Efficient Model Training in Edge Networks With Hierarchical Split Learning

Songge Zhang, *Student Member, IEEE*, Wen Wu, *Senior Member, IEEE*, Lingyang Song, *Fellow, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—In this paper, we propose an efficient model training scheme, named Group-based Hierarchical Split Learning (GHSL), which can accelerate the artificial intelligence (AI) training process in edge networks in a "first-sequential-then-parallel" manner. Specifically, the proposed scheme hierarchically splits an AI model into a user-side and server-side model, while dividing a number of users into multiple groups. Users in each group train user-side models with the interaction of the shared server-side model sequentially; different groups perform the above training process parallelly; the AI models of each group are aggregated into a global model. We also carry out the convergence analysis for the proposed scheme over non-independent and identically distributed data, which reveals that the convergence rate depends on user grouping. Furthermore, we propose a data-driven two-stage user grouping algorithm to minimize the overall training delay, taking user resource heterogeneity and the black-box training process into account. The proposed algorithm first utilizes the Gaussian process regression approach to determine the number of groups, and then employs the coalition game theory to determine the optimal user grouping decision. Comprehensive simulation results demonstrate that the proposed scheme can reduce training delay, user-side computational workload, and communication overhead by up to 19%, 53%, and 54%, respectively, comparing to state-of-the-art benchmarks.**

***Index Terms*—Hierarchical split learning, convergence analysis, user grouping.**

Received 18 November 2024; revised 11 March 2025; accepted 2 May 2025. Date of publication 12 May 2025; date of current version 3 September 2025. This work was supported in part by the Peng Cheng Laboratory Major Key Project under Grant PCL2023AS1-5 and Grant PCL2024A01, in part by the Natural Science Foundation of China under Grant 62201311, in part by the Young Elite Scientists Sponsorship Program by CAST under Grant 2023QNRC001, in part by the GuangDong Basic and Applied Basic Research Foundation under Grant 2023B0303000019, and in part by Shenzhen Science and Technology Program under Grant JCYJ20241202125910015. Recommended for acceptance by F. Wang. *(Corresponding author: Wen Wu.)*

Songge Zhang is with the School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen 518055, China, and also with Frontier Research Center, Pengcheng Laboratory, Shenzhen 518055, China (e-mail: zhangsongge@stu.pku.edu.cn).

Wen Wu is with Frontier Research Center, Pengcheng Laboratory, Shenzhen 518055, China (e-mail: wuw02@pcl.ac.cn).

Lingyang Song is with the State Key Laboratory of Advanced Optical Communication Systems and Networks, School of Electronics, Peking University, Beijing 100871, China, also with the School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen 518055, China, and also with the Hunan Institute of Advanced Sensing and Information Technology, Xiangtan University, Xiangtan 411105, China (e-mail: lingyang.song@pku.edu.cn).

Xuemin Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo N2L 3G1, Canada (e-mail: sshen@uwaterloo.ca).

Digital Object Identifier 10.1109/TMC.2025.3569407

I. Introduction

FUELLED by powerful computing servers and well-curated datasets, artificial intelligence (AI) techniques have facilitated a number of applications across a wide range of fields, such as facial recognition, speech recognition, and autonomous driving [1], [2], [3], [4], [5]. Supporting these applications requires training AI models on large user datasets. While centralizing data collection consumes significant bandwidth, slows down training, and raises privacy issues [6], [7], [8]. To enable training with vast amounts of private data in edge networks, edge AI, e.g., federated learning (FL), is a potential paradigm. FL allows AI models to be trained locally on multiple users [9], which still encounters several issues in mobile edge networks due to the limited computing and communication capabilities of users. With the increase of the data size of the state-of-the-art AI models,[1] uploading the model from mobile users to the edge server results in significant transmission delay [12]. Training large AI models also poses significant computation workloads on resource-limited mobile users [13].

Split learning (SL) is an emerging paradigm that can effectively reduce the user-side computational workload as compared to FL. Generally, SL partitions the whole AI model into the *user-side model* (i.e., the first few layers) running on the user and *server-side* model (i.e., the last few layers) running on the edge server. In the SL process, the user first trains the user-side model based on the local data and then transmits the intermediate results to the edge server to complete the forward propagation (FP) phase. Next, in the backward propagation (BP) phase, the edge server updates the parameters of the server-side model and returns the corresponding gradients to the user, thus updating the user-side model. The above process continues until the model is converged. Since only a part of the AI model is trained at the device, the user-side computation workload is reduced compared with the whole AI model training in FL. In addition, small-volume smashed data and user-side models are transmitted, such that communication overhead is reduced compared with whole AI model transmission in FL. In SL, all users interact with the server in a *sequential* manner to complete the training process, such that the training delay is the sum of all users' individual training delays. Hence, the SL scheme suffers from significant training delay for a large number of users.

[1]The data sizes of AlexNet and VGG16 are more than 200 MB and 500 MB, respectively [10], [11].

Recent works have explored several novel schemes to accelerate SL by leveraging parallel mechanisms. Thapa et al. parallelized the training process of multiple user-side and server-side models to reduce training delay [14]. Wu et al. accelerated model training by dividing users into multiple clusters and parallelizing user-side models within each cluster [15]. While the existing works focus on reducing the AI model training delay, the edge server still needs to store a number of server-side models for each individual user, which may exhaust storage resources on the server. Parallelization schemes need to reduce the number of server-side models and efficiently handle smashed data from multiple users to improve training efficiency. Hence, an efficient model training scheme should consider resource constraints in edge networks and accommodate a large number of users.

In this paper, we propose an efficient model training scheme, named Group-based Hierarchical SL (GHSL), which partitions users into groups and parallelizes the training process in each group to reduce training delay and communication overhead. Different from the existing SL schemes, the proposed scheme does not require storing server-side models for each user and is not constrained by the number of smashed data. Specifically, the scheme first divides users into multiple groups, each group collaboratively trained by a server-side model, employing a "first-*sequential*-then-*parallel*" manner. Sequential training entails that each user in each group sequentially trains their user-side model using local data, while the server trains its server-side model using the corresponding smashed data from the users. After each user updates its model, it passes the updated user-side model on to the next user for further training, ensuring a sequential training pattern within each group. Parallel training involves multiple groups and server-side models conducting concurrent training. Once all users complete training, the latest user-side models from each group and their corresponding server-side models are aggregated into a global model for the next round of training. By leveraging parallel training to reduce per-round training delay and sequential training to reduce the number of training rounds, the proposed scheme effectively reduces the overall training delay.

In addition, we theoretically derive the convergence upper bound of the proposed scheme over non-independent and identically distributed (non-IID) data. Analytical results reveal that the convergence rate depends on data heterogeneity and the number of users, which highlights the importance of user grouping to effectively implement the GHSL scheme. Furthermore, we formulate an overall training delay minimization problem by optimizing user grouping. To solve the problem, we propose a data-driven two-stage user grouping algorithm to address the black-box nature of the optimization problem. In the first stage, a Gaussian regression process (GRP)-based algorithm is proposed to determine the number of groups. In the second stage, the coalition game theory is utilized to determine the optimal grouping decision taking user heterogeneity into account. Extensive simulation results demonstrate that the proposed scheme outperforms state-of-the-art benchmarks, reducing the overall training delay by up to 19%, user-side computational workload by up to 53%, and communication overhead by up to 54%.

The main contributions of this paper are summarized as follows:

1) We propose the GHSL scheme to speed up AI model training in edge networks by leveraging parallel training across groups and sequential training within each group.
2) We analyze the convergence rate of the GHSL over non-IID data, which depends on the user grouping strategy.
3) We develop a data-driven user grouping algorithm to minimize the overall training delay, considering heterogeneous user resources and black-box training processes.

The remainder of this paper is organized as follows. Section II provides a review on the related works. Sections III and IV introduce the considered scenario and the proposed GHSL scheme, respectively. Section V presents theoretical convergence analysis of the proposed scheme. In Section VI, we analyze training delay and present the problem formulation. Section VII proposes the two-stage user grouping algorithm. Section VIII showcases the simulation results. Finally, Section IX concludes the paper.

II. Related Work

The SL is an emerging paradigm in distributed learning that has garnered significant attention and research interest in recent years. In the pioneering study, Gupta et al. introduce the concept of SL, detailing the AI model training along with an SL variant [16]. Subsequently, Gao et al. present detailed comparative analysis between SL and FL, evaluating their performance across both IID and non-IID data distribution. The results show that SL has a faster convergence rate than FL [17]. The work in [18] investigates SL's advantage, i.e., it can lessen the user-side computational workload by dividing the AI model and collectively training. The study concludes that SL is suitable for scenarios with a large number of users, while FL is more appropriate for situations with moderate user scale and smaller models. Recent research in the field has been directed towards improving the overall efficiency and reducing the training delay of SL. This is achieved by enhancing both communication and computing efficiencies. Several studies have been focusing on reducing communication overhead by minimizing the volume of data transmitted in each training epoch [19], [20], [21]. Specifically, Gao et al. utilize a combination of image and radio frequency signals to enhance the accuracy of millimeter-wave power prediction, thereby reducing communication costs and enhancing the efficiency of multimodal SL [21]. The strategies are devised to optimize the frequency of model parameter transmission in [22], [23], [24], aiming to further cut down communication overhead. Chen et al. propose a loss-based asynchronous training scheme to minimize the communication overhead of SL in [24]. In this scheme, while the server-side model is trained as usual, the user-side model updates only when the difference in loss compared to the last update exceeds a predefined threshold. Moreover, several studies explore enhancing SL's computing efficiency by optimizing local computation and inference processes on users [25], [26], [27]. Enhancing the computing speed of user-side models locally (both training and inference time) is another approach to improve training efficiency. For example, Pham et al. advocate for the binarization of local SL layers in [27], aiming to achieve faster computation and reduce memory usage. Additionally, Samikwa et al. propose an innovative network architecture coupled with horizontal DNN partitioning

in [28], aiming to speed up the inference process. These works speed up SL by reducing communication overhead or improving computation efficiency, but still suffer from long training delay with a large number of users [29].

Recently, several hybrid schemes have been proposed to enhance SL performance. Implementing distributed model training in edge networks, FL and SL each confront unique challenges. FL struggles with the necessity of training complete models on each user's device, imposing heavy computational workload; SL is limited by the sequential training manner, resulting in significant training delay in the scenarios with a large number of users. The integration of FL and SL presents a compelling solution, potentially maximizing their respective benefits while mitigating their individual drawbacks. Recent research highlighted in [15], [30], [31], [32], has concentrated on combining the strengths of FL and SL to improve the training performance by integrating FL's ability for parallel updates across multiple users with SL's reduced computing demands on user devices. Liu et al. propose a hybrid scheme that enables users to dynamically choose between FL and SL based on their computing capabilities and network conditions in [30]. This scheme demonstrates superior learning accuracy compared to FL alone, and the reduction in communication overhead in comparison to SL. Further hybridization efforts aim to enhance SL's performance by infusing it with FL's collaborative mechanisms. For instance, Wu et al. propose a CPSL framework that divides users into groups, wherein multiple user-side models within a group are trained parallelly based on local data, and different groups engage in sequential training with a single server-side model [15]. Yin et al. present a different scheme that allows for parallel training of two SL processes simultaneously [31]. This scheme enables each user to start training a new user-side model training immediately after one training round ends, which aims to reduce the training delay. Finally, Xu et al. enhance the AI model training speed by parallelizing the training of multiple user-side and server-side models in [32], where each user collaborates exclusively with a corresponding server-side model for synchronized training. These works accelerate SL through parallel training or pipeline training.

Different from these hybrid schemes, the proposed scheme involves partitioning users into groups. In each group, users perform sequential training and share a server-side model. Across these groups, training is in parallel. We further provide convergence analysis under non-IID data. To further reduce overall delay, we develop a data-driven two-stage user grouping algorithm to determine the number of groups and optimal grouping strategies.

III. Considered Scenario

In this paper, we consider a typical edge network scenario, as shown in Fig. 1. The scenario is composed of the users and an edge server, which can be described as follows.

• *Users:* Mobile devices possess their local data and computing capabilities but are constrained by computation power. They can train partial DNN models, i.e., user-side model.

• *Edge Server:* A wireless access point (AP) is equipped with an edge server. The AP has high computing capability and can collect network information, such as communication link conditions. Another function of the AP is to perform model aggregation, including server-side and user-side model aggregation. Instead of deploying a separate aggregation server, the edge server handles model aggregation due to its low computational overhead, which can also avoid communication overhead for transmitting server-side models to an additional aggregation server.

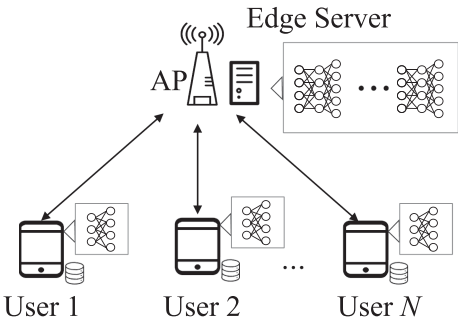


Fig. 1. Considered scenario.

The set of user is denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$, where N represents the number of users. Each user $n \in \mathcal{N}$ has its own dataset $\{(\mathbf{x}_{n,1}, y_{n,1}), \ldots, (\mathbf{x}_{n,D_n}, y_{n,D_n})\}$, where D_n is the size of the dataset, and $\{\mathbf{x}_{n,1}, \ldots, \mathbf{x}_{n,D_n}\}$ represent the raw data and $\{y_{n,1}, \ldots, y_{n,D_n}\}$ are the corresponding labels. The total amount of data from all users is $D = \sum_{n\in\mathcal{N}} D_n$. Users have their computing capabilities and can train user-side models based on the input data. The user-side model for each user n is defined as $\mathbf{w}_n^u$. The AP acts as an edge server with superior computing power compared to users and possesses access to network information, such as communication link conditions. In the context of SL, the AP trains a server-side model using the intermediate parameters uploaded by users. The server-side model is defined as $\mathbf{w}^s$. The whole trained model is represented by

$$\mathbf{w} = \{\mathbf{w}^u, \mathbf{w}^s\}. \tag{1}$$

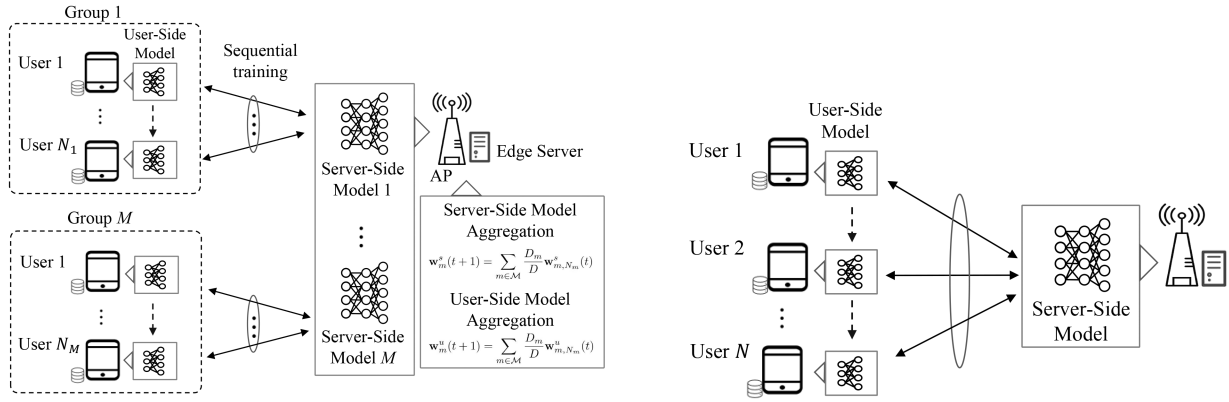
A summary of important notations in this paper is given in Table I.

In the GHSL scheme, individual users employ SL to collaboratively train a shared model. Each user $n \in \mathcal{N}$ trains their user-side model $\mathbf{w}_n^u$ based on their local dataset. Simultaneously, the server trains the server-side model $\mathbf{w}_n^s$ using the intermediate outputs uploaded by users. The goal of the system is to jointly optimize the user-side and server-side models to minimize the global loss function:

$$L(\mathbf{w}) = \frac{1}{N} \sum_{n\in\mathcal{N}} L_n(\mathbf{w}_n^u, \mathbf{w}_n^s), \tag{2}$$

where the $L_n(\cdot, \cdot)$ is the loss function for user n.

In the vanilla SL scheme, user-side models are trained sequentially, with each user training its model one after another, as shown in Fig. 2(b). Once a user completes its training, the user-side model parameters are then passed on to the next user for further training. Each training round continues until all users have finished training. However, SL's sequential training manner results in significant training delay, particularly when there are



(a) Proposed GHSL (b) Vanilla SL

Fig. 2. (a) In the GHSL scheme, users within each group are trained sequentially, and the groups are trained parallelly; (b) in the vanilla SL scheme, users are trained sequentially.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
$\alpha^F_{m,n}$,$\alpha^B_{m,n}$	FP and BP computation workloads for user-side model
$\alpha^F_{s,m}$,$\alpha^B_{s,m}$	FP and BP computation workloads for server-side model
$\beta_{m,n}$	Batch size of user n in group m
B	Transmission bandwidth
D_n	The dataset size of user n
η_s,η_u	Learning rates of server-side and user-side models
f_s	CPU frequency of the server
$f_{m,\,n}$	CPU frequency of the user n in group m
$R(\cdot)$	Training round function
$k(\cdot)$	Covariance function in GPR
ℓ	Lipschitz constant parameter
l	Length scale of kernel function parameter
$L\,(\cdot)$	Global loss function
$L_{m,n}\,(\cdot)$	Local loss function for user n in group m
$\nabla L_{m,n}\,(\cdot)$	Gradient of the output layer
$T(\cdot)$	Total training delay in each round
$\mathcal{N}$	The set of users
$m(\cdot)$	Mean function in GPR
M	The number of groups
P_n,P_s	Transmission power of the user and server
Ω_s, Ω_g	Smashed data and cut layer gradient size for user
$\Omega_{m,n}$	User-side model parameter size of user n in group m
σ	Output variance of kernel function parameter
$\mathbf{w}^u_{m,n}(t)$	User-side model of user n in group m
$\mathbf{w}^s_{m,n}(t)$	Sever-side model of user n in group m
$\mathbf{w}^u_m(t)$	Aggregated user-side model in group m
$\mathbf{w}^s_m(t)$	Aggregated server-side model in group m

a large number of users [33]. To address this issue, we propose a novel group-based scheme that aims to speed up the model training process.

IV. PROPOSED GHSL SCHEME

In this section, we propose a GHSL scheme designed to expedite the training process, as depicted in Fig. 2(a). The comprehensive scheme of the GHSL is detailed in Algorithm 1. To reduce the overall delay in the training process, we first partition users into multiple groups that are parallel-trained. Let M represent the total number of groups within the set $\mathcal{M}$, where the set of users in group m is denoted by $\mathcal{N}_m$ and the last user in group m is denoted as N_m. The proposed scheme consists of three steps: model distribution, model training, and model aggregation. The details are as follows.

1) *Model Distribution:* Model distribution refers to broadcasting the user-side model to users at the beginning of each training round. Specifically, the AP collects the channel conditions and computing capabilities of the users, then partitions them into different groups based on Algorithms 2 and 3. The DNN model is subsequently split into user-side and server-side components. Finally, the initial user-side model is transmitted to the first user in each group through a wireless communication link, initiating the training process for the current round. For the n-th user in the m-th group, let $\mathbf{w}^u_{m,n}(t)$ and $\mathbf{w}^s_{m,n}(t)$ represent the initial models on the user side and the server side, respectively, where $t \in \mathcal{T} = \{1, \ldots, T\}$ denotes the sequence of training rounds. In each training round, the cut layer of the user remains unchanged.

2) *Model Training:* Model training involves model execution, model updating, and model sharing, which is to complete the SL process.

2.1) *Model Execution:* During the training process, data sampling, user-side model's FP, and server-side model's FP are necessary. Specifically, the mini-batch data for user n in group m during the t-th training round is denoted by $\mathbf{b}_{m,n}(t)$. It is extracted from the local data dataset as the input for FP during model training. Then, the user executes the FP phase of model training, inputs the sampled data into the user-side model $\mathbf{w}^u_{m,n}(t)$, and obtains the smashed data of the cut layer. For example,

$$\mathbf{s}_{m,n}(t) = h_1\left(\mathbf{b}_{m,n}(t), \mathbf{w}^u_{m,n}(t)\right), \quad \forall n \in \mathcal{N}_m, \tag{3}$$

where $h_1(\cdot)$ is a function that maps the input layer to the cut layer. Specifically, $h_1(\cdot)$ can be certain layers in a DNN model, such as activation layers, pooling layers, or convolution layers, among others. The $\mathbf{s}_{m,n}(t)$ is obtained after several neural network layer operations and computations, and it will subsequently be uploaded to the AP. After receiving the smashed data from the user, the AP will input it into the server-side model and continue with the FP phase, i.e.,

$$\mathbf{r}_{m,n}(t) = h_2\left(\mathbf{s}_{m,n}(t), \mathbf{w}^s_{m,n}(t)\right), \quad \forall n \in \mathcal{N}_m. \tag{4}$$

Once (3) and (4) are complete, the BP process is finished.

Algorithm 1: Group-Based Hierarchical Split Learning (GHSL) Scheme.

Input: $B,\ \eta_s,\ \eta_u,\ M,$ and N;
Output: $\mathrm{w}^{\star}$;
for *training round* $t = 1, 2, \ldots, T$ **do**
  ▷ Model Distribution
  AP collects the computing capabilities and channel conditions of all the users, and then partitions users into M groups;
  AP broadcasts the latest user-side model to the first users in the group m;
  for *each group in parallel* **do**
    ▷ Model Training
    for *each user* **do**
      ▷ Model Execution
      User draws a mini-batch of data samples $\mathbf{b}_{m,n}(t)$;
      User executes user-side model $\mathbf{w}^{u}_{m,n}(t)$ and obtain smashed data $\mathbf{s}_{m,n}(t)$;
      User transmits smashed data to the AP using wireless link ;
      AP receives smashed data and executes the server-side model $\mathbf{w}^{s}_{m,n}(t)$;
      ▷ Model Updating
      AP updates the server-side model based on (5);
      AP transmits the smashed data's gradient to corresponding users;
      User update the user-side model based on (6);
      ▷ Model Sharing
      User sends the latest user-side model to the next user ;
    end
    The last user N in each group sends the latest user-side model to the AP;
  end
  ▷ Model Aggregation
  AP aggregates server-side models and user-side models into new models based on (8) and (9).
end

2.2) *Model Updating:* Updates are required for both the server-side and the user-side models, which aim to minimize the whole model. Specifically, the AP can compute the loss based on the prediction result $\mathbf{r}_{m,n}(t)$ and ground-truth labels. Then, the AP computes the loss function gradient of the output layer $\nabla L_{m,n}(\mathbf{w}^{s}_{m,n}(t))$ and that of the cut layer $\nabla L_{m,n}(\mathbf{w}^{u}_{m,n}(t))$, where $L_{m,n}(\cdot)$ represents the local loss function of user n in group m. The former is used to update the server-side model in the AP, while the latter is transmitted to the user for subsequent updating of the user-side model. The model update procedure finalizes the BP for both server-side and user-side model training, which can be executed using methods such as stochastic gradient descent, as delineated below:

$$\mathbf{w}^{s}_{m,n}(t) = \mathbf{w}^{s}_{m,n-1}(t) - \eta_s \nabla L_{m,n}\big(\mathbf{w}^{s}_{m,n-1}(t)\big), \quad (5)$$

and

$$\mathbf{w}^{u}_{m,n}(t) = \mathbf{w}^{u}_{m,n-1}(t) - \eta_u \nabla L_{m,n}\big(\mathbf{w}^{u}_{m,n-1}(t)\big), \quad (6)$$

where η_s and η_u are the learning rates for the server-side model and user-side model update, respectively.

2.3) *Model Sharing:* The model sharing stage involves transferring the user-side model from the current user to the next one within the group to continue the training process, which is given by

$$\mathbf{w}^{u}_{m,n+1}(t) \leftarrow \mathbf{w}^{u}_{m,n}(t), \quad \forall n \in \mathcal{N}_m \setminus \{N_m\}, \quad m \in \mathcal{M}. \quad (7)$$

The users sequentially perform the training process until the last user. Finally, the last user sends the latest user-side model to the AP.

3) *Model Aggregation:* Model aggregation is performed at the AP, where the well-trained user-side and server-side models are aggregated into the whole model. The model aggregation stage occurs at the AP and takes place after all users within each group have completed training user-side models and sent them to the AP. This stage involves both server-side model aggregation and user-side model aggregation by using the FedAVG algorithm [34], i.e.,

$$\mathbf{w}^{s}_{m}(t+1) = \sum_{m \in \mathcal{M}} \frac{D_m}{D} \mathbf{w}^{s}_{m,N_m}(t), \quad (8)$$

and

$$\mathbf{w}^{u}_{m}(t+1) = \sum_{m \in \mathcal{M}} \frac{D_m}{D} \mathbf{w}^{u}_{m,N_m}(t), \quad (9)$$

where $D_m = \sum_{n=1}^{N_m} D_n, \forall m \in \mathcal{M}$. After that, the aggregated server-side model and user-side model are adopted as the initial model in the next training round. The training process will continue until a satisfactory level of accuracy is reached.

V. Convergence Analysis

In this section, the convergence of the proposed GHSL scheme is analyzed. For the tractability of converge analysis, key assumptions are given as follows [31].

Assumption 1 (ℓ-Smoothness): The local loss function $L_{m,n}(\cdot)$ of user n in group m is ℓ-smooth with a Lipschitz constant ℓ, i.e.,

$$L_{m,n}(\mathbf{w}_2) \le L_{m,n}(\mathbf{w}_1) + \langle \nabla L_{m,n}(\mathbf{w}_1), \mathbf{w}_2 - \mathbf{w}_1 \rangle + \frac{\ell}{2}\|\mathbf{w}_2 - \mathbf{w}_1\|^2, \quad \forall \mathbf{w}_1, \mathbf{w}_2. \quad (10)$$

Assumption 2 (μ-Strong Convexity): The global loss function satisies a μ-strong convex condition with $\mu > 0$, i.e.,

$$L(\mathbf{w}_2) \ge L(\mathbf{w}_1) + \langle \nabla L(\mathbf{w}_1), \mathbf{w}_2 - \mathbf{w}_1 \rangle + \frac{\mu}{2}\|\mathbf{w}_1 - \mathbf{w}_2\|^2, \quad \forall \mathbf{w}_1, \mathbf{w}_2. \quad (11)$$

Assumption 3 (Bounded Gradient Norm): The expected squared norm of the stochastic gradients for each user is bounded, i.e.,

$$E\|\nabla L_{m,n}(\mathbf{w})\|^2 \le G^2, \quad \forall n \in \mathcal{N}_m, \quad \forall \mathbf{w}. \quad (12)$$

where G represents the upper bound of the expected squared norm of the stochastic gradients.

Assumption 4 (Bounded Divergence): The divergences of the local loss function and global loss functions are bounded as by

$$\frac{1}{N}\sum_{n=1}^{N} \|\nabla L_{m,n'}(\mathbf{w}_{m,n}) - \nabla L_{m,n}(\mathbf{w}_{m,n})\|^2 \le \epsilon^2, \quad \forall n \in \mathcal{N}. \quad (13)$$

where ϵ is also employed to quantify the degree of non-IID data [35].

Lemma 1: During t-th training round, once the training of the n-th user in the m-th group is completed, the expected difference between their model parameter $\mathbf{w}_{m,n}(t)$ and the global parameter $\bar{\mathbf{w}}_{m,n}(t)$ satisfies the following inequality

$$\mathbb{E}\left(\|\bar{\mathbf{w}}_{m,n}(t)-\mathbf{w}_{m,n}(t)\|^2\right)\leq \eta^2 n^2\epsilon^2, \tag{14}$$

where η represents the learning rate and $\bar{\mathbf{w}}_{m,n}(t)=[\mathbf{w}_m^s,\mathbf{w}_m^u]$.

Proof: The proof of this Lemma 1 is provided in the Appendix, available online. □

Lemma 2: The difference between the average parameter of user n and user $n-1$ can be expressed as

$$\mathbb{E}\left(\|\bar{\mathbf{w}}_{m,n}(t)-\bar{\mathbf{w}}_{m,n-1}(t)\|^2\right)=\eta^2\mathbb{E}\left(\left\|\sum_{m=1}^{M}\frac{D_m}{D}\nabla L_{m,n}(\mathbf{w}_{m,n-1}(t))\right\|^2\right). \tag{15}$$

Proof: The proof of this Lemma 2 is provided in the Appendix, available online. □

Lemma 3: The inner product between $\nabla L(\bar{\mathbf{w}}_{m,n-1}(t))$ and $\bar{\mathbf{w}}_{m,n}(t)-\bar{\mathbf{w}}_{m,n-1}(t)$ is bounded as

$$\begin{aligned}&\mathbb{E}\left(\langle\nabla L(\bar{\mathbf{w}}_{m,n-1}(t)),\bar{\mathbf{w}}_{m,n}(t)-\bar{\mathbf{w}}_{m,n-1}(t)\rangle\right)\\&\leq \ell^2\eta^2(n-1)^2\epsilon^2-\frac{\eta}{2}\mathbb{E}\left(\left\|\sum_{m=1}^{M}\frac{D_m}{D}\nabla L_{m,n}(\mathbf{w}_{m,n-1}(t))\right\|^2\right)\\&\quad-\frac{\eta}{2}\mathbb{E}\left(\|\nabla L(\bar{\mathbf{w}}_{m,n-1}(t))\|^2\right).\end{aligned} \tag{16}$$

Proof: The proof of this Lemma 3 is provided in the Appendix, available online. □

Lemma 4: Let $\mathbf{w}^\star$ denote the optimal parameter. For any parameter vector $\mathbf{w}$, there exists a constant $\mu>0$ such that the squared gradient of the loss function is at least twice the discrepancy between the loss and the optimal loss, i.e.,

$$\|\nabla L(\mathbf{w})\|^2\geq 2\mu(L(\mathbf{w})-L(\mathbf{w}^*)). \tag{17}$$

Proof: The proof of this Lemma 4 is provided in the Appendix, available online. □

Theorem 1: The convergence bound for the proposed GHSL scheme after T training rounds is as follows.

$$\begin{aligned}\mathbb{E}[L(\mathbf{w}(T))-L(\mathbf{w}^\star)]&\leq(1-\eta\mu)^{NT}\mathbb{E}[L(\mathbf{w}(0))-L(\mathbf{w}^\star)]\\&+\sum_{t=1}^{T}(1-\eta\mu)^{Nt}\sum_{n=1}^{N}(1-\eta\mu)^n\left(\ell^2\eta^2(N-n)^2\epsilon^2\right.\\&\left.+(\eta^2\ell-\eta)MG^2/2\right),\end{aligned} \tag{18}$$

where $\mathbb{E}(L(\mathbf{w}(T)))$ denotes the expected loss of the trained model after T iterations, and $L(\mathbf{w}^*)$ represents the optimal loss.

Proof: Given Assumption 1, the following inequality holds.

$$\begin{aligned}\mathbb{E}[L(\bar{\mathbf{w}}_{m,n}(t))]&\leq\mathbb{E}[\langle\nabla L(\bar{\mathbf{w}}_{m,n-1}(t)),\bar{\mathbf{w}}_{m,n}(t)-\bar{\mathbf{w}}_{m,n-1}(t)\rangle\\&+L(\bar{\mathbf{w}}_{m,n-1}(t))+\frac{\ell}{2}\|\bar{\mathbf{w}}_{m,n}(t)-\bar{\mathbf{w}}_{m,n-1}(t)\|^2].\end{aligned} \tag{19}$$

By applying Lemmas 2 and 3, (19) can be deduced as

$$\begin{aligned}\mathbb{E}[L(\bar{\mathbf{w}}_{m,n}(t))]&\leq\mathbb{E}\left(L(\bar{\mathbf{w}}_{m,n-1}(t))+4\eta^2G^2\ell^2(n-1)^2\right.\\&-\frac{\eta}{2}\|\nabla L(\bar{\mathbf{w}}_{m,n-1}(t))\|^2-\frac{\eta}{2}\left\|\sum_{m=1}^{M}\frac{D_m}{D}\nabla L_{m,n}(\mathbf{w}_{m,n-1}(t))\right\|^2\\&\left.+\frac{\ell}{2}\eta^2\left\|\sum_{m=1}^{M}\frac{D_m}{D}\nabla L_{m,n}(\mathbf{w}_{m,n-1}(t))\right\|^2\right)\\&\leq\mathbb{E}\left(L(\bar{\mathbf{w}}_{m,n-1}(t))-\frac{\eta}{2}\|\nabla L(\bar{\mathbf{w}}_{m,n-1}(t))\|^2\right)\\&+\frac{\eta^2\ell-\eta}{2}\mathbb{E}\left(\left\|\sum_{m=1}^{M}\frac{D_m}{D}\nabla L_{m,n}(\mathbf{w}_{m,n-1}(t))\right\|^2\right)\\&+\ell^2\eta^2(n-1)^2\epsilon^2\\&\overset{(a)}{\leq}\mathbb{E}\left(L(\bar{\mathbf{w}}_{m,n-1}(t))-\eta\mu(L(\bar{\mathbf{w}}_{m,n-1}(t))-L(\mathbf{w}^\star))\right)\\&+\frac{\eta^2\ell-\eta}{2}\mathbb{E}\left(\left\|\sum_{m=1}^{M}\frac{D_m}{D}\nabla L_{m,n}(\mathbf{w}_{m,n-1}(t))\right\|^2\right)\\&+\ell^2\eta^2(n-1)^2\epsilon^2.\end{aligned} \tag{20}$$

where (a) is based on Lemma 4. Particularly, when $\frac{\eta^2\ell-\eta}{2}>0$ and $\eta>1/\ell$, $\mathbb{E}(L(\bar{\mathbf{w}}_t^n)-L(\mathbf{w}^\star))$ adheres to the subsequent relation:

$$\begin{aligned}&\mathbb{E}\left(L(\bar{\mathbf{w}}_{m,n}(t))-L(\mathbf{w}^\star)\right)\\&\leq\mathbb{E}\left((1-\eta\mu)(L(\bar{\mathbf{w}}_{m,n-1}(t))-L(\mathbf{w}^\star))\right)+\ell^2\eta^2(n-1)^2\epsilon^2\\&\quad+\frac{\eta^2\ell-\eta}{2}\mathbb{E}\left(\left\|\sum_{m=1}^{M}\frac{D_m}{D}\nabla L_{m,n}(\mathbf{w}_{m,n-1}(t))\right\|^2\right)\\&\leq\mathbb{E}[(1-\eta\mu)(L(\bar{\mathbf{w}}_{m,n-1}(t))-L(\mathbf{w}^\star))]+\ell^2\eta^2(n-1)^2\epsilon^2\\&\quad+\frac{\eta^2\ell-\eta}{2}MG^2.\end{aligned} \tag{21}$$

Utilizing a recursive approach, we deduce the disparity between the model parameters obtained upon training N users within each group during round t and the optimal parameters as follows:

$$\begin{aligned}&\mathbb{E}\left(L(\bar{\mathbf{W}}(t))-L(\mathbf{w}^\star)\right)\\&\leq(1-\eta\mu)^N\mathbb{E}\left((L(\bar{\mathbf{w}}_{m,1}(t))-L(\mathbf{w}^\star))\right)\\&\quad+\sum_{n=1}^{N}(1-\eta\mu)^n\left(\ell^2\eta^2(N-n)^2\epsilon^2+\frac{\eta^2\ell-\eta}{2}MG^2\right)\\&\leq(1-\eta\mu)^N\mathbb{E}\left((L(\mathbf{W}(t-1))-L(\mathbf{w}^\star))\right)\\&\quad+\sum_{n=1}^{N}(1-\eta\mu)^n\left(\ell^2\eta^2(N-n)^2\epsilon^2+\frac{\eta^2\ell-\eta}{2}MG^2\right).\end{aligned} \tag{22}$$

Upon completing T rounds, we have

$$\mathbb{E}\left(L(\mathbf{W}(T))-L(\mathbf{w}^\star)\right)$$

$$\leq (1-\eta\mu)^{NT}\mathbb{E}\left(L(\mathbf{W}(0))-L(\mathbf{w}^{\star})\right) + \sum_{t=1}^{T}(1-\eta\mu)^{Nt}\sum_{n=1}^{N}(1-\eta\mu)^{n}\left(\ell^{2}\eta^{2}(N-n)^{2}\epsilon^{2} + (\eta^{2}\ell-\eta)MG^{2}/2\right) \quad (23)$$

Hence, Theorem 1 is proved. □

Remark 1: According to Theorem 1, we can conclude that the increase in training rounds leads to the convergence of the proposed GHSL scheme. Additionally, the convergence performance is influenced by user grouping and user data heterogeneity. Specifically, a larger number of groups and a high level of non-IID data result in a large value on the right side term in (18). This enlarges the gap between the global loss and the expected optimal loss, leading to a slow convergence rate.

Remark 2: The number of groups causes a trade-off between the number of training rounds and the per-round training delay. With a fixed number of users, increasing the number of groups reduces per-round training delay, which is contrary to its effect on the number of training rounds due to model aggregation. Additionally, with hierarchical user resources, different grouping strategies impact the delay of a single training round due to the straggler effect, where the delay of each training round is equal to the longest delay within a group. Therefore, group decisions are necessary.

VI. Training Delay Analysis and Problem Formulation

This section introduces the decision variables, analyzes the per-round training delay in the GHSL scheme, and proposes a problem focused on minimizing the overall training delay. In the training process of GHSL, the following decision variables should be determined.

• *Group Number Decision:* The number of groups M remains constant throughout the training process. It is determined at the beginning of the training and based on historical training data. Subsequently, M is treated as a variable subject to the constraint

$$1 \leq M \leq N, \quad M \in \mathbb{Z}^{+} \quad (24)$$

where N represents the total number of users. This constraint ensures that the number of groups does not surpass the number of users. When $M = 1$, the GHSL scheme defaults to the conventional SL scheme, characterized by a single server-side model where all users participate in sequential training within the same group.

• *User Grouping Decision:* In each training round, user grouping is determined based on channel conditions and computing capabilities. The decision is a binary matrix, denoted as $\mathbf{A} \in \mathbb{R}^{N\times M}$, where each element $a_{n,m}$ adheres to the constraint:

$$a_{n,m} \in \{0,1\}, \quad \forall n \in \mathcal{N}, m \in M, \quad (25)$$

and

$$\sum_{m\in M} a_{n,m} = 1, \quad \forall n \in \mathcal{N}. \quad (26)$$

where $a_{n,m} = 1$ indicates that device n is associated to group m; $a_{n,m} = 0$, otherwise.

A. Training Delay Analysis

The delay of each training round is determined by the FP, BP, and the communication process between the user and the server. The specific details are as follows.

1) *Forward Propagation Delay:* The delay induced by FP training comprises two main components: the delay originating from the user-side model's FP and the delay from the server-side model's FP. Let $\alpha^{F}_{m,n}$ represent the user-side model's FP computation workload in floating point operations (FLOPs) based on the cut layer and $\beta_{m,n}$ denote the batch size of user n in group m. The FP training delay for each user per round can be expressed as

$$T^{F}_{m,n} = \frac{\beta_{m,n}\alpha^{F}_{m,n}}{f_{m,n}\zeta}, \quad \forall n \in \mathcal{N}_m, \quad m \in \mathcal{M}. \quad (27)$$

where ζ denotes the number of FLOPs per cycle for each user, and $f_{m,n}$ represents the local computing speed measured in cycles per second for each user's FP phase. Similar to the user's FP phase, $\alpha^{F}_{s,m}$ represents the server-side model's FP computation workload, which varies depending on the layer from the cut layer to the output layer. Without loss of generality, we assume that the AP evenly allocates the computing frequency to the server-side model. The average computing frequency of the server allocated to user n is given by $f_{s,m} = \beta_m\alpha^{F}_{s,m}/T^{F}_{s,m}\zeta_s, \quad \forall m \in \mathcal{M}$, where ζ_s denotes the number of FLOPs cycles for the server and $f_{s,m}$ is the central processing unit capability of the server-side model m. The number of the server-side model is equal to the number of groups. Consequently, we can determine the time cost of server FP training as follows:

$$T^{F}_{s,m} = \frac{\beta_{m,n}\alpha^{F}_{s,m}}{\zeta_s f_{s,m}}, \quad \forall n \in \mathcal{N}_m, \quad m \in \mathcal{M}. \quad (28)$$

2) *Backward Propagation Delay:* Similarly, the training delay associated with the BP process comprises two components: the delay incurred by the server-side models and the delay experienced by the user-side models. We represent the workloads in the server-side and user-side model BP processes as $\alpha^{B}_{m,n}$ and $\alpha^{B}_{s,m}$, respectively. Consequently, the BP delay on the server can be articulated as follows:

$$T^{B}_{s,m} = \frac{\beta_{m,n}\alpha^{B}_{s,m}}{\zeta_s f_{s,m}}, \quad \forall n \in \mathcal{N}_m, \quad m \in \mathcal{M}. \quad (29)$$

The BP delay in each user can be represented as:

$$T^{B}_{m,n} = \frac{\beta_{m,n}\alpha^{B}_{m,n}}{f_{m,n}\zeta}, \quad \forall n \in \mathcal{N}_m, \quad m \in \mathcal{M}. \quad (30)$$

3) *Communication Latency:* The communication delay of the proposed scheme arises from three interdependent processes: (i) The AP broadcasts the partitioned user-side model to the first user in each group via downlink transmission. (ii) During training, each user transmits smashed data to the AP for forward propagation and subsequently receives gradients at the cut layer through backward propagation. (iii) Upon completing local training, each user sequentially forwards its updated user-side model parameters to the next user within the group. After all

intra-group users complete their training cycles, the final user uploads the latest model to the AP for global aggregation.

Phase (i): Initial Downlink Model Distribution. At the beginning of each training round, the AP broadcasts the user-side model to the first user within each group. Let $\Omega_{u,m}, \quad \forall m \in \mathcal{M}$ represent the data size in bits of the user-side model, where the size depends on the cut layer. We define the downlink signal-to-noise ratio (SNR) as γ_s. Consequently, the transmission rate from the AP to the user can be represented as $R_s = B\log_2(1+\gamma_s)$, and the AP-user link transmission delay can be estimated by $T_s^C = \Omega_{u,m}/R_s$.

Phase (ii): Forward-Backward Propagation Exchange. During the collaborative training process with the AP, each user transmits the smashed data generated by the user-side model to the AP. Subsequently, the AP returns the gradients of the cut layer post-training. Let Ω_s and Ω_g represent the data sizes of the smashed data and the gradients, respectively. The response latency are represented as $T_{m,n}^s = \Omega_s/Rm,n$ and $T_{m,n}^g = \Omega_g/Rm,n$.

Phase (iii): Updated User-Side Model Deliver. Upon completing the training, each user delivers the updated user-side model to the subsequent user, with the transmission rate for each user denoted by

$$R_{m,n} = B_{m,n}\log_2\left(1+\gamma_{m,n}\right), \qquad \forall n \in \mathcal{N}_m, \quad m \in \mathcal{M}. \tag{31}$$

where $B_{m,n}$ denotes the bandwidth resources allocated to each group, with the sum of allocations for all groups, $\sum_{m\in\mathcal{M}} B_{m,n} = B$. It should be noted that the scenario employs time division duplexing, thereby allowing the assumption of identical AP-to-user and user-to-user channel conditions due to channel reciprocity. Accordingly, the latency for user-to-user transmission of the user-side model is denoted as $T_{m,n}^C = \Omega_{m,n}/R_{m,n}$. where $\Omega_{m,n}$ represent the well-trained user-side model for user $n \in \mathcal{N}_m$.

4) *Overall Training Delay:* In each training round, the AP distributes the initial user-side model to the first user in each group. Subsequently, within each group, every user sequentially interacts with the AP to complete both the forward and backward propagation processes, which entails the transmission of smashed data and the gradients of the cut layer. Furthermore, the well-trained user-side model is passed on to the next user. It is important to note that the final user in each group is responsible for transferring the last user-side model back to the AP for model aggregation. Hence, we derive that the total training delay for m group is represented by

$$T_m(\mathbf{A},M) = \sum_{n=1}^{N_m}\left(T_{m,n}^F + T_{s,m}^F + T_{m,n}^F + T_{s,m}^B + T_{m,n}^s + T_{m,n}^g + T_{m,n}^C\right) + T_s^C, \qquad \forall m \in \mathcal{M}. \tag{32}$$

The delay in each training round can be represented as

$$T(\mathbf{A},M) = \max_{m\in\mathcal{M}} T_m(\mathbf{A},M). \tag{33}$$

The overall training delay is influenced not only by the delay of each training round but also by the total number of training rounds. We define the total training delay as $D(\mathbf{A},M) = T(\mathbf{A},M)R(\mathbf{A},M)$, where $R(\cdot)$ represents the function for the total number of training rounds.

B. Problem Formulation

To jointly design the edge network and the GSFL scheme, we propose an optimization problem that aims to minimize the total training delay by identifying the optimal group number decision and user grouping decision, which can be expressed as

$$\begin{aligned} \mathcal{P}: \quad & \min_{\mathbf{A},M} D\left(\mathbf{A},M\right) \\ \text{s.t.} \quad & (24), (25), \text{and} (26), \end{aligned} \tag{34}$$

where M is the number of groups in the GHSL scheme, and N is the total number of users. Constraint (24) ensures that the number of groups remains within a feasible range, preventing M from exceeding the total number of users. Constraint (25) defines $\mathbf{A}$ as a binary matrix where each user is assigned to a single group. Additionally, constraint (26) guarantees that each user is exclusively associated with only one group.

This problem is a binary optimization problem that requires a hierarchical decision process, where the number of groups is determined first, followed by the optimization of the user grouping strategy. Solving this problem presents two major challenges. The first challenge arises from the black-box nature of the objective function. The relationship between AI model accuracy and system parameters lacks a closed-form expression, making it infeasible to apply traditional gradient-based optimization methods. Without an explicit mathematical form, direct optimization is impractical, and the relationship between training parameters and model accuracy needs to be explored through patterns observed in empirical training data. The second challenge arises from the 0-1 binary nature of the optimization problem. The objective function is unknown and may be non-convex, making it difficult to find an optimal solution. Additionally, group capacity constraints limit the number of users per group, further increasing the complexity of the binary optimization process.

VII. Two-Stage User Grouping Algorithm

In this section, we propose a two-stage user grouping algorithm, using a Gaussian process regression (GPR)-based algorithm to determine the number of groups and a coalition game-based algorithm to identify the optimal user grouping decision.

A. First Stage: GPR-Based Group Optimization Algorithm

In this paper, we utilize the observational dataset obtained from GHSL and GPR to train a predictive model for the function of training rounds. GPR is a probabilistic regression technique predicated on the assumption that data points conform to a multivariate Gaussian distribution. It utilizes the posterior function

to predict the outputs of unknown data points, as shown in Algorithm 2. We also propose a GPR-based grouping algorithm, as shown in Algorithm 3, to identify the optimal number of user groups. This is achieved by estimating the number of training rounds and calculating the delay per round, with the aim of determining the number of groups that minimize overall training delay.

1) *Collect Dataset:* In the prediction model, we first define the input and output variables. The input variables are the number of users N and the number of group partitions M, which are both integers and have specific constraints. The output variable is the number of training rounds R, which can be obtained from the GHSL scheme. Therefore, our goal is to establish a mapping relationship $f : (N, M) \mapsto H$, which is learned and predicted through the training round prediction model. To establish the prediction model, we consider an observation dataset $\mathcal{D} = \{(N_i, M_i, R_i)\}_{i=1}^{I}$, where I is the total number of data points. Each data point consists of an input pair $(N_i, M_i) = \mathbf{z}_i$ and the corresponding output R_i. Based on these observations, we can construct a GPR model to estimate the function $H(\mathbf{z}_i) = f(\mathbf{z}_i) + \epsilon$, where ϵ is the noise term added to the observations, usually assumed to be Gaussian noise. When collecting observational data, the system lacks information on resource heterogeneity. Therefore, we assume the users are homogeneous, and each group has identical users.

The GPR-based user grouping algorithm primarily encompasses the following stages:

2) *Define Prior Function:* we assume that the function $H(\mathbf{z}_i)$ is a Gaussian process, which means that for any set of inputs $\mathbf{z}_i$, the output of the function $H(\mathbf{z}_i)$ follows a Gaussian distribution. The $H(\mathbf{z}_i)$ is a prior function capturing the data distribution before observing any actual data, which can be defined as

$$H(\mathbf{z}_i) \sim \mathcal{GP}(m(\mathbf{z}_i), k(\mathbf{z}_i, \mathbf{z}_j)). \tag{35}$$

where $\mathcal{GP}$ denotes a Gaussian process, $m(\mathbf{z}_i)$ is the mean function typically assumed to be zero, and $k(\mathbf{z}_i, \mathbf{z}_j)$ is the covariance function.

3) *Compute Covariance Matrix:* The kernel function determines the similarity between any two points in the input space. The covariance matrix $\mathbf{K}$ is calculated by

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{z}_1, \mathbf{z}_1) & \cdots & k(\mathbf{z}_1, \mathbf{z}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{z}_t, \mathbf{z}_1) & \cdots & k(\mathbf{z}_t, \mathbf{z}_t) \end{bmatrix}. \tag{36}$$

In the proposed algorithm, the kernel function is defined as a radial basis function (RBF), also known as the Gaussian kernel, which can be given by

$$k(\mathbf{z}_i, \mathbf{z}_j) = \sigma^2 \exp\left(-\frac{1}{2l^2}\left((N_i - N_j)^2 + (M_i - M_j)^2\right)\right), \tag{37}$$

where the kernel function parameters σ^2 and l, denoting output variance and length scale, respectively, are encapsulated in the parameter set $\boldsymbol{\theta}$. This relationship can be expressed as $\boldsymbol{\theta} = \{\sigma^2, l\}$. These parameters need to be estimated from the observation dataset to optimize the kernel function. In prediction model training, our goal is to learn the parameters of the kernel function and to make predictions for unknown points $\mathbf{z}^\star$ based on the observed training data. Training the GPR model aims to maximize the log-likelihood function, which is usually achieved through iterative optimization algorithms, such as gradient ascent.

Algorithm 2: Gaussian Process Regression Algorithm.

1: **Input:** Training dataset $\mathcal{D} = \{(N_i, M_i, T_i)\}_{i=1}^{I}$, the kernel function, learning rate ϕ;
2: **Output:** Predictive model $H(N, M)$;
3: **Initialize:** Define a Gaussian process $\mathcal{GP}$ and initialize kernel function parameters θ;
4: Construct covariance matrix $\mathbf{K}$ for the training input based on the kernel function ;
5: Calculate the log-likelihood function in (38) with the observed data under the $\mathcal{GP}$ prior;
6: Calculate the gradient of L with respect to the kernel function parameters θ;
7: Update kernel function parameters $\boldsymbol{\theta}$ iteratively using gradient descent until convergence:
$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L};$$
8: Calculate the posterior distribution for new input point $\mathbf{z}^\star$ based on (41) and (42);
9: Output the posterior mean $\mu^\star$ and variance $(\sigma^*)^2$ of the predictive function $H(N, M)$;

4) *Optimize Kernel Function:* With the prior and kernel defined, the next step in GPR training involves optimizing the kernel parameters. This optimization is achieved by maximizing the log-likelihood function with the observed data and the prior function. The log-likelihood function serves as an indicator of the model's efficacy, with a given set of kernel parameters, in explaining the observed training data. It is formulated as

$$\mathcal{L} = -\frac{1}{2}\log\left(|\mathbf{K} + \sigma_n^2\mathbf{I}|\right) - \frac{1}{2}\mathbf{y}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y} - \frac{I}{2}\log 2\pi, \tag{38}$$

where $\mathbf{y}$ represents the target values of the training data, σ_n^2 is the variance of the noise term, and $\mathbf{I}$ is the identity matrix. Maximizing this function allows us to fine-tune the kernel parameters to best fit the training data.

To maximize the likelihood function, the proposed algorithm employs a method, such as gradient ascent, to identify the optimal kernel parameters. In each iteration, the update of parameters can be achieved by calculating the gradient of the likelihood function with respect to the parameters. In the gradient ascent algorithm, the parameter update rule can be represented as

$$\theta' = \theta + \phi \nabla_\theta \mathcal{L}, \tag{39}$$

Where θ represents the parameters of the kernel function, ϕ is the learning rate, and $\nabla_\theta \mathcal{L}$ is the gradient of the likelihood function with respect to the parameter θ.

Algorithm 3: Optimal Group Number Determination Algorithm.

Input: User count N and group decision $\mathbf{A}^\star$;
Output: Optimal group number $M^\star$;
Initialize: Initialize an empty list T_{total} to store total training delay for different group numbers;
for $m = 1, \ldots, N$ **do**
  Determine the number of training rounds $H(m, N)$ for user count N and group number m using Algorithm 2's predictive function;
  Calculate per-round computation delay $T(\mathbf{A}^\star, N)$ for user count N and group number m using (33);
  Compute total training delay $T(\mathbf{A}^\star, M) \times H(m, N)$;
  Append $T(\mathbf{A}^\star, M) \times H(m, N)$ to T_{total};
end for
Find the minimum value in T_{total} and its corresponding group number $M^\star$.

5) *Calculate Posterior Function:* Upon determining the optimal parameters, the next step involves computing the posterior function. This function characterizes the conditional distribution of the output $R^\star$ for new inputs $\mathbf{z}^\star$, contingent upon the training dataset $\mathcal{D}$. The posterior function is formulated as

$$P\left(H(\mathbf{z}^\star)|\mathcal{D}\right) \sim \mathcal{N}(\mu^\star, (\sigma^*)^2). \tag{40}$$

The mean $\mu^\star$ and variance $(\sigma^*)^2$ of the posterior distribution are computed using the optimized kernel parameters, the training data inputs $\mathbf{Z} = \{(N_i, M_i)\}_{i=1}^I$, and outputs $\mathbf{y} = \{R_i\}_{i=1}^I$. The mean of the posterior function is defined as

$$\mu^\star = k(\mathbf{z}^\star, \mathbf{Z})^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{y}. \tag{41}$$

The posterior variance $(\sigma^*)^2$, which quantifies the uncertainty in the predictions at new input points, can be expressed as

$$(\sigma^*)^2 = k(\mathbf{z}^\star, \mathbf{z}^\star) - k(\mathbf{z}^\star, \mathbf{Z})^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} k(\mathbf{z}^\star, \mathbf{Z}). \tag{42}$$

Computational Complexity Analysis: The computational complexity of the Algorithm 2 is critically analyzed, focusing on its two most computationally intensive operations: the construction of the covariance matrix and its inversion. The construction of the covariance matrix, $\mathbf{K}$, is executed by evaluating a kernel function across all pairs of training points, yielding a complexity of $O(I^2)$, where I represents the total number of training data points [36]. The predominant computational demand arises during the matrix inversion step and is integral to both the log-likelihood computation and parameter updating process. This operation exhibits a computational complexity of $O(I^3)$, utilizing standard numerical techniques such as lower-upper decomposition. Given the algorithm iterates over G iterations, the overall training phase complexity scales as $O(GI^3)$. During the prediction phase, calculating the posterior mean and variance for a new input involves operations with the pre-computed inverse covariance matrix, maintaining a complexity of $O(I^2)$.

Based on Algorithm 2, the predicted number of training rounds can be deduced given the number of users and groups. Furthermore, the training delay for each round is calculable via (33). By iterating over all possible scenarios for M, the optimal group number $M^\star$ that minimizes the total delay, as defined in (43), can be identified. The entire procedure is outlined in Algorithm 3.

B. Second Stage: Coalition Game-Based User Grouping Strategy Algorithm

The user grouping sub-problem aims to determine the optimal user grouping decision, expressed mathematically as

$$\begin{aligned} \mathcal{P}_1: \quad & \min_{\mathbf{A}, M^\star} D\left(\mathbf{A}, M^\star\right) \\ \text{s.t.} \quad & (25) \text{ and } (26). \end{aligned} \tag{43}$$

where $D(\mathbf{A}, M^\star)$ represents the total training delay under the optimal number of groups $M^\star$. The optimization variables are the binary user grouping decisions, dependent on users' computing capabilities and channel conditions, thus making this sub-problem a binary optimization problem with group capacity constraints. To effectively tackle this issue, we develop a user grouping algorithm grounded in coalition game theory. The most important aspect of the coalition game setting is the formation of coalitions. Specifically, each user has different preferences over potential coalitions and adopts the preference relation to compare any two collections of coalitions. In this regard, we present the following definition of preference relation.

• *Preference Order:* For any user $n \in \mathcal{N}_m$, the preference order $\succeq_n$ is defined as a complete, transitive, and reflexive binary relation over the set of all partitions that user n can possibly form.

In the user grouping algorithm, users can choose to join or leave a coalition based on their preference order. Specifically, a user will opt to join a coalition that it prefers over others. For any user $n \in \mathcal{N}_m$, considering two partitions $\mathcal{U}_m$ and $\mathcal{U}_{m'}$ of the grouping strategy $\mathbf{A}$, $\mathcal{U}_m \succeq_n \mathcal{U}_{m'}$ indicates that user n prefers being part of coalition $\mathcal{U}_m$ over $\mathcal{U}_{m'}$. This does not account for scenarios where user n holds equal preference for $\mathcal{U}_m$ and $\mathcal{U}_{m'}$. The preference operation $\mathcal{U} \succeq_n \mathcal{U}'$ is defined as

$$\mathcal{U}_m \succ_n \mathcal{U}_{m'} \Leftrightarrow R(\mathcal{U}_m) + R(\mathcal{U}_{m'} \setminus n) > R((\mathcal{U}_m \setminus n) + R(\mathcal{U}_{m'}). \tag{44}$$

This definition implies that user n prefers membership in coalition $\mathcal{U}_m$ over $\mathcal{U}_{m'}$, provided that such a strategy decreases the overall training delay. Users will execute strategy switches based on their preference order in accordance with the switching rules defined below.

• *Switch Operation:* Given a partition $\mathbf{A} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_{M^\star}\}$ of the user group strategy, if a user $n \in \mathcal{N}_m$ performs a switch from $\mathcal{U}_m$ to $\mathcal{U}_{m'}$ where $\mathcal{U}_m \neq \mathcal{U}_{m'}$, the current partition $\mathbf{A}$ is modified to a new partition $\mathbf{A}'$ such that $\mathbf{A}' = (\mathbf{A} \setminus \{\mathcal{U}_m, \mathcal{U}_{m'}\}) \cup \{\mathcal{U}_m \setminus \{n\}, \mathcal{U}_{m'} \cup \{n\}\}$.

The system is initialized with any random coalition partition $\mathbf{A} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_{M^\star}\}$. For each user $n \in \mathcal{N}_m$, it is assumed

that the current coalition is $\mathcal{U}_m$, where $\mathcal{U}_m \subseteq \mathbf{A}$. We then randomly select another coalition $\mathcal{U}_{m'}$ and assume the preference $\mathcal{U}_{m'} \succ_n \mathcal{U}_m$ is satisfied, meaning a switch from $\mathcal{U}_m$ to $\mathcal{U}_{m'}$ will occur, updating the current coalition partition to the new partition $\mathbf{A}'$ as defined. The switch operation is executed only if the preference relation defined in (44) is met. In this mechanism, every user $n \in \mathcal{N}_m$ can leave its current coalition and join another, provided that the new coalition is strictly preferred according to the definition in (44). Additionally, the user grouping strategy must significantly decrease the overall training delay in the new coalition. Generally, the proposed coalition formation game aims to find a coalition structure that maximizes total utility rather than individual player payoffs.

Based on the switch rule defined previously, we develop a coalition formation algorithm for user grouping, as detailed in Algorithm 4. In the proposed algorithm, users execute switch operations until reaching a Nash equilibrium partition. The proposed algorithm is described as follows:

1) The algorithm is activated by selecting a random coalition partition, which is then designated as the current partition $\hat{\mathcal{U}}$. The iteration counter and the counter for consecutive unsuccessful switch operations, denoted by iter and num, respectively, are both reset to zero.
2) User n is randomly selected based on a predetermined permutation. The user n then selects an alternate coalition $\mathcal{U}_{m'}$ distinct from its current coalition $\mathcal{U}_m$. Following this, it calculates the utilities for both coalitions, culminating in a decision on whether to switch.
3) If user n satisfies the preference relation defined in (44), it notifies the affected coalitions of the switch, prompting an update to the current coalition partition.
4) To enhance the convergence rate and minimize the algorithm's complexity, the algorithm leverages the count of consecutive unsuccessful switch operations, num. This count is reset to zero following a successful operation. If not, it increments by one. The proposed algorithm terminates when num reaches ten times the number of users, at which point the algorithm converges to the final Nash equilibrium.

Theorem 2: Regardless of the initial alliance partition $\mathbf{A}_0$, Algorithm 4 converges to a final partition $\mathbf{A}^\star$, consisting of multiple disjoint alliances.

Proof: In our proposed algorithm, the number of users is finite, and the number of groups can be determined to be fixed through Algorithm 3. Thus, the maximum number of partitions that Algorithm 4 can form is $M^\star$. Based on the preference relationships in (44), and the switching rules defined in Definition 4, each user can autonomously decide its potential coalitions, aiming to improve system utility by selecting a new partition. Moreover, for a given set of users $\mathcal{N}_m$, the number of partitions is a Bell number. As each switching operation creates a new partition, and the number of partitions is finite, Algorithm 4 ensures convergence to the final Nash equilibrium $\mathbf{A}^\star$.

Computational Complexity Analysis: The complexity of Algorithm 4 primarily depends on the frequency of coalition switching operations. In each iteration, randomly selected user equipment assesses its preferences for its current and potential coalitions and computes the total benefits these coalitions offer, which is the total training delay under the GHSL scheme. If users prefer the potential coalition, they will switch from their current one to the new one. Each iteration involves the selection of only one user, preventing multiple operations. If K denotes the number of iterations, then the computational complexity of Algorithm 4 is $O(K)$.

Algorithm 4: Coalition Game-Based User Grouping Algorithm.

Input: Device computing capabilities, channel conditions, and optimal number of groups $M^\star$;
Output: User grouping decision $\mathbf{A}^\star$;
Initialization: Randomly select a feasible coalition partition $\mathbf{A}_0$ of the user set $\mathcal{N}_m$, and initialize $\hat{\mathbf{A}} = \mathbf{A}_0$, $k = 0$, and $P = 0$;
repeat
  Increase $k = k + 1$;
  Select a user $n \in \mathcal{N}_m$ by a predetermined permutation and identify its current coalition $\mathcal{U}_m \in \hat{\mathbf{A}}$;
  Randomly select another coalition $\mathcal{U}_{m'} \in \hat{\mathbf{A}}$, ensuring $\mathcal{U}_{m'} \neq \mathcal{U}_m$;
  if Preference relation $\mathcal{U}_{m'} \succ_n \mathcal{U}_m$ is satisfied **then**
    User n leaves its current coalition $\mathcal{U}_m$ and joins $\mathcal{U}_{m'}$;
    Update $\hat{\mathbf{A}}$ according to switch rule by
    $\hat{\mathcal{U}} \leftarrow \{\hat{\mathcal{U}} \setminus \{\mathcal{U}_{m'}, \mathcal{U}_m\}\} \cup \{\mathcal{U}_m \setminus \{n\}, \mathcal{U}_{m'} \cup \{n\}\}$;
    Reset $P = 0$;
  else
    Increase $P = P + 1$;
  end if
until $\hat{\mathbf{A}}$ converges to a Nash equilibrium partition $\mathbf{A}^\star$.

VIII. Performance Evaluation

A. Simulation Setting

In this simulation, we utilize three datasets for image classification [37]: GTSRB, with over 50,000 images in 43 traffic sign categories; F-MNIST, a collection of 70,000 grayscale images across 10 fashion-related classes; and CIFAR-10, comprising 60,000 color images in 10 varied categories including animals and vehicles. For non-IID settings, we implement the Dirichlet distribution for data allocation to follow the non-IId characteristics of data, reflecting the diversity of data collected by various devices. Specifically, we employed the method described in [35], [38] to synthesize data for non-identical clients. Each client draws data from each category, denoted by q_i, where the vector $q = (q_1, q_2, \ldots, q_i)$ follows a Dirichlet distribution, i.e., $q \sim \mathrm{Dir}(\Omega_1 p)$. Here, p represents the categories of the data, and Ω_1 is a parameter controlling the uniformity across clients. This parameter ranges from 0, where each client randomly holds only one category, to infinity, approximating an i.i.d. scenario where

TABLE II
SIMULATION PARAMETERS

Parameter	Value
CPU capability of the user, f_s	2 GHz
CPU capability of the edge server, $f_m(n)$	100 GHz
Batch size	128
Number of users, N	30
Bandwidth, B	30 MHz
SNR	17 dB
User-side model learning rate, η_u	0.01
Server-side model learning rate, η_s	0.01
Dirichlet coefficient, Ω_1	0.5

all clients can access all categories. For principal simulation parameters, refer to Table II.

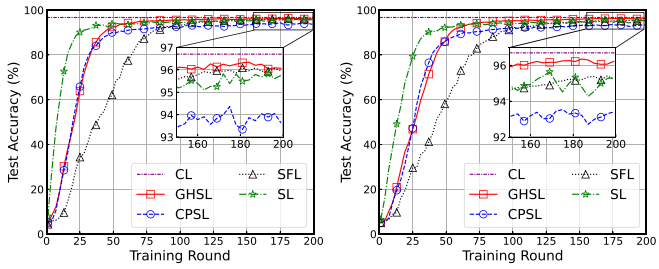
Our simulation evaluation includes three models: a 27-layer convolutional neural network (CNN) comprising 7 fully connected layers and 20 convolutional layers. The seventh convolutional layer is designated as the cut layer. The user-side model, including these 7 convolutional layers, occupies 0.074 MB and requires 19.92 MFlops for FP and 39.84 MFlops for BP. The server-side model occupies 22.315 MB, with forward and backward propagations requiring 118.23 MFlops and 236.46 MFlops, respectively. Additionally, the sizes of the smashed data and the gradients at the cut layer are 0.064 MB and 0.064 MB, respectively. The other two models evaluated are ResNet18 and LeNET, with user-side model sizes of 0.0071 MB and 0.0054 MB, and server-side sizes of 42.618 MB and 1 MB, respectively, Table III. The CNN model is the default configuration for our simulations unless specified otherwise.

We compare the proposed scheme with the following three benchmarks:

1) *Centralized Learning (CL):* We train all the data on a complete model, which achieves the highest accuracy that can be reached by all benchmarks and the proposed GHSL scheme.
2) *SL [16]:* Users are trained sequentially. Each user locally trains the user-side model and interacts with a server to train the server-side model. The latest user-side model is then passed to the next user.
3) *Split Federated Learning (SFL) [32]:* Multiple users are trained in parallel. Upon the completion of each training round, all user-side and server-side models are aggregated.
4) *Group-based Parallel Split Learning (CPSL) [15]:* In the CPSL scheme, users are organized into multiple groups. Within each group, users simultaneously train several user-side models and update a shared server-side model. Across groups, the training is sequential.

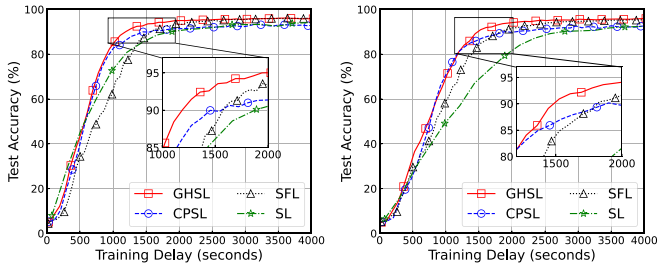
B. Simulation Results

1) Training Performance of the GHSL Scheme: 1) *Accuracy Performance:* Fig. 3(a) and (b) show the accuracy performance of the global model with the number of training rounds in both IID and non-IID settings. The results indicate that the accuracy performance of the GHSL scheme is comparable to that of SL and centralized learning (CL), mainly due to the sequential training by the users within the group. However, GHSL requires more training rounds to achieve the same accuracy, which is due to the model aggregation process. Furthermore, it is observed a decrease in the accuracy of the CPSL scheme, which might be due to information loss caused by concatenating compressed data into a matrix.



(a) IID (b) Non-IID

Fig. 3. Accuracy performance of CNN on the GTSRB dataset varies with each training round.



(a) IID (b) Non-IID

Fig. 4. Accuracy performance of CNN on the GTSRB dataset varies with each training delay.

TABLE III
AI MODEL PARAMETERS

Architecture	# Parameters	# Layers	Comp. (MFlops)
LeNet [15]	4.3 million	12	91
CNN [15]	5.8 million	27	138
ResNet18 [39]	11.2 million	18	560

2) *Delay Performance:* The total training delay is the product of the number of training rounds and per-round training delay. Fig. 4(a) and (b) display the assessment of total training delay for various schemes. Under both IID and non-IID settings, the proposed GHSL scheme consistently outperforms the baseline in accelerating model training delay. Specifically, the training delay for the GHSL, SFL, CPSL, and SL schemes are about 1,782 seconds, 2,110 seconds, 2,502 seconds, and 3,402 seconds under non-IID data, respectively. This variation is attributed to the difficulty CPSL faces in converging with larger batch sizes, while SFL is influenced by an increase in the number of training rounds due to the aggregation of multiple server-side models. Additionally, the per-round training delay for GHSL, CPSL, SFL, and SL are 29.7 seconds, 20.1 seconds, 27.8 seconds, and 75.6 seconds, respectively. Table IV presents the performance of the proposed GHSL scheme across various datasets and models. Simulation results indicate that the GHSL scheme achieves the required accuracy more rapidly than other benchmarks on the MNIST dataset with the LeNet model, a finding also replicated in

TABLE IV
PERFORMANCE COMPARISON OF THE PROPOSED SCHEME AND BENCHMARKS

Dataset	Scheme	IID				non-IID			
		#Round	Delay (seconds)	Comp. (GFlops)	Comm. (GB)	#Round	Delay (seconds)	Comp. (GFlops)	Comm. (GB)
MNIST with **LeNet**	SL	21	850.5 (2.5×)	1.06	1.44	26	1053.0 (2.8×)	1.31	1.79
	SFL	96	895.4 (2.6×)	4.84	6.70	138	1287.1 (3.4×)	6.96	9.63
	CPSL	44	647.3 (1.9×)	2.22	3.07	58	853.2 (2.2×)	2.92	4.05
	GHSL	25	**340.7** (1.0×)	1.26	1.72	28	**381.6** (1.0×)	1.41	1.93
CIFAR-10 with **ResNet18**	SL	93	2.78×10^4(2.7×)	654.95	90.11	128	3.83×10^4(3.2×)	901.44	124.03
	SFL	403	3.25×10^4(3.1×)	2838.13	393.87	562	4.53×10^4(3.7×)	3957.89	549.27
	CPSL	174	2.06×10^4(2.0×)	1225.40	170.06	196	2.32×10^4(1.9×)	1380.33	191.56
	GHSL	109	**1.03** $\times 10^4$(1.0×)	767.63	105.77	128	**1.21** $\times 10^4$(1.0×)	901.43	124.20

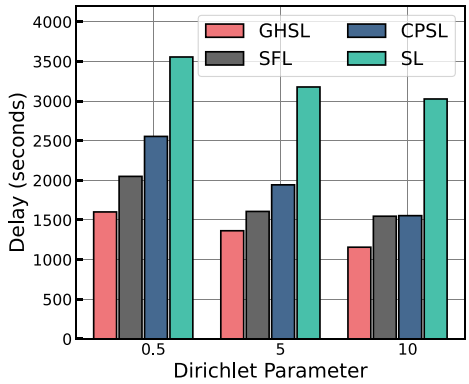


Fig. 5. The impact of the non-IID level.

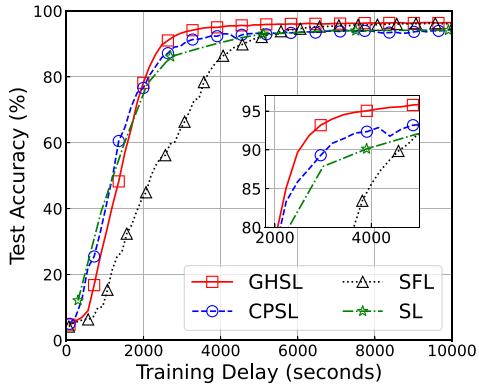
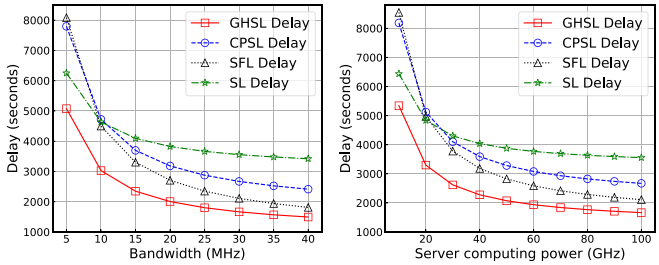


Fig. 6. Train delay performance for a large number of users ($N = 100$).

simulations using the CIFAR-10 dataset with the ResNet model. The results confirm the robustness of the proposed scheme. Particularly, when compared with CPSL and SFL, GHSL can reduce up to 3.4 times the delay consumption to reach the required accuracy of 96% on the MNIST dataset under a non-IID setting. When compared with SFL, GHSL can reduce up to 2.9 times delay consumption to reach the required accuracy of 80% on CIFAR-10 under the non-IID setting.

3) *System Overhead:* Table IV illustrates that the GHSL training incurs lower system overhead, including reduced total communication costs and user computing requirements. GHSL benefits from excellent convergence due to its sequential training within groups, resulting in minimal system cost compared to other hybrid benchmarks. However, compared to the vanilla SL scheme, GHSL incurs higher communication costs and computational workloads due to increased training rounds prompted by model aggregation, though these costs remain within acceptable limits.

4) *Impact of Data Distribution:* We explore the training performance under non-IID settings. As shown in Fig. 5, simulation results demonstrate that the proposed GHSL scheme consistently achieves the minimum training delay across varying degrees of non-IID data distribution, thereby proving the scheme's robustness. Moreover, compared to the CPSL scheme, our scheme exhibits significantly greater performance gains under highly non-IID. For instance, as the Dirichlet coefficient decreases from 10 to 0.5, the gain increases by 30%. This is because the proposed scheme avoids concatenating smashed data. When the data distribution varies significantly, large input sizes can degrade training performance.

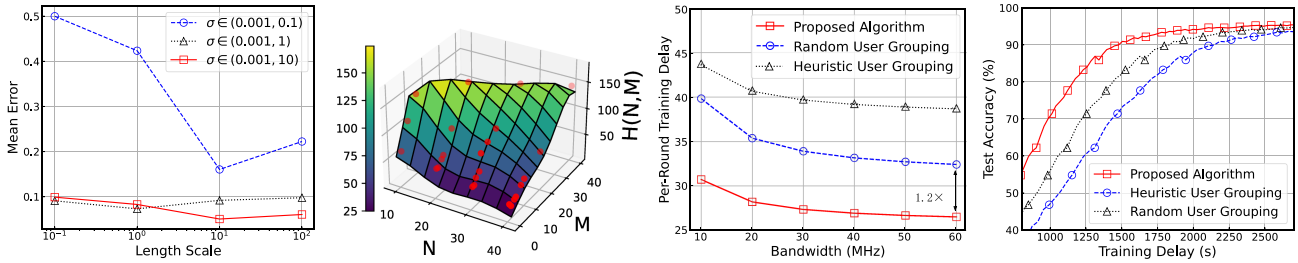


(a) Bandwidth resource (b) Computing resource

Fig. 7. Training delay performance under different amount of bandwidth and computing resource.

5) *Impact of User Number:* We further analyze training performance under scenarios involving a large number of users in Fig. 6. Specifically, we evaluate the performance of three hybrid schemes with 100 users. The simulation results indicate that the proposed GHSL scheme maintains the lowest training delay. Additionally, GHSL exhibits a significant delay performance improvement over SFL as the number of users increases, achieving approximately a 70% reduction. This improvement is attributed to the increasing number of server-side models associated with a larger user base, which typically complicates model convergence in SFL. Our proposed scheme, along with the CPSL scheme, features a grouped, parallel design that effectively addresses the challenges posed by large-scale user environments.

6) *Impact of Network Resource:* We assess the impact of resource heterogeneity on training performance, as depicted in Fig. 7(a) and (b), which respectively show training delay variations with changes in system bandwidth and server computing



(a) Variation of mean error with kernel parameters (b) Validation of training round estimation (c) Per-round training delay with different grouping algorithms (d) Overall training delay taking the proposed grouping algorithm

Fig. 8. Performance of the proposed GPR-based group optimization algorithm and coalition game-based user grouping strategy algorithm.

resources. Although these system resources do not directly affect the training process, they influence communication and computation delay, necessitating theoretical calculations of latency for each round. In the results, the proposed GHSL scheme demonstrates superior capability in adapting to changes in system resources, consistently maintaining the shortest training delay. Furthermore, the benefits of the GHSL scheme increase under resource constraints: delay decreased by an average of 30% with a system bandwidth of 40 MHz and by 34% with a bandwidth of 5 MHz, compared to SFL. Similar outcomes were observed with variations in server computing power. As Table IV shows, reaching specified test accuracy requires increased communication overhead and computation Flops, leading to greater training delay when system resources are limited. Compared to SFL and CPSL, SL performs better when available resources are limited because each user exclusively utilizes channel bandwidth and server computing resources in the SL scheme.

2) Performance of User Grouping Algorithm: 1) *Prediction Performance of GPR:* In Fig. 8(b), we conducted simulations to validate the proposed estimation methods for the function of training rounds. We collect 25 datasets from the GHSL scheme to fit the training rounds function. The result illustrates that the red dots represent actual training round values for given numbers of users and groups, with a fitted surface passing through these points. Given optimal kernel parameters, the comparison between predicted and actual values shows that the estimation error rate is less than 5%.

Kernel parameter settings influence predictions made via Gaussian process regression, prompting us to assess the impact of various kernel settings on mean error values. Specifically, Gaussian process regression aims to maximize the likelihood function to identify optimal kernel parameters. Therefore, the boundaries of these parameters affect the fitting outcomes. We explored how mean error varies with changes in the RBF kernel by setting different boundary ranges for the constant kernel. In Fig. 8(a), the results indicate that defining constant kernel (σ) within the range of 0.001 to 10 significantly reduces prediction errors, and the mean error rate is below 5% when the RBF parameter is 10.

2) *Delay Performance of the Grouping Algorithm:* A comparison of the proposed coalition game-based grouping algorithm with two other benchmarks is presented in Fig. 8(c). The first benchmark is a heuristic user grouping algorithm, where users are grouped based on their computing capabilities, grouping those with similar capacities together. The second benchmark is that users are randomly divided. In the simulations, the computing capabilities of users and the SNR of received signals are set to follow a normal distribution, with mean values of 2 GHz and 15 dB, and standard deviations of 0.5 GHz and 5 dB respectively. The simulation results demonstrate that the proposed grouping algorithm significantly reduces the training delay per round compared to the benchmarks. Specifically, it reduces delay by an average of 45% compared to the heuristic algorithm and 25% compared to the random algorithm. This is because the proposed user grouping algorithm mitigates the straggler effect between the groups under hierarchical system resources.

Fig. 8(d) shows the training delay performance of the proposed scheme and user grouping algorithm. It can be seen that the training delay with the proposed scheme is about 1,750 seconds, which is 45% lower than the 2,542 seconds by combining the GHSL scheme with the heuristic grouping algorithm, and 24% lower than the 2,169 seconds with the random grouping algorithm. These results indicate that the combination of the two designs effectively reduces the overall training delay.

IX. Conclusion

In this paper, we have proposed a novel GHSL scheme to expedite SL in edge networks. The proposed scheme partitions the users into multiple groups and trains users in each group sequentially while groups are trained parallelly. This can expedite the training process. In addition, we have conducted a convergence analysis of the GHSL scheme, illustrating that the user grouping decision impacts the convergence rate. Moreover, we have designed a two-stage algorithm to identify the group number and then make the optimal user grouping decision, thereby minimizing the overall training delay. Simulation results have demonstrated that the proposed scheme outperforms the benchmarks in terms of training delay. The proposed scheme is suitable for edge networks with limited spectrum resources and mobile devices with constrained computing resources due to low communication overhead and user-side computational workload. For the future work, we will investigate the impact of user mobility on the performance of the GHSL scheme.

References

[1] Z. Yang, M. Chen, K. K. Wong, H. V. Poor, and S. Cui, "Federated learning for 6G: Applications, challenges, and opportunities," *Engineering*, vol. 8, pp. 33–41, 2022.

[2] I. Masi, Y. Wu, T. Hassner, and P. Natarajan, "Deep face recognition: A survey," in *Proc. 31st SIBGRAPI Conf. Graph. Patterns Images*, 2018, pp. 471–478.
[3] C. Y. Chandan, K. Sahu, and R. Rai, "Artificial intelligence (AI) in augmented reality (AR)-assisted manufacturing applications: A review," *Int. J. Prod. Res.*, vol. 59, no. 16, pp. 4903–4959, 2021.
[4] Z. Feng, X. Chen, Q. Wu, W. Wu, X. Zhang, and Q. Huang, "FedDD: Toward communication-efficient federated learning with differential parameter dropout," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5366–5384, May 2024, doi: 10.1109/TMC.2023.3311188.
[5] X. Shen et al., "AI-assisted network-slicing based next-generation wireless networks," *IEEE Open J. Veh. Technol.*, vol. 1, pp. 45–66, 2020.
[6] Q. Pu et al., "Low latency geo-distributed data analytics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 421–434, 2015.
[7] S. Chen, Y. Xu, H. Xu, Z. Jiang, and C. Qiao, "Decentralized federated learning with intermediate results in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 341–358, Jan. 2024.
[8] X. Cao, Z. Lyu, G. Zhu, J. Xu, L. Xu, and S. Cui, "An overview on over-the-air federated edge learning," *IEEE Wireless Commun.*, vol. 31, no. 3, pp. 202–210, Jun. 2024.
[9] X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic network virtualization and pervasive network intelligence for 6G," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 1, pp. 1–30, First Quarter, 2022.
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[12] K. Pfeiffer, M. Rapp, R. Khalili, and J. Henkel, "Federated learning for computationally-constrained heterogeneous devices: A survey," *ACM Comput. Surv.*, vol. 55, no. 14, pp. 300–360, 2023.
[13] D. Yang et al., "DetFed: Dynamic resource scheduling for deterministic federated learning over time-sensitive networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5162–5178, May 2024.
[14] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "SplitFed: When federated learning meets split learning," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 8485–8493.
[15] W. Wu et al., "Split learning over wireless networks: Parallel design and resource management," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1051–1066, Apr. 2023.
[16] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, 2018.
[17] Y. Gao et al., "End-to-end evaluation of federated learning and split learning for Internet of Things," 2020, *arXiv: 2003.13376*.
[18] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," 2019, *arXiv: 1909.09145*.
[19] X. Wu, X. Yao, and C.-L. Wang, "FedSCR: Structure-based communication reduction for federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1565–1577, Jul. 2021.
[20] J. Guo, J. Wu, A. Liu, and N. N. Xiong, "LightFed: An efficient and secure federated edge learning system on model splitting," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2701–2713, Nov. 2022.
[21] Y. Koda et al., "Communication-efficient multimodal split learning for mmWave received power prediction," *IEEE Commun. Lett.*, vol. 24, no. 6, pp. 1284–1288, Jun. 2020.
[22] J. Liu et al., "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 674–690, Feb. 2023.
[23] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *Proc. IEEE Int. Conf. Commun.*, 2020, pp. 1–6.
[24] X. Chen, J. Li, and C. Chakrabarti, "Communication and computation reduction for split learning using asynchronous training," in *Proc. IEEE Workshop Signal Process. Syst.*, 2021, pp. 76–81.
[25] Y. Jiang et al., "Model pruning enables efficient federated learning on edge devices," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 12, pp. 10374–10386, Dec. 2023.
[26] N. Bouacida, J. Hou, H. Zang, and X. Liu, "Adaptive federated dropout: Improving communication efficiency and generalization for federated learning," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–9.
[27] N. D. Pham, A. Abuadbba, Y. Gao, K. T. Phan, and N. Chilamkurti, "Binarizing split learning for data privacy enhancement and computation reduction," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 3088–3100, 2023.
[28] E. Samikwa, A. D. Maio, and T. Braun, "DISNET: Distributed micro-split deep learning in heterogeneous dynamic IoT," *IEEE Internet Things J.*, vol. 11, no. 4, pp. 6199–6216, Feb. 2024.
[29] W. Wu et al., "AI-native network slicing for 6G networks," *IEEE Wireless Commun.*, vol. 29, no. 1, pp. 96–103, Feb. 2022.
[30] X. Liu, Y. Deng, and T. Mahmoodi, "Wireless distributed learning: A new hybrid split and federated learning approach," *IEEE Trans. Wireless Commun.*, vol. 22, no. 4, pp. 2650–2665, Apr. 2023.
[31] B. Yin, Z. Chen, and M. Tao, "Predictive GAN-powered multi-objective optimization for hybrid federated split learning," *IEEE Trans. Commun.*, vol. 71, no. 8, pp. 4544–4560, Aug. 2023.
[32] C. Xu, J. Li, Y. Liu, Y. Ling, and M. Wen, "Accelerating split federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 23, no. 6, pp. 5587—599, Jun. 2024, doi: 10.1109/TWC.2023.3327372.
[33] S. Zhang et al., "Cluster-HSFL: A cluster-based hybrid split and federated learning," in *Proc. IEEE/CIC Int. Conf. Commun. China*, 2023, pp. 1–2.
[34] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[35] C. Feng, H. H. Yang, D. Hu, Z. Zhao, T. Q. S. Quek, and G. Min, "Mobility-aware cluster federated learning in hierarchical wireless networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 10, pp. 8441–8458, Oct. 2022.
[36] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*, vol. 2. Cambridge, MA, USA: MIT Press, 2006.
[37] M. Hu et al., "AutoFL: A Bayesian game approach for autonomous client participation in federated edge learning," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 194–208, Jan. 2024.
[38] J. Wang, S. Wang, R.-R. Chen, and M. Ji, "Demystifying why local aggregation helps: Convergence analysis of hierarchical SGD," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 8548–8556.
[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

Songge Zhang (Student Member, IEEE) received the BE degree in electronic information engineering from Zhengzhou University, Zhengzhou, China, in 2019, and the MS degree in traffic information engineering and control from Beihang University, Beijing, China, in 2022. He is currently working toward the PhD degree in communication and information systems with Peking University, Shenzhen, China. His research interests include edge intelligence, network for AI, and network resource management.

Wen Wu (Senior Member, IEEE) received the BE degree in information engineering from the South China University of Technology, Guangzhou, China, in 2012, the ME degree in electrical engineering from the University of Science and Technology of China, Hefei, China, in 2015, and the PhD degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2019. He worked as a post-doctoral fellow with the Department of Electrical and Computer Engineering, University of Waterloo. He is currently an associate researcher with the Frontier Research Center, Pengcheng Laboratory, Shenzhen, China. His research interests include 6G networks, network intelligence, and network virtualization. He serves as track co-chairs for IEEE VTC, and Workshop co-chairs for IEEE INFOCOM, IEEE Globecom, and IEEE ICCC. He serves on the editorial board for the *IEEE Networking Letter*, *Hindawi Wireless Communications and Mobile Computing*, *China Communications*, and *Springer Peer-to-Peer Networking and Applications*. He has published more than 100 refereed IEEE journal and conference papers and six books/book chapters. He received the World Top 2% Scientist Award 2023-2024, USENIX Security Distinguished Paper Award, IEEE HITC Award for Excellence (Early Career Researcher), and IEEE CIC/ICCC Best Paper Award.

Lingyang Song (Fellow, IEEE) received the PhD degree from the University of York, U.K., in 2007. He worked as a research fellow with the University of Oslo, Norway, until rejoining Philips Research U.K. in March 2008. In May 2009, he joined the School of Electronics Engineering and Computer Science, Peking University, and is now a Boya distinguished professor. His main research interests include wireless communications, mobile computing, and machine learning. He is the co-author of many awards, including the IEEE Leonard G. Abraham Prize in 2016, IEEE ICC 2014, IEEE ICC 2015, IEEE Globecom 2014, and the best demo award in ACM MobiHoc 2015. He received the National Science Fund for Distinguished Young Scholars in 2017, and the First Prize in the Nature Science Award of the Ministry of Education of China in 2017. He has served as an IEEE ComSoc Distinguished Lecturer (2015–2018), an area editor of the *IEEE Transactions on Vehicular Technology* (2019–), and is currently the director of the IEEE Communications Society Asia Pacific Board (2024–2025). He is a clarivate analytics highly cited researcher.

Xuemin (Sherman) Shen (Fellow, IEEE) received the PhD degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is a University professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on network resource management, wireless network security, Internet of Things, 5G and beyond, and vehicular networks. He is a registered professional engineer of Ontario, Canada, an Engineering Institute of Canada fellow, a Canadian Academy of Engineering fellow, a Royal Society of Canada fellow, a Chinese Academy of Engineering foreign member, and an Engineering Academy of Japan International fellow. He received the "West Lake Friendship Award" from Zhejiang Province in 2023, the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, the R.A. Fessenden Award in 2019 from IEEE, Canada, Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society (ComSoc), and Technical Recognition Award from Wireless Communications Technical Committee (2019) and AHSN Technical Committee (2013). He has also received the Excellent Graduate Supervision Award in 2006 from the University of Waterloo and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He serves/served as the general chair for the 6 G Global Conference'23, and ACM Mobihoc'15, Technical Program Committee chair/co-chair for IEEE Globecom'24, '16 and '07, IEEE Infocom'14, and IEEE VTC'10 Fall. He is the past president of the IEEE ComSoc, the vice president for Technical & Educational Activities, vice president for Publications, member-at-large on the Board of Governors, chair of the Distinguished Lecturer Selection Committee, and member of IEEE Fellow Selection Committee of the ComSoc. He served as the editor-in-chief of the *IEEE Internet of Things Journal*, *IEEE Network*, and *IET Communications*.