Setting

Environment Setup. We build the HSFL framework emulation system by PyTorch 1.10.0, use the sleep function in python to simulate the transmission delay between devices, and adopt widely used deep learning tasks and the corresponding deep learning model to evaluate our framework.

Datasets and Models. We adopt three widely used datasets and the corresponding models: 1) CIFAR-10 [10] dataset and the ResNet18 model in [11], 2) Fashion-MNIST [12] dataset and the ResNet34 model in [11]; and 3) Mnist [13] dataset and CNN model retrieved from PyTorch[1]. The CIFAR-10 dataset contains a train set of 50,000 training images and a test set of 10,000 test images. Each image is a 32x32 color image associated with a label from 10 classes, the Fashion-MNIST dataset consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 gray scale image associated with a label from 10 classes; and the mnist dataset contains a training set of 60000 training examples, and a test set of 10000 test examples. For Non-IID data distribution at clients, we assign each client 2 different sample labels, each label with 300 samples.

Other parameters and baselines: In specific, we set $\mu^E = 1\mathrm{e}^{-5}$s/km, $\mu^C = 1\mathrm{e}^{-6}$s/km, where μ^E is the transmission latency between the client and edge servers, μ^C is the transmission latency between the client and cloud server. For parameters related to model training, we set the size of local mini batch as $\beta = 32$ and the learning rate $\tau = 0.005$. We choose three different geo-locations network topologies from the Internet Topology Zoo [14], i.e., Aarnet (Australia,19 nodes), Belenet2010 (Belgium, 19 nodes) and Chinanet (China, 38 nodes). These networks have the latitude and longitude information of clients, which is used to calculate the distance among clients via the Haversine formula [15]. We obtain the shortest distance between clients using the Dijkstra algorithm and addition one location in Hong Kong, China, is included to represent the cloud server.

- **Classical FL (FL):** Distributed clients directly communicate with the cloud server to commit model parameters [1].
- **Cloud-based SFL (SFL):** This work combines the strengths of FL and SL, which allows a cloud server (master) to assist resource-constrained clients in completing model training, which then submits model parameters to another server (fed server) for aggregation [6].

[1]Retrieved from https://github.com/pytorch/examples/blob/master/mnist.

Algorithm 1: Two-Stage Client Assignment (TSCA)

Input: (1) Data size D_i; (2) Number of client-side model training layers l_i^t; (3) client-side training time $t_{i,t}^a$.
Output: Client assignment scheme $\mathcal{B}(t)$
Initialize $s_e \leftarrow \emptyset, P_e \leftarrow 0$ **for each e** $\in \mathcal{M}$;
Function `CheckEIE` (c_a,c_r,e)**:**
  The information entropy $H(P)$.
  $\Delta H \leftarrow H(P_e - c_r + c_a) - H(P_e)$
  if $\Delta H > 1e^{-3}$ **then**
    return True
  else
    return False
  end
Dynamic Stage:
for $e \in \mathcal{M}$ **do**
  for $i \in \mathcal{N}$ **do**
    if $|s_e| < V_e^E$ **then**
      $s_e \leftarrow s_e + i$
    else if $|s_e| = V_e^E$ **then**
      for $i_s \in s_e$ **do**
        if *CheckEIE*(i,i_s,e) **then**
          $\Delta F \leftarrow F(s_e - i_s + i) - F(s_e)$
          if $\Delta F < 0$ **then**
            $s_e \leftarrow s_e + i - i_s$
      end
  end
end
Exchange Stage:
for $(e, e') \in \mathcal{M}$ **do**
  for $i \in s_e$ **do**
    for $k \in s_{e'}$ **do**
      if *CheckEIE*(i,k,e) *or CheckEIE*(i,k,e') **then**
        $F_b \leftarrow F(s_e) + F(s_{e'})$
        $F_a \leftarrow F(s_e - i + k) + F(s_{e'} - k + i)$
        Compute $\Delta F_{i,k} \leftarrow F_a - F_b$
    end
    Find the maximum $\Delta F_{i,k}$
    $s_e \leftarrow s_e - i + k$
    $s'_e \leftarrow s'_e + i - k$
  end
end

Algorithm 2: Self Adaption Assignment(SAA)

Input: (1) Current cloud aggregation round t; (2) Number of client-side model training layers l_i^t;
Output: Clients assignment scheme $\mathcal{B}(t)$
if *t=0* **then**
  $\mathcal{B}(t) \leftarrow$ Algorithm 1.
else
  $\Delta F \leftarrow |F_t(\mathcal{B}(t-1)) - F_{t-1}(\mathcal{B}(t-1))|$
  if $\Delta F >$ **Z then**
    $\mathcal{B}(t) \leftarrow \mathcal{B}$
  else
    $\mathcal{B}(t) \leftarrow$ Algorithm 1.
  end
end
return $\mathcal{B}(t)$

- **Astraea (HFL):** Based on the hierarchical federated learning framework, which tries to make the data distribution of each edge mediator more balanced [7].

B. Performance Comparison with baselines

We first conduct performance experiments against the HSFL framework and baselines, mainly investigating their accuracy at the same number of iterations. Given the number of resource-constrained clients $N = [10, 20, 30, 40]$ and the number of edge servers $M = 3$. In particular, FL and SFL are without the edge aggregation stage. Therefore, to make the cloud aggregation frequency the same, we set the cloud frequency $CF = E_l \cdot E_e$. **CIFAR-10:** We use CIFAR-10 dataset and resnet18 model, we set $E_l = 2, E_e = 2$ and $CF = 4$. As shown in Fig. 3, the advantage of HSFL in accuracy is valid. For example, as shown in Fig. 3(d) which records the result of the client performing 80 iterations, the accuracy of baselines is 19.77% (SFL), 17.53% (FL), and 20.81% (HFL), but our method accuracy achieves 22.86% (HSFL). **Fashion-MNIST:** In Fig. 4, we use the Fashion-MNIST dataset and ResNet34 model to compare the performance, we set $E_l = 2, E_e = 2$ and $CF = 4$. For example, as shown in Fig. 4(d) which records the result of the client performing 80 iterations, the accuracy of baselines is 23.92% (SFL), 23.2% (FL), and 24.34% (HFL), but our method achieves 25.48% (HSFL). The above experiments demonstrate that HSFL can improve accuracy in the same iteration.

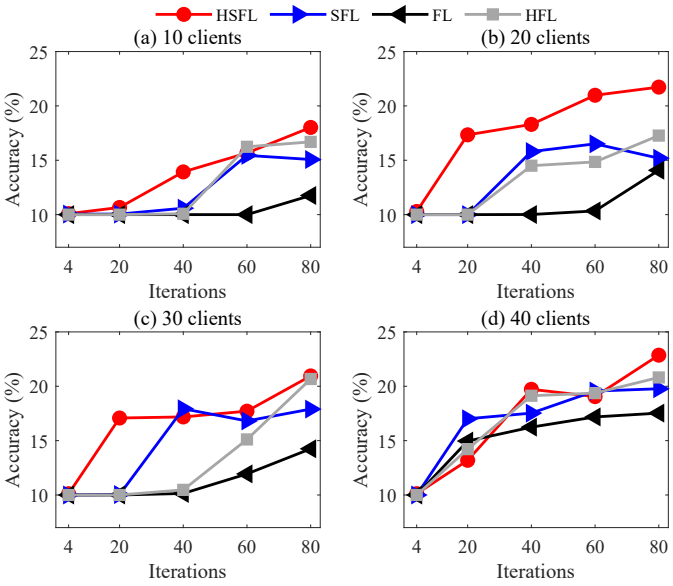

Fig. 3: Accuracy comparison of our method and baselines on the CIFAR-10 dataset for different client numbers. The number of the clients are set to 10, 20, 30, and 40.

Algorithm 3: Workflow of HSFL

Input: (1) H is the number of cloud aggregation;
for *cloud aggregation round* $t = 1, 2, 3, .., H$ **do**
  if *t=1* **then**
    Initialize $\mathcal{W}_j^E$ (Edge server-side model)
    Initialize $\mathcal{W}_i^C$ (Client-side model)
    Client scheme $\mathcal{B}(t) \leftarrow$ execute Algorithm 1;
  else
    Client scheme $\mathcal{B}(t) \leftarrow$ execute Algorithm 2;
  end
  $s_j \leftarrow \mathcal{B}(t)$ get clients set from $\mathcal{B}(t)$
  /*Edge Training*/
  for *each server* $j \in \mathcal{M}$ ***in parallel*** **do**
    for *each client* $i \in s_j$ ***in parallel*** **do**
      for *local iteration* $l = 1, 2, 3, ..., l_e$ **do**
        /*Client-side training*/
        Gets smash data and label from client i
        $(\Delta x_i, y_i) \leftarrow \mathcal{W}_i^C$
        Send (Δx_i and labels y_i) to server j
        /*Server-side training*/
        Calculate $\widetilde{y_i}$ with Δx_i on W_j^E
        loss $l_j \leftarrow$ loss_function$(y_i, \widetilde{y_i})$
        /*Server-side backward*/
        Get gradient $\Delta g_i \leftarrow$ backward on(l_i)
        Send Δg_i to client i
        /*Client-Side Backward*/
        client i backward on Δg_i
      end
    end
    /*Edge aggregation*/
    client send model parameters w_i to edge server j
    $W_j^E \leftarrow \frac{\sum_{i \in s_j} D_i w_i}{\sum_{i \in s_j} D_i}$
    Client-side model update $\leftarrow W_j^E$
  end
  /*Cloud aggregation*/
  for *edge server* $j \in \mathcal{M}$ ***in parallel*** **do**
    $D_j \leftarrow \sum_{i \in s_j} D_i$
    $W_j^E \leftarrow \frac{\sum_{i \in s_j} D_i w_i}{\sum_{i \in s_j} D_i}$
  end
  $W_{t+1}^C \leftarrow \frac{\sum_{j \in \mathcal{M}} D_j w_j^E}{\sum_{j \in \mathcal{M}} D_j}$
  Server-side model update $\leftarrow W_{t+1}^C$
end

C. Impact of Edge

With the framework performance guaranteed, we then stduy the impact of number of edge servers, including N,M,V_j^E. Specifically, we set $N = [20, 30]$, $M = [2, 3, 4]$ and $V_j^E = [6, 4, 3]$. As shown in Fig. (5), the performance of model accuracy and iterations on the Mnist dataset and the CNN model. We can achieve the following advantages. First, our approach can outperform the other two baselines in all settings. For instance, in Fig.5(a), when $M = 3$, the accuracy

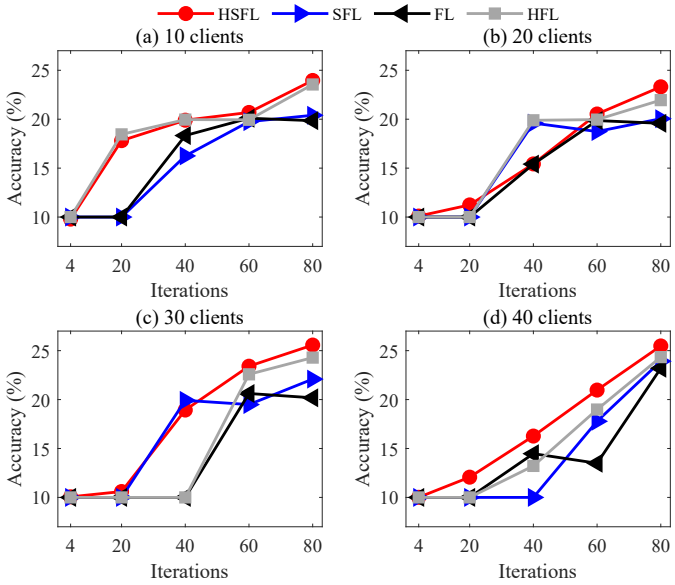

Fig. 4: Accuracy comparison of our method and baselines on the Fashion-Mnist dataset for different client numbers. The number of the clients are set to 10, 20, 30, and 40.

of our approach can achieve 92.746% with 80 times iterations , while the approach of SFL and FL can only achieve the performance of 75.15% and 61.33%. Second, our method converges faster than other two baselines in all settings. For example, in Fig.5(c), where $M = 3$, the loss of our approach can decrease to 0.48 with 20 times iterations, while the approach of SFL and FL can only achieve the loss of 1.44 and 1.67.

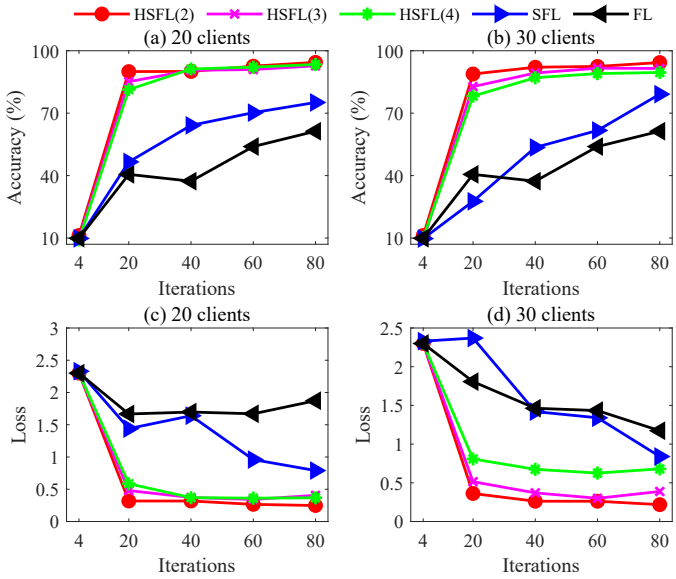

Fig. 5: The accuracy vs. Iterations under different settings of M

D. Efficiency Comparison with SplitFL

We compare the efficiency with SplitFL (SFL) [6] and use the CIFAR-10 dataset and the Fashion-Mnist dataset. We

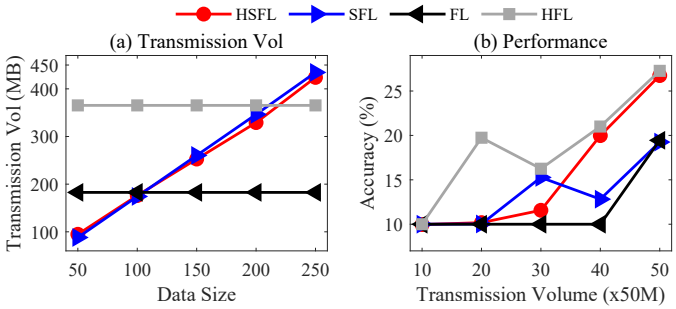

Fig. 6: The relationship between transmission volume and accuracy.

investigate the time required to complete $t = [5, 10, 15, 20]$ rounds of cloud aggregation. In order to simulate the dynamic change of client computing resources, in each round of cloud aggregation, we randomly select the number of layers l_i^t for model training on the client-side between L_i and R_i. As shown in Table II and Table III, where $N = 20$, $M = 3$ and $V_j^E = 3$, the edge server are located in network topologies Aarnet, Belenet and Chinanet. We can see that our method can outperform the SFL significantly in these network topologies. For instance, in Table III, finish 20 times cloud aggregation, the seconds of 3218.94 is required in SFL, while our proposed HSFL only takes 2914.06 seconds.

TABLE II: Fashion-Mnist Training Time (s).

	Cloud aggregation			
	5	10	15	20
HSFL	679.93	1312.46	1935.35	2523.05
SFL	790.27	1470.12	2145.44	2825.99

TABLE III: CIFAR-10 Training Time (s).

	Cloud aggregation			
	5	10	15	20
HSFL	809.27	1489.37	2196.40	2914.06
SFL	863.71	1653.78	2445.18	3218.94

E. Transmission Volume with Baselines

Whether the HSFL framework or other baselines, clients invariably require communication with cloud servers (or edge servers). Therefore, we study the client's transmission volume as crucial. For this reason, we use the Fashion-MNIST dataset to investigate the transmission volume of different data sizes. As shown in Fig. (6a), when the data size of each client is greater than 100, the client's transmission volume under the HSFL framework is higher than FL. In Fig. (6b), we compared the model's accuracy when the client uses the same transmission volume. The results show that our method significantly improves other baselines.

V. Related Work

Emerging distributed learning technology, such as Federated Learning (FL) [1], Hierarchical Federated Learning (HFL) [16], Split Learning (SL) [17], and Split Federated Learning (SFL) [6], has garnered a lot of attention both in industry and academia in recent years.

FL brings some unique challenges, including huge communication costs, imbalanced data, privacy and resource constraints [2], [18], [19], etc. Efficiency [20]–[22], incentives mechanism [23]–[25], privacy [26]–[28], client selection [29], [30] and some application frameworks such as visual detection Platform [31] etc.

In order to reduce the communication overhead and improve the convergence speed of the model, Hierarchical Federated Learning (HFL), first mentioned in [16], reduces the client's communication overhead by introducing edge servers. In [4], the overall communication overhead minimization problem is considered in the HFL scenario. Although these works [32]–[35] mention other issues with HFL, many potential issues with HFL remain to be exploited.

Except for data privacy, the model's privacy has also gradually attracted attention. Split Learning (SL) proposed in [17], [36], which considers both model privacy and data privacy divides a deep learning network into multiple parts that reside on different devices. In [37], the performance of FL and SL is verified in different experimental environments. In [38], expanded the application scope of SL and can use SL on 1D CNN models. Many works focus on the security and privacy of SL, [39] propose an atack method to expose security risks in SL, protection and attack always go together, [40] attention is paid to how to mix intermediate results and labels without losing efficiency. Inspired by SL, to implement efficient FL models on resource-constrained devices, SFL [6] combines the advantages of FL and SL, using a powerful server to assist multiple clients for model training and aggregation of model parameters. However, the clients participating in SFL training frequently communicate with the cloud server, generating a considerable transmission delay and significantly increasing the training time, which is unacceptable for IoT devices. In order to address this issue, we propose the HSFL framework to improve the convergence speed of the model and reduce the time required for clients to participate in SFL training.

VI. Conclusion

In this work, we present the design, implementation, and evaluation of HSFL, an efficient split federated learning framework via hierarchical organization. Then, we formulate the Cloud Aggregation Time Minimization problem and design an efficient client assignment mechanism to associate clients and edge servers. Extensive experimental results demonstrate the effectiveness and efficiency of our framework. We believe that our proposed framework will provide a valuable solution for split federated learning. In the future, we will propose a decentralized approach to designing client assignment schemes and reduce the communication overhead of clients.

References

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of

deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282, 2017.

[2] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020.

[3] Ang Li, Jingwei Sun, Pengcheng Li, Yu Pu, Hai Li, and Yiran Chen. Hermes: an efficient federated learning framework for heterogeneous mobile clients. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 420–437, 2021.

[4] Yongheng Deng, Feng Lyu, Ju Ren, Yongmin Zhang, Yuezhi Zhou, Yaoxue Zhang, and Yuanyuan Yang. Share: Shaping data distribution at edge for communication-efficient hierarchical federated learning. In *IEEE ICDCS*, pages 24–34, 2021.

[5] Zhiyuan Wang, Hongli Xu, Jianchun Liu, He Huang, Chunming Qiao, and Yangming Zhao. Resource-efficient federated learning with hierarchical aggregation in edge computing. In *IEEE INFOCOM*, pages 1–10, 2021.

[6] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. *arXiv preprint arXiv:2004.12088*, 2020.

[7] Moming Duan, Duo Liu, Xianzhang Chen, Yujuan Tan, Jinting Ren, Lei Qiao, and Liang Liang. Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications. In *IEEE ICCD*, pages 246–254, 2019.

[8] Mahinda Mailagaha Kumbure and Pasi Luukka. A generalized fuzzy k-nearest neighbor regression model based on minkowski distance.

[9] Xiangjing Lai, Jin-Kao Hao, Fred Glover, and Zhipeng Lü. A two-phase tabu-evolutionary algorithm for the 0–1 multidimensional knapsack problem. *Information sciences*, 436, 2018.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE CVPR*, pages 770–778, 2016.

[12] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[13] Y.LeCun. The mnist database of handwritten digits. In *http://yann. lecun. com/exdb/mnist/*, 1998.

[14] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE JSAC*, vol.29, no. 9, pp. 1765-1775, 2011.

[15] C Carl Robusto. The cosine-haversine formula. *The American Mathematical Monthly*, vol.64, no. 1, pp. 38-40, 1957.

[16] Lumin Liu, Jun Zhang, SH Song, and Khaled B Letaief. Client-edge-cloud hierarchical federated learning. In *IEEE ICC*, pages 1–6, 2020.

[17] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[18] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[19] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 2021.

[20] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In *IEEE INFOCOM*, pages 63–71, 2018.

[21] Nguyen H Tran, Wei Bao, Albert Zomaya, Minh NH Nguyen, and Choong Seon Hong. Federated learning over wireless networks: Optimization model design and analysis. In *IEEE INFOCOM*, pages 1387–1395, 2019.

[22] Bing Luo, Xiang Li, Shiqiang Wang, Jianwei Huang, and Leandros Tassiulas. Cost-effective federated learning design. In *IEEE INFOCOM*, pages 1–10, 2021.

[23] Yongheng Deng, Feng Lyu, Ju Ren, Yi-Chao Chen, Peng Yang, Yuezhi Zhou, and Yaoxue Zhang. Fair: Quality-aware federated learning with precise user incentive and model aggregation. In *IEEE INFOCOM*, pages 1–10, 2021.

[24] Yufeng Zhan, Peng Li, Zhihao Qu, Deze Zeng, and Song Guo. A learning-based incentive mechanism for federated learning. *IEEE Internet of Things Journal*, 7(7):6360–6368, 2020.

[25] Ming Tang and Vincent WS Wong. An incentive mechanism for cross-silo federated learning: A public goods perspective. In *IEEE INFOCOM*, pages 1–10, 2021.

[26] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[27] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM*, pages 2512–2520, 2019.

[28] Liang Gao, Li Li, Yingwen Chen, Wenli Zheng, ChengZhong Xu, and Ming Xu. Fifl: A fair incentive mechanism for federated learning. In *ICPP*, pages 1–10, 2021.

[29] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *ACM HPDC*, pages 125–136, 2020.

[30] Baturalp Buyukates and Sennur Ulukus. Timely communication in federated learning. In *IEEE INFOCOM WKSHPS*, pages 1–6, 2021.

[31] Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. Fedvision: An online visual object detection platform powered by federated learning. In *AAAI*, volume 34, pages 13172–13179, 2020.

[32] Seyyedali Hosseinalipour, Sheikh Shams Azam, Christopher G Brinton, Nicolo Michelusi, Vaneet Aggarwal, David J Love, and Huaiyu Dai. Multi-stage hybrid federated learning over large-scale d2d-enabled fog networks. *IEEE/ACM Transactions on Networking*, 2022.

[33] Jinze Wu, Qi Liu, Zhenya Huang, Yuting Ning, Hao Wang, Enhong Chen, Jinfeng Yi, and Bowen Zhou. Hierarchical personalized federated learning for user modeling. In *ACM WWW*, pages 957–968, 2021.

[34] Siqi Luo, Xu Chen, Qiong Wu, Zhi Zhou, and Shuai Yu. Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning. *IEEE Transactions on Wireless Communications*, 19(10):6535–6548, 2020.

[35] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *IEEE IJCNN*, pages 1–9, 2020.

[36] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[37] Yansong Gao, Minki Kim, Sharif Abuadbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit A Camtepe, Hyoungshick Kim, and Surya Nepal. End-to-end evaluation of federated learning and split learning for internet of things. *arXiv preprint arXiv:2003.13376*, 2020.

[38] Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. Can we use split learning on 1d cnn models for privacy preserving training? In *ACM ASIA-CCS*, pages 305–318, 2020.

[39] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning. In *ACM SIGSAC*, pages 2113–2129, 2021.

[40] Danyang Xiao, Chengang Yang, and Weigang Wu. Mixing activations and labels in distributed training for split learning. *IEEE Transactions on Parallel and Distributed Systems*, vol. 33(no. 11):pp. 3165–3177, 2021.