ACCELERATING FEDERATED SPLIT LEARNING VIA LOCAL-LOSS-BASED TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated learning (FL) operates based on model exchanges between the server and the clients, and suffers from significant communication as well as client-side computation burden. Emerging split learning (SL) solutions can reduce the client-side computation burden by splitting the model architecture between the server and the clients. However, SL-based ideas still require significant time delay, since each participating client should wait for the backpropagated gradients from the server in order to update its model. Also, the communication burden can still be substantial, depending on various factors like local dataset size and shape of cut layer activations/gradients. In this paper, we propose a new direction to FL/SL based on updating the client/server-side models in parallel, via local-loss-based training specifically geared to split learning. The parallel training of split models substantially *shortens latency* while *obviating server-to-clients communication*. We provide latency analysis that leads to optimal model cut as well as general guidelines for splitting the model. We also provide a theoretical analysis for guaranteeing convergence and understanding interplay among different hyperparameters and system constraints. Extensive experimental results indicate that our scheme has significant communication and latency advantages over existing FL and SL ideas.

1 INTRODUCTION

Federated learning (FL) (McMahan et al., 2017; Konečnỳ et al., 2016b;a; Li et al., 2020) is being regarded as a promising direction for distributed learning, as it enables clients to collaboratively train a global model without directly uploading their data to the server. However, in FL, each client should repeatedly download the entire model from the server, update the model, and upload it back to the server. This training process in FL causes significant computation/communication burdens especially with deep neural networks having large numbers of model parameters. Moreover, when the computing powers and the transmission rates of the clients are low (e.g., mobile/IoT devices), FL requires significant time delay for training the model. These issues can limit the application of FL in practical scenarios aiming to train a *large-scale model* using local data of clients given *low computing powers* and *low transmission rates*.

Split learning (SL) (Gupta & Raskar, 2018; Vepakomma et al., 2018; Singh et al., 2019; Koda et al., 2020; Thapa et al., 2020) is another recent approach for this setup, which can reduce the computation burden at the clients by splitting the model $\mathbf{w}$ into two parts before training begins: the first few layers (client-side model $\mathbf{w}_C$) are allocated to the clients, and the remaining layers (server-side model $\mathbf{w}_S$) are allocated to the server. Since each client only need to train the first few layers of the model, the computational burden at each client is reduced compared to FL.

However, existing SL-based ideas still have two critical issues in terms of latency and communication efficiency. First, existing SL solutions still require significant time delay, since each participating client should wait for the backpropagated gradients from the server in order to update its model. Moreover, the communication burden can still be substantial for transmitting the forward/backward signals via uplink/downlink communications at each global round.

Contributions: In this paper, we propose a fast and communication-computation efficient solution that provides a new direction to federated/split learning, by simultaneously reducing the following three key resources in distributed learning: computation, communication, and latency. The computation burden at the clients is reduced by splitting the full model into the client-side and server-side

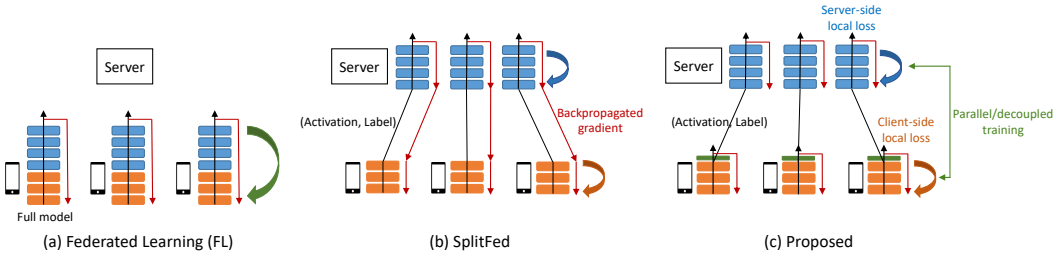


Figure 1: Model update process of FL, SplitFed and our idea. FL suffers from large time delay and communication/computation burdens for exchanging/updating the full model. Although SplitFed can reduce the client-side computation burden, it still requires large time delay since the clients should wait for the backpropagated signals from the server in order to update their model. The communication burden can be also large for transmitting both the forward activations and backward gradients at every global round. The proposed idea enables to save all three resources (computation, communication, latency) simultaneously via model splitting and local-loss-based training highly tailored to split learning. Our approach has significant advantage especially when clients having low computing powers and transmission rates (e.g., mobile/IoT devices) collaborate to train a *large-scale model*.

models, as in split learning. To handle the high communication resource requirement and high latency requirement of current FL and SL approaches, we propose a local-loss-based training method highly tailored to the split learning setup. By introducing two different local loss functions (i.e., the client-side and server-side local losses), the client-side models can be updated without receiving the backpropagated signals from the server, significantly improving communication efficiency and latency. Fig. 1 compares our approach[1] with FL and the state-of-the-art SL idea, termed SplitFed (Thapa et al., 2020). Our main contributions can be summarized as follows:

- We propose a **new federated split learning algorithm** that can simultaneously save the **three key resources** (computation, communication, latency) of current FL/SL systems, via model splitting and local-loss-based training specifically geared to the split learning setup.
- We provide **latency analysis** and provide an optimal solution on splitting the model. We also provide theoretical analysis to **guarantee convergence** of our scheme and to understand interplay among key system constraints and hyperparameters in reaching the convergence.
- We show via experiments that our approach **outperforms existing FL and SL-based ideas** in practical setups where a number of devices having low computing powers and low transmission rates collaborate to train a large-scale model.

2 Backgrounds and Related Works

Consider a system with a single server and N clients having their own local/private data. FL and SL are the recent ideas that aim to train a model in this setup. The goal is generally to find $\mathbf{w}^*$ that minimizes the loss function defined as $F(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N} F_k(\mathbf{w})$. Here, $F_k(\mathbf{w})$ is the loss function at client k defined as $F_k(\mathbf{w}) = \frac{1}{|D_k|}\sum_{x\in D_k} \ell(x;\mathbf{w})$ where D_k is the dataset of client k and $\ell(x;\mathbf{w})$ is the loss computed by the model $\mathbf{w}$ and the data sample x.

Federated learning: In FL (McMahan et al., 2017; Konečnỳ et al., 2016b;a; Li et al., 2020), the above problem is solved via repeated model download at the clients and aggregation at the server. At every global round t, the server randomly selects A_t, a set of K clients participating in FL in this round. Each client $k \in A_t$ downloads the model $\mathbf{w}^t$ from the server and performs local update to obtain $\mathbf{w}_k^{t+1}$. The server aggregates the models from all clients to obtain $\mathbf{w}^{t+1} = \frac{1}{K}\sum_{k\in A_t} \mathbf{w}_k^{t+1}$ and moves on to the next round. This process is repeated until some stopping condition is met, e.g., when the model achieves a desired level of performance. It is shown in (Li et al., 2019) that this algorithm converges to the optimal solution of the above objective function. However, when training a large-scale model, FL causes significant communication burden between the server and the clients for model exchange, and considerable computation burden for training the model at low-powered clients (e.g., mobile devices), resulting in a large time delay for training.

Split learning: SL (Gupta & Raskar, 2018; Vepakomma et al., 2018; Singh et al., 2019; Koda et al., 2020; Thapa et al., 2020) is another direction to train the model in this setup while reducing the computation burden at the clients compared to FL. The basic idea of SL approaches is to split the

[1]Any aggregation rule (e.g., FedAvg) can be utilized for the model aggregation process of FL, SplitFed, and our approach.

model $\mathbf{w}$ into two modules as $\mathbf{w} = [\mathbf{w}_C, \mathbf{w}_S]$. The first few layers $\mathbf{w}_C$ correspond to the client-side model, and the remaining layers $\mathbf{w}_S$ correspond to the server-side model. Among existing SL ideas, SplitFed (Thapa et al., 2020) achieves the state-of-the-art performance by parallelizing SL; SplitFed enables to train the model in a split learning setup with multiple clients updating their models in parallel as in FL. At each global round t, the server randomly selects a set A_t that consists of K participating clients in the current round. Each client $k \in A_t$ downloads the client-side model $\mathbf{w}_C^t$ from the server, performs forward propagation with its local data, and sends the output and the corresponding labels to the server. The server proceeds forward propagation, computes the loss, and performs backpropagation to update the server-side models in parallel, as in Fig. 1(b). Now the server transmits the corresponding backpropagated signal to each client. Each client k can update the model by proceeding backpropagation to obtain $\mathbf{w}_{C,k}^t$. Finally, the server aggregates the updated models from all clients to obtain $\mathbf{w}_C^{t+1} = \frac{1}{K}\sum_{k\in A_t} \mathbf{w}_{C,k}^{t+1}$. The server-side models are also aggregated to obtain $\mathbf{w}_S^{t+1}$. It can be easily seen that this SplitFed algorithm also converges to the optimal solution $\mathbf{w}^*$ of the above objective function, but with a less computation burden at the clients compared to FL. However, although SplitFed can reduce the client-side computation burden compared to FL, existing SL-based ideas still have issues in terms of latency and communication efficiency; all participating clients should receive the backpropagated signals from the server in order to update their models, which require significant time delay and communication resources at each global round.

Local-loss-based training: Local-loss-based training (Nøkland & Eidnes, 2019; Belilovsky et al., 2019; 2020; Laskin et al., 2020; Xiong et al., 2020) represents schemes that aim to train a model using local error signals (local loss functions) instead of using the conventional global loss function. Auxiliary networks are utilized to compute the local error signals. By utilizing the local losses, layer-wise training (Nøkland & Eidnes, 2019; Belilovsky et al., 2019) or module-wise training (Belilovsky et al., 2020; Laskin et al., 2020; Xiong et al., 2020) is possible without receiving the backpropagated signals from the previous layer or module. While existing works on local-loss-based training consider a centralized setup (i.e., central node having the entire dataset and the model), in this paper, we specifically focus on a distributed setup (i.e., data distributed across the clients and model splitted between the client/server) and propose a local-loss-based training method highly tailored to split learning. Theoretical and experimental results indicate that our new algorithm can provide a new direction to federated/split learning via local-loss-based training.

Variants of SL: Instead of parallelizing the update process at the server (as in Fig. 1(b)), sequentially updating the server-side model is also possible (Thapa et al., 2020); given a single server-side model, it can be updated asynchronously via forward/backward propagation using each client's output. Although this sequential update process can speed up training in the beginning, the convergence to the optimal solution is *not guaranteed* and often achieves a lower accuracy as shown later in Section 5. Note that SplitFed with this sequential update process also has the same issues on communication efficiency and latency described above. Finally, we note that the authors of He et al. (2020) also defines local loss functions in a SL setup but take a different approach. In He et al. (2020), the models of the clients and the server are updated in an alternative way. In contrast, our work specifically focus on training the client-side and the server-side models in parallel, which is motivated by the works on local-loss-based training (Belilovsky et al., 2020) that trains each module of the full model in parallel in a centralized setup. In other words, in our scheme, the client does not wait until the server-side update is finished, and the server does not wait until the client-side model update is finished. Since each client obtains different personalized models, the goal is different from our approach aiming to obtain a single global model and thus is not directly comparable. Under this different approach/goal, the latency analysis and convergence analysis are also unique to our work.

3 Proposed Algorithm

In this section, we describe our algorithm that simultaneously handles communication/computation burden and latency issues of current FL and SL approaches. To reduce the computation burden at the clients, we first split the model $\mathbf{w}$ into the client-side model $\mathbf{w}_C$ and the server-side model $\mathbf{w}_S$, as in SL. Our goal is to obtain the optimal model $\mathbf{w}^* = [\mathbf{w}_C^*, \mathbf{w}_S^*]$ after training.

3.1 Local Loss Functions

In order to improve latency and communication efficiency, instead of considering the conventional loss function that is computed at the output of the model $\mathbf{w}$, our idea is to consider two different *local loss functions* specifically geared to the split learning setup.

Client-side local loss function: We first describe the client-side local loss function. We introduce an auxiliary network $\mathbf{a}_C$ to make a prediction at the client-side and then compute the local loss. Here, the auxiliary network $\mathbf{a}_C$ is the extra layers connected to the client-side model $\mathbf{w}_C$; the output of $\mathbf{w}_C$ becomes the input of $\mathbf{a}_C$. Both convolutional neural networks or multilayer perceptrons (MLP) can be utilized for the auxiliary network $\mathbf{a}_C$, which is our design choice. In this work, we adopt a MLP for $\mathbf{a}_C$ as in (Belilovsky et al., 2020; Laskin et al., 2020) to match the dimension between the output of $\mathbf{a}_C$ and the target label. This auxiliary network enables to update the client-side model independently of the layers at the server-side. When training is finished, the auxiliary network is discarded and prediction is made with the full model $\mathbf{w} = [\mathbf{w}_C, \mathbf{w}_S]$. We show later in Section 5 that only a very small size auxiliary network is sufficient (0.1% of the entire model size $|\mathbf{w}|$) to achieve the state-of-the-art performance. Our goal is to find $\mathbf{w}_C^*$ and $\mathbf{a}_C^*$ that minimizes the client-side loss function $F_C(\cdot,\cdot)$ which is the average of local loss functions of all clients:

$$\min_{\mathbf{w}_C,\mathbf{a}_C} F_C(\mathbf{w}_C,\mathbf{a}_C) = \min_{\mathbf{w}_C,\mathbf{a}_C} \frac{1}{N}\sum_{k=1}^{N} F_{C,k}(\mathbf{w}_C,\mathbf{a}_C), \tag{1}$$

where $F_{C,k}(\cdot,\cdot)$ is the local loss function of the client-side model at client k, defined as $F_{C,k}(\mathbf{w}_C,\mathbf{a}_C) = \frac{1}{|D_k|}\sum_{x\in D_k} \ell(x;\mathbf{w}_C,\mathbf{a}_C)$. Here, $\ell(x;\mathbf{w}_C,\mathbf{a}_C)$ is the loss computed based on input x, client-side model $\mathbf{w}_C$ and the auxiliary network $\mathbf{a}_C$. Any loss function is applicable to our work (e.g., cross-entropy loss).

Server-side local loss function: The local loss function of the server-side model $\mathbf{w}_S$ is defined based on the optimal client-side model $\mathbf{w}_C^*$ defined in (1). We would like to find $\mathbf{w}_S^*$ that minimizes the server-side loss function $F_S(\cdot)$:

$$\min_{\mathbf{w}_S} F_S(\mathbf{w}_S) = \min_{\mathbf{w}_S} \frac{1}{N}\sum_{k=1}^{N} F_{S,k}(\mathbf{w}_S,\mathbf{w}_C^*), \tag{2}$$

where $F_{S,k}(\cdot,\cdot)$ is the local loss function of the server-side model corresponding to client k, defined as $F_{S,k}(\mathbf{w}_S,\mathbf{w}_C^*) = \frac{1}{|D_k|}\sum_{x\in D_k} \ell(g_{\mathbf{w}_C^*}(x);\mathbf{w}_S)$. Here, note that the input of $\ell(g_{\mathbf{w}_C^*}(x);\mathbf{w}_S)$ is $g_{\mathbf{w}_C^*}(x)$, which is defined as the output of the model $\mathbf{w}_C^*$ given the input x. We also note that the auxiliary network is not necessary at the server-side; the loss can be directly computed without the auxiliary network after finishing forward propagation of the server-side model.

3.2 Algorithm Description

Now we describe our algorithm to solve the above problem. Starting from the initial model $\mathbf{w}^0 = [\mathbf{w}_C^0, \mathbf{w}_S^0]$, we obtain $\mathbf{w}^T = [\mathbf{w}_C^T, \mathbf{w}_S^T]$ after T global rounds. As in (Thapa et al., 2020), we consider two different servers, the main server and the fed server. The main server updates $\mathbf{w}_S$ while the fed server only aggregates the models sent from the clients via FedAvg. The details will be clarified soon. In the beginning of each global round t, the server randomly selects the participating group A_t with K clients. Now we have the following four steps with steps 3 and 4 working in parallel.

Step 1 (Model download): At a specific global round t, each client $k \in A_t$ downloads $\mathbf{w}_C^t$ and $\mathbf{a}_C^t$ from the fed server and lets $\mathbf{w}_{C,k}^t = \mathbf{w}_C^t$, $\mathbf{a}_{C,k}^t = \mathbf{a}_C^t$.

Step 2 (Forward propagation and upload): Based on the downloaded model $\mathbf{w}_{C,k}^t$, each client k performs forward propagation for all data samples $x \in \tilde{D}_k$ in a specific mini-batch $\tilde{D}_k \subset D_k$. Specifically, client k obtains $g_{\mathbf{w}_{C,k}^t}(x)$ for all $x \in \tilde{D}_k$, which is the output of the model $\mathbf{w}_{C,k}^t$ given an input data x. Then each client k uploads $g_{\mathbf{w}_{C,k}^t}(x)$ to the main server for all $x \in \tilde{D}_k$.

Step 3 (Client-side model update and aggregation): Now based on the local loss function, each client k updates its model $\mathbf{w}_{C,k}^t$ and the auxiliary network $\mathbf{a}_{C,k}^t$ as $\mathbf{w}_{C,k}^{t+1} = \mathbf{w}_{C,k}^t - \eta_t \tilde{\nabla}_{\mathbf{w}} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t)$ and $\mathbf{a}_{C,k}^{t+1} = \mathbf{a}_{C,k}^t - \eta_t \tilde{\nabla}_{\mathbf{a}} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t)$, where η_t is the learning rate at round t and $\tilde{\nabla} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t)$ is the derivative for a specific mini-batch, i.e., $\tilde{\nabla} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t) = \frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k} \nabla \ell(x;\mathbf{w}_{C,k}^t,\mathbf{a}_{C,k}^t)$. After the model update process, the fed server aggregates the client-side models as $\mathbf{w}_C^{t+1} = \frac{1}{K}\sum_{k\in A_t} \mathbf{w}_{C,k}^{t+1}$. The auxiliary networks are also aggregated as $\mathbf{a}_C^{t+1} = \frac{1}{K}\sum_{k\in A_t} \mathbf{a}_{C,k}^{t+1}$.

Step 4 (Server-side model update and aggregation, working in parallel with step 3): While the client-side models are being updated using the local errors, in parallel with step 3, the main server also updates the server-side model. Based on $g^t_{\mathbf{w}_{C,k}}(x)$ received from each client k in step 2, the main server performs model update according to $\mathbf{w}^{t+1}_{S,k} = \mathbf{w}^t_S - \eta_t \tilde{\nabla} F_{S,k}(\mathbf{w}^t_S, \mathbf{w}^t_C)$ in parallel for all $k \in A_t$, where $\tilde{\nabla} F_{S,k}(\mathbf{w}^t_S, \mathbf{w}^t_C) = \frac{1}{|\tilde{D}_k|} \sum_{x \in \tilde{D}_k} \nabla \ell(g_{\mathbf{w}^t_{C,k}}(x); \mathbf{w}^t_S)$. Now the server-side models are aggregated as $\mathbf{w}^{t+1}_S = \frac{1}{K} \sum_{k \in A_t} \mathbf{w}^{t+1}_{S,k}$. We note that the model update process in steps 3 and 4 can be repeated for multiple mini-batches to obtain $\mathbf{w}^{t+1}_{C,k}$ and $\mathbf{w}^{t+1}_{S,k}$ before the aggregation process.

After repeating the overall procedure for T global rounds, we discard the auxiliary networks and obtain the final global model $\mathbf{w}^T = [\mathbf{w}^T_C, \mathbf{w}^T_S]$. This full global model is utilized at inference stage to make predictions. The detailed procedure of our method is summarized in Algorithm 1 in Appendix.

3.3 Analysis: Computation, Communication and Latency

Notations and assumptions: Let $|\mathbf{w}|$ be the number of model parameters of $\mathbf{w}$, and α be the fraction of model parameters in $\mathbf{w}_C$: we have $|\mathbf{w}_C| = \alpha|\mathbf{w}|$ and $|\mathbf{w}_S| = (1-\alpha)|\mathbf{w}|$. We note that the auxiliary network is our design choice which can be made to have a significantly small number of parameters, i.e., $|\mathbf{a}| \ll |\mathbf{w}|$. Hence, we neglect the effect of the auxiliary network for latency analysis. We show later in Section 5 that the state-of-the-art performance can be achieved with negligible size of auxiliary network (0.1% of the entire model size $|\mathbf{w}|$). For analysis, we assume that all layers have the same size of q. Let P_C and P_S be the computing powers at the client and the server, respectively, where $P_C \ll P_S$ holds. We assume that the computation time for model update is proportional to the data size and the model size; for a given local dataset size $|D|$, model size $|\mathbf{w}|$ and computing power P, the required time for updating the model for one epoch is assumed to be $\frac{|D||\mathbf{w}|}{P}$. Here, we assume that the required time for the forward propagation is $\frac{\beta|D||\mathbf{w}|}{P}$ while the required time for the backpropagation is $\frac{(1-\beta)|D||\mathbf{w}|}{P}$. We consider a full-batch update for analysis. Finally, given a single client, we let both the uplink transmission rate from the client to server and the downlink transmission rate from the server to client as R. When K clients are communicating with the server simultaneously, the transmission rate of each client reduces to $\frac{R}{K}$.

Analysis for our scheme: In the beginning of each global round of our scheme, K clients simultaneously download $\mathbf{w}_C$ from the fed server, which requires latency of $\frac{\alpha|\mathbf{w}|K}{R}$. The forward propagation and transmitting the output to the main server requires additional latency of $\frac{\alpha\beta|D||\mathbf{w}|}{P_C} + \frac{q|D|K}{R}$. Now in parallel, each client updates its model and sends it to the fed server for aggregation, while the main server updates the server-side model. The latency is determined by the maximum value of these two, which leads to additional delay of $\max\left(\frac{\alpha|\mathbf{w}|K}{R} + \frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}, \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}\right)$.

Comparison with FL and SplitFed: Table 1 compares the key three metrics of different methods. SplitFed and our method can have significantly smaller computation load at the clients compared to FL. Since the proposed idea does not require downlink communication between the server and the clients for transmitting the gradients, our scheme requires smaller communication load compared to SplitFed. Here, the gap increases with more clients participating in each round (i.e., with a larger K). Regarding the latency, we first highlight that regardless of the system parameters, our scheme has lower latency compared to SplitFed. The gain increases with a larger K. Now we have the following question: in which conditions are the SL-based methods (including our scheme) better than FL? With some manipulations, it can be easily seen that SplitFed has lower latency compared to FL if and only if $P_S > (\frac{2}{R|D|} + \frac{1}{P_C K})^{-1}$, $|\mathbf{w}| > \frac{2q|D|}{R} / (\frac{2K}{R} + \frac{|D|}{P_C} - \frac{|D|K}{P_S})$ and $\alpha < 1 - \frac{\alpha_1}{\alpha_2}$, where $\alpha_1 = \frac{2q|D|}{R}$ and $\alpha_2 = (\frac{2K}{R} + \frac{|D|}{P_C} - \frac{|D|K}{P_S})|\mathbf{w}|$. Since our approach is always faster than SplitFed regardless of the system parameters, our scheme is also faster than FL under these three conditions. We have the following two observations from these conditions. First is the effect of model size $|\mathbf{w}|$: it can be seen from the second condition that the model size $|\mathbf{w}|$ should be larger than a certain threshold. It can be also seen that the third condition can be easily satisfied with a larger $|\mathbf{w}|$. Our second observation is the impact of the client-side computing power P_C and the transmission rate R: it can be seen from all three conditions that as P_C and R become smaller, SplitFed and our scheme have better latency than FL even with a smaller computing power at the server (i.e., smaller P_S) and with a smaller model size (i.e., smaller $|\mathbf{w}|$), regardless of α. These two observations confirm that SL-based

Table 1: Computation load (per client), total communication load, and latency required for one global round.

Methods	Computation	Communication	Latency (total cost)
FL	$\|D\|\|\mathbf{w}\|$	$2\|\mathbf{w}\|K$	$\frac{2\|\mathbf{w}\|K}{R} + \frac{\|D\|\|\mathbf{w}\|}{P_C}$
SplitFed	$\alpha\|D\|\|\mathbf{w}\|$	$(2q\|D\| + 2\alpha\|\mathbf{w}\|)K$	$\frac{(2q\|D\|+2\alpha\|\mathbf{w}\|)K}{R} + \frac{\alpha\|D\|\|\mathbf{w}\|}{P_C} + \frac{(1-\alpha)\|D\|\|\mathbf{w}\|K}{P_S}$
Ours	$\alpha\|D\|\|\mathbf{w}\|$	$(q\|D\| + 2\alpha\|\mathbf{w}\|)K$	$\frac{(q\|D\|+\alpha\|\mathbf{w}\|)K}{R} + \frac{\alpha\beta\|D\|\|\mathbf{w}\|}{P_C} + \max\left(\frac{\alpha\|\mathbf{w}\|K}{R} + \frac{\alpha(1-\beta)\|D\|\|\mathbf{w}\|}{P_C}, \frac{(1-\alpha)\|D\|\|\mathbf{w}\|K}{P_S}\right)$

methods (including our scheme) have advantage over FL when clients having *low computing powers* and *low transmission rates* collaborate to train a *large-scale model*.

Optimal splitting: Now we have another important question: how should we split the model for our scheme? In other words, what is the optimal α? The following theorem provides a guideline on splitting the model to minimize latency of our scheme, where the proof is given in Appendix.

Theorem 1 *If $P_S \leq (\frac{1}{R|D|} + \frac{\beta}{P_C K})^{-1}$, optimal α that minimizes the latency of our scheme is $\alpha^* = \frac{1}{P_S\left(\frac{1}{R|D|} + \frac{1-\beta}{P_C K}\right)+1}$. Otherwise, the overall latency is an increasing function of α.*

When the computing power of the main server P_S is smaller than $(\frac{1}{R|D|} + \frac{\beta}{P_C K})^{-1}$, the optimal α decreases with decreasing client-side computing power P_C, since larger latency is required for updating the client-side model with a smaller P_C. Moreover, α^* decreases with decreasing transmission rate R, since exchanging the model parameters between the server and the clients requires more latency with a smaller R. If P_S is larger than the threshold $(\frac{1}{R|D|} + \frac{\beta}{P_C K})^{-1}$, it is beneficial to assign as many layers as possible to the server in order to reduce latency.

Here, we note that the schemes relying on model splitting (including SplitFed and our scheme) require transmissions of activations (at the cut-layer) and the corresponding labels to the server. This may lead to a privacy issue due to data leakage. A detailed discussion regarding the privacy issue is described in Appendix.

4 Convergence Analysis

In this section, we provide the convergence behavior of our scheme on non-convex loss functions with the following standard assumptions in FL (Li et al., 2019; Reisizadeh et al., 2020) and local-loss-based training (Belilovsky et al., 2020).

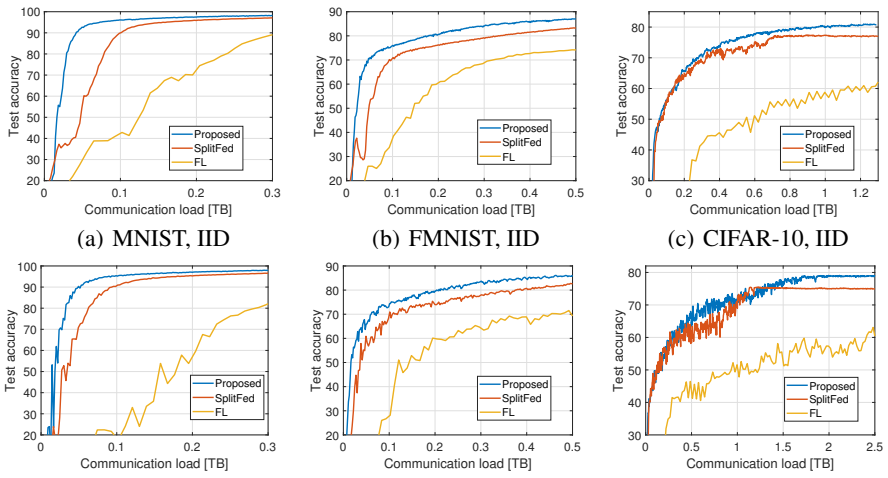
Assumption 1 *The client-side and server-side loss functions are L-smooth, i.e., $\|\nabla F_C(\mathbf{w}) - \nabla F_C(\mathbf{v})\| \leq L\|\mathbf{w} - \mathbf{v}\|$, $\|\nabla F_S(\mathbf{w}) - \nabla F_S(\mathbf{v})\| \leq L\|\mathbf{w} - \mathbf{v}\|$ hold for all $\mathbf{w}$ and $\mathbf{v}$.*

Assumption 2 *The squared norm of the stochastic gradient is upper bounded, i.e., $\|\nabla \ell(x; \mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t)\|^2 \leq G_1$ for all $k = 1, 2, \ldots, K$ and $t = 0, 1, \ldots, T-1$. Similarly, considering the server-side loss, we have $\|\nabla \ell(g_{\mathbf{w}_{C,k}^t}(x); \mathbf{w}_{S,k}^t)\|^2 \leq G_2$ for all $k = 1, 2, \ldots, K$ and $t = 0, 1, \ldots, T-1$.*

Assumption 3 *The learning rates satisfy $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$.*

The smoothness assumption (Assumption 1) is a standard assumption that holds for linear/logistic regression models and neural networks with sigmoid activations. Assumption 2 is equivalent to the assumption that the expected squared norm of gradient is uniformly bounded, which is again commonly adopted in distributed/federated learning (Li et al., 2019; Yu et al., 2019; Stich et al., 2018; Stich, 2019) and local-loss-based training (Belilovsky et al., 2020). The conditions in Assumption 3 (Robbins & Monro, 1951) can be easily satisfied by setting a diminishing learning rate as $\eta_t = \frac{\eta_0}{1+t}$.

In each global round t, the output distribution of a specific client-side model after forward propagation (which is the input distribution of the server-side model) is determined by $\mathbf{w}_{C,k}^t$ and D_k. Let $z_{C,k}^t = g_{\mathbf{w}_{C,k}^t}(x)$ be the output of the k-th client-side model at global round t, following the probability distribution of $p_{C,k}^t(z)$. Here $p_{C,k}^t(z)$ is time-varying, and we let $p_{C,k}^*(z)$ be the output distribution of the k-th client-side model with $\mathbf{w}_C^*$ and D_k. We also define the distance between these two distributions as $d_{C,k}^t = \int \|p_{C,k}^t(z) - p_{C,k}^*(z)\| dz$. Now we provide our main theorem which shows the convergence behaviors of the client/server-side models. The proof is in Appendix.



(a) MNIST, IID (b) FMNIST, IID (c) CIFAR-10, IID

(d) MNIST, non-IID (e) FMNIST, non-IID (f) CIFAR-10, non-IID

Figure 2: Test accuracy versus communication load.

Theorem 2 *Suppose Assumptions 1 and 2 hold. Let* $\Gamma_T = \sum_{t=0}^{T-1} \eta_t$. *After running Algorithm 1, the client-side model converges as*

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \mathbb{E}[||\nabla F_C(\mathbf{w}_C^t, \mathbf{a}_C^t)||^2] \leq \frac{4(F_C(\mathbf{w}_C^0, \mathbf{a}_C^0) - F_C(\mathbf{w}_C^*, \mathbf{a}_C^*))}{3\Gamma_T} + \frac{G_1 L}{2}\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t^2, \quad (3)$$

Moreover, the server-side model converges as

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \mathbb{E}[||\nabla F_S(\mathbf{w}_S^t)||^2] \leq \frac{4(F_S(\mathbf{w}_S^0) - F_S(\mathbf{w}_S^*))}{3\Gamma_T} + G_2\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\left(\eta_t \frac{1}{N}\sum_{k=1}^{N} d_{C,k}^t + \frac{L}{2}\eta_t^2\right). \quad (4)$$

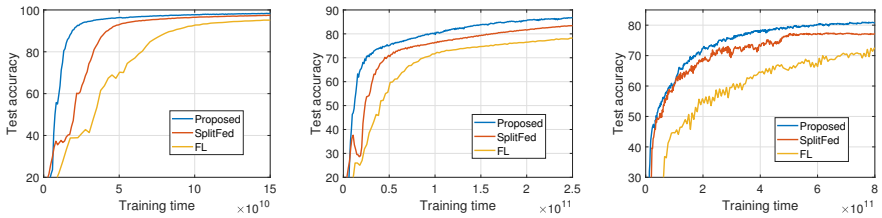
Consider a diminishing step sizes $\eta_t = \frac{\eta_0}{1+t}$ which satisfy the conditions in Assumption 3. Then, as in the results of (Belilovsky et al., 2020), our algorithms converges to a stationary point: it can be seen from (3) that the right-hand side converges to zero as T grows. Regarding the server-side model, it can be seen from (4) that the sequence of expected gradient norm $\mathbb{E}[||\nabla F_S(\mathbf{w}_S^t)||^2]$ accumulates around 0 as $\inf_{t \leq T-1} \mathbb{E}[||\nabla F_S(\mathbf{w}_S^t)||^2] \leq \mathcal{O}(\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \frac{1}{N}\sum_{k=1}^{N} d_{C,k}^t)$. This is in the same form of the result in (Belilovsky et al., 2020) which considers the local-loss-based training method in a centralized setup. The difference here is that our rate $\mathcal{O}(\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \frac{1}{N}\sum_{k=1}^{N} d_{C,k}^t)$ is expressed as the average of distance $d_{C,k}^t$ of all clients, since we consider a distributed setup with multiple clients.

Since there is only a single global loss function in existing FL and SplitFed frameworks (and thus backpropagation is not decoupled), we note that the convergence behaviors of previous methods follow equation (3). Compared to the existing approaches, in our work, two different *local loss functions* are considered with decoupled/parallel training: the input distribution of the server-side model is time-varying, leading to the result in equation (4) for the server-side model.

5 Experiments

We validate our algorithm on MNIST (LeCun et al., 1998), FMNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky et al., 2009) datasets. MNIST and FMNIST are split into $60,000$ training samples and $10,000$ test samples, while CIFAR-10 is split into $50,000$ training samples and $10,000$ test samples. For MNIST and FMNIST, we utilized a CNN having 5 convolutional layers and 3 fully connected (FC) layers as in AlexNet. The number of model parameters is $|\mathbf{w}| = 3,868,170$. For CIFAR-10, we utilized VGG-11 with $|\mathbf{w}| = 9,231,114$.

Data distribution: We distribute the training set of each dataset to the clients for training, and utilized the original test set of each dataset to evaluate the performance of the global model. We consider a system with $N = 1000$ clients. Hence, each client has 60 data samples for MNIST and FMNIST, and 50 data samples for CIFAR-10. We consider two different data distribution setups, IID and non-IID. In an IID setup, data samples from each class is equally distributed across all $N = 1000$



(a) MNIST (b) FMNIST (c) CIFAR-10

Figure 3: Test accuracy versus training time in an IID setup.

Table 2: Performance of different schemes at a specific time in Fig. 3.

	MNIST		FMNIST		CIFAR-10	
Methods	IID	Non-IID	IID	Non-IID	IID	Non-IID
FL	92.80%	89.02%	78.21%	75.77%	72.44%	62.84%
SplitFed	96.47%	95.47%	83.42%	82.44%	77.06%	75.02%
Proposed	**97.73%**	**97.01%**	**86.70%**	**85.74%**	**80.72%**	**78.91%**

clients in the system. Hence, each client has all 10 classes in its local dataset. In a non-IID setup, similar to the data distribution method in (McMahan et al., 2017), the training set is first divided into 5000 shards (12 data samples in each shard for MNIST/FMNIST, and 10 data samples in each shard for CIFAR-10). Then we allocate 5 shards to each client to model the non-IID scenario.

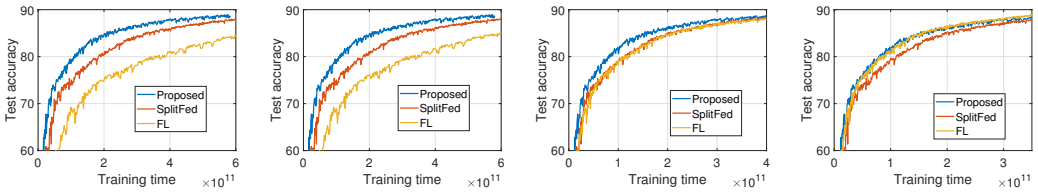
Baselines: We compare our result with FL and SplitFed (Thapa et al., 2020), the two well-known frameworks that enable multiple clients to update their models in parallel. Although various model aggregation methods are applicable to these schemes, for a fair comparison, we adopted FedAvg (McMahan et al., 2017) for all three methods, FL, SplitFed and our idea. In FL, the entire model $\mathbf{w}$ is updated at each client and sent to the server for aggregation via FedAvg. In SplitFed and the proposed method, the model $\mathbf{w}$ is split into $\mathbf{w}_C$ and $\mathbf{w}_S$, and the updated models are aggregated via FedAvg.

Model splitting and auxiliary network: For the CNN model that is utilized for MNIST and FMNIST, we split the model and allocate the first 4 convolutional layers to the client, and the remaining 1 convolutional layer and 3 FC layers to the server: we have $|\mathbf{w}_C| = 387,840$ and $|\mathbf{w}_S| = 3,480,330$, i.e., $\frac{|\mathbf{w}_C|}{|\mathbf{w}|} = 0.10$. To minimize the computation burden at the clients, we let the size of the auxiliary network $\mathbf{a}_C$ to be significantly small; a single FC layer with $23,050$ parameters is utilized as the auxiliary network $\mathbf{a}_C$ at the end of $\mathbf{w}_C$, which is 0.60% of the entire model size $|\mathbf{w}|$. For VGG-11 utilized for CIFAR-10, we split the model as $|\mathbf{w}_C| = 972,554$ and $|\mathbf{w}_S| = 8,258,560$ to have $\frac{|\mathbf{w}_C|}{|\mathbf{w}|} = 0.11$. A FC layer with $10,250$ parameters is utilized as the auxiliary network $\mathbf{a}_C$, which is 0.11% size of the entire model $\mathbf{w}$.

Implementation details: At each global round, the server randomly samples $K = 300$ out of $N = 1000$ clients in the system to participate. Cross-entropy loss is adopted for both the client-side and server-side losses. We consider a fixed learning rate of 0.01 and a momentum of 0.9. The mini-batch size is set to 10, and the number of epochs at each client is set to one: at each global round, each client performs 6 local updates for MNIST, FMNIST and 5 local updates for CIFAR-10.

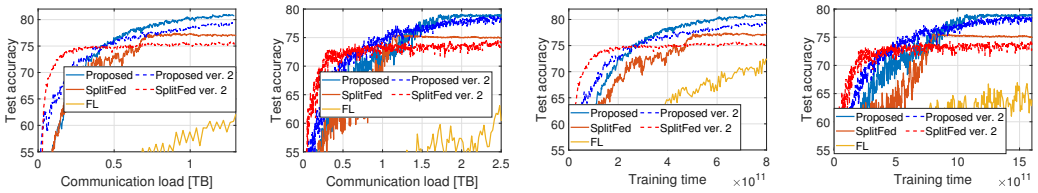
Test accuracy versus communication load: Fig. 2 shows the performance of each method as a function of communication load in both IID and non-IID scenarios. With only a very small size auxiliary network, the proposed idea performs better than SplitFed since the downlink communication for transmitting the backpropagated signals is not required. FL has the worst performance since the entire model $\mathbf{w}$ is transmitted between the server and the clients at every global round.

Test accuracy versus training time: In Fig. 3, we evaluate the test accuracy of each scheme as a function of training time. The training time is evaluated by the latency results in Table 1, where the parameters are set to $P_C = 1$, $P_S = 100$, $R = 1$, $\beta = 0.2$. Additional results with other system parameters are shown in Appendix. Again, it can be seen that our scheme performs better than SplitFed since each client can update the model directly by its local loss function, without waiting for the backpropagated signal from the server. Moreover, FL requires significantly larger communication/computation time compared to our method for transmitting/updating the full model $\mathbf{w}$ at the clients. Table 2 compares the accuracy of each scheme at a specific time (1.5×10^{11} for



(a) $P_C = 1, R = 1$ (b) $P_C = 10, R = 1$ (c) $P_C = 1, R = 4$ (d) $P_C = 10, R = 4$

Figure 4: Effect of client-side computing power P_C and transmission rate R. FMNIST is utilized for training CNN in a non-IID setup. Our scheme is beneficial especially when the clients have relatively small computing powers and small transmission rates (e.g., mobile/IoT devices).



(a) Comm. load, IID (b) Comm. load, non-IID (c) Training time, IID (d) Training time, non-IID

Figure 5: Effect of sequential update process at the server. CIFAR-10 is utilized for training VGG-11.

MNIST, 2.5×10^{11} for FMNIST, and 8×10^{11} for CIFAR-10). The overall results are consistent with the results in Fig. 3, confirming significant advantage of the proposed idea.

Effect of client-side computing power P_C and transmission rate R: In Fig. 4, we observe the performance of each scheme depending on two important parameters, the client-side computing power P_C and the transmission rate R. Other parameters are set to be same as in Fig. 3. If both P_C and R are small, FL requires significant computation/communication time and thus achieves lower performance compared to others. However, as P_C and R increases, FL shows comparable performance with our method since updating the entire model at the clients and transmitting the entire model does not require significant delay with large P_C and R. The overall results confirm the advantage of our scheme in practical scenarios where clients having low computing powers (small P_C) and low transmission rates (small R) aim to train a shared global model.

Variants of SplitFed and proposed scheme: Instead of parallelizing the server-side update process of SplitFed and our scheme, one can think of updating the server-side model sequentially in the order of arrivals of the results from the clients (Thapa et al., 2020). Fig. 5 shows the results of this idea where "Proposed ver. 2" and "SplitFed ver. 2" denote schemes that sequential server-side update process is applied to our idea and SplitFed, respectively. The parameters are set to be same as in Fig. 3. As expected, although the sequential update method can speed up training in the beginning, the convergence to the optimal solution is not theoretically guaranteed and thus achieves a lower accuracy. One can also consider vanilla SplitNN (Gupta & Raskar, 2018) where only one client participates in each global round under the splitted model setup. Note that compared to FL, SplitFed and our scheme, parallel client computation is not available for this conventional SplitNN. Even in this case, our local-loss-based training geared to split learning can be applied to the training process of a participating client to improve the communication efficiency and latency of vanilla SplitNN.

Other experimental results: Various other experimental results with different model splitting, different number of local updates, and different system parameters are given in Appendix.

6 Conclusion

We proposed a new federated split learning scheme that is both fast and efficient in terms of communication/computation burdens. The key idea is to update the client-side and server-side models in parallel via local-loss-based training highly tailored to split learning. We provided an optimal solution on splitting the model to minimize the latency, and presented a theoretical analysis that guarantees convergence of the proposed method. Extensive experimental results confirmed the advantage of our idea compared to FL and SplitFed. We believe that our results provide a new direction to the federated/split learning community for training a large-scale model in practical settings.

REFERENCES

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pp. 583–593. PMLR, 2019.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pp. 736–745. PMLR, 2020.

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

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. *Advances in Neural Information Processing Systems*, 33, 2020.

Yusuke Koda, Jihong Park, Mehdi Bennis, Koji Yamamoto, Takayuki Nishio, Masahiro Morikura, and Kota Nakashima. Communication-efficient multimodal split learning for mmwave received power prediction. *IEEE Communications Letters*, 24(6):1284–1288, 2020.

Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016a.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016b.

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

Michael Laskin, Luke Metz, Seth Nabarrao, Mark Saroufim, Badreddine Noune, Carlo Luschi, Jascha Sohl-Dickstein, and Pieter Abbeel. Parallel training of deep networks with local updates. *arXiv preprint arXiv:2012.03837*, 2020.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2019.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In *International Conference on Machine Learning*, pp. 4839–4850. PMLR, 2019.

Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pp. 2021–2031. PMLR, 2020.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. Detailed comparison of communication efficiency of split learning and federated learning. *arXiv preprint arXiv:1909.09145*, 2019.

Sebastian U Stich. Local sgd converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. *Advances in Neural Information Processing Systems*, 31:4447–4458, 2018.

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

Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

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

Yuwen Xiong, Mengye Ren, and Raquel Urtasun. Loco: Local contrastive representation learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5693–5700, 2019.

A ALGORITHM DESCRIPTION

The detailed procedure of our method is summarized in Algorithm 1.

Algorithm 1 Proposed Training Algorithm

Input: Initialized model $\mathbf{w}^0 = [\mathbf{w}_C^0, \mathbf{w}_S^0]$ and the auxiliary network $\mathbf{a}_C^0$
Output: Model $\mathbf{w}^T = [\mathbf{w}_C^T, \mathbf{w}_S^T]$ after T global rounds
for each global round $t = 0, 1, ..., T-1$ **do**
Step 1 and Step 2: Model download, forward propagation and upload
for each client $k \in A_t$ **in parallel do**
  Download $\mathbf{w}_C^t$ and $\mathbf{a}_C^t$ from the fed server and let $\mathbf{w}_{C,k}^t = \mathbf{w}_C^t$, $\mathbf{a}_{C,k}^t = \mathbf{a}_C^t$
  Calculate $g_{\mathbf{w}_{C,k}^t}(x)$ for all $x \in \tilde{D}_k$ // Forward propagation
  Send $g_{\mathbf{w}_C^t}(x)$ for all $x \in \tilde{D}_k$ and the corresponding labels to the main server
end for
Step 3: Client update and aggregation
for each client $k \in A_t$ **in parallel do**
  $\mathbf{w}_{C,k}^{t+1} = \mathbf{w}_{C,k}^t - \eta_t \tilde{\nabla}_{\mathbf{w}} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t)$ // Update client model with client-side local loss
  $\mathbf{a}_{C,k}^{t+1} = \mathbf{a}_{C,k}^t - \eta_t \tilde{\nabla}_{\mathbf{a}} F_{C,k}(\mathbf{w}_{C,k}^t, \mathbf{a}_{C,k}^t)$ // Update auxiliary network with client-side local loss
  Transmit $\mathbf{w}_{C,k}^{t+1}$ and $\mathbf{a}_{C,k}^{t+1}$ to the fed server
end for
$\mathbf{w}_C^{t+1} = \frac{1}{K} \sum_{k \in A_t} \mathbf{w}_{C,k}^{t+1}$ // Aggregation of client-side models
$\mathbf{a}_C^{t+1} = \frac{1}{K} \sum_{k \in A_t} \mathbf{a}_{C,k}^{t+1}$ // Aggregation of auxiliary networks
Step 4: Server update and aggregation (in parallel with Step 3)
for $k \in A_t$ **in parallel do**
  Receive $g_{\mathbf{w}_{C,k}^t}(x)$ from each client k
  $\mathbf{w}_{S,k}^{t+1} = \mathbf{w}_S^t - \eta_t \tilde{\nabla} F_{S,k}(\mathbf{w}_S^t, \mathbf{w}_C^t)$ // Update server model with server-side local loss
end for
$\mathbf{w}_S^{t+1} = \frac{1}{K} \sum_{k \in A_t} \mathbf{w}_{S,k}^{t+1}$ // Aggregation of server-side models
end for

B PROOF OF THEOREM 1

Case 1: We first consider the case with

$$\alpha \geq \frac{1}{P_S \left(\frac{1}{R|D|} + \frac{1-\beta}{P_C K} \right) + 1}, \tag{5}$$

which is equivalent to $\frac{\alpha|\mathbf{w}|K}{R} + \frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C} \geq \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}$. Hence, we have $\max\left(\frac{\alpha|\mathbf{w}|K}{R} + \frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}, \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S} \right) = \frac{\alpha|\mathbf{w}|K}{R} + \frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}$. Now the latency of our scheme can be rewritten as

$$\frac{(q|D| + \alpha|\mathbf{w}|)K}{R} + \frac{\alpha\beta|D||\mathbf{w}|}{P_C} + \frac{\alpha|\mathbf{w}|K}{R} + \frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}, \tag{6}$$

which is an increasing function of α. To minimize this latency in the range of $\alpha \geq \frac{1}{P_S \left(\frac{1}{R|D|} + \frac{1-\beta}{P_C K} \right) + 1}$, the optimal solution is $\alpha = \frac{1}{P_S \left(\frac{1}{R|D|} + \frac{1-\beta}{P_C K} \right) + 1}$.

Case 2: Now consider

$$\alpha \leq \frac{1}{P_S \left(\frac{1}{R|D|} + \frac{1-\beta}{P_C K} \right) + 1}, \tag{7}$$

which leads to $\frac{\alpha|\mathbf{w}|K}{R} + \frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C} \leq \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}$, i.e., $\max\left(\frac{\alpha|\mathbf{w}|K}{R} + \frac{\alpha(1-\beta)|D||\mathbf{w}|}{P_C}, \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S} \right) = \frac{(1-\alpha)|D||\mathbf{w}|K}{P_S}$. In this case, the latency of

our scheme becomes

$$\left(\frac{|\mathbf{w}|K}{R}+\frac{\beta|D||\mathbf{w}|}{P_C}-\frac{|D||\mathbf{w}|K}{P_S}\right)\alpha+\frac{q|D|}{R}+\frac{|D||\mathbf{w}|K}{P_S}. \tag{8}$$

Here, if $P_S \leq (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, the latency is a decreasing function of α and the optimal solution minimizing the latency becomes $\alpha = \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}$ in the range of $\alpha \leq \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}$. Otherwise, i.e., $P_S > (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, latency is an increasing function of α.

Now we combine the results of both case 1 and case 2. If $P_S \leq (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, the optimal solution minimizing the latency becomes $\alpha = \frac{1}{P_S\left(\frac{1}{R|D|}+\frac{1-\beta}{P_C K}\right)+1}$. Otherwise, i.e., if $P_S > (\frac{1}{R|D|}+\frac{\beta}{P_C K})^{-1}$, the latency is an increasing function of α, which completes the proof.

C Proof of Theorem 2

For notational simplicity, we let $f_{C,k}(\mathbf{w}_C^t) := F_{C,k}(\mathbf{w}_C^t, \mathbf{a}_C^t)$ and $f_{S,k}(\mathbf{w}_S^t) := F_{S,k}(\mathbf{w}_S^t, \mathbf{w}_C^t)$.

C.1 Convergence of client-side model

Due to the L-smoothness of client-side loss function, we can write

$$F_C(\mathbf{w}_C^{t+1}) \leq F_C(\mathbf{w}_C^t) + \nabla F_C(\mathbf{w}_C^t)^T(\mathbf{w}_C^{t+1}-\mathbf{w}_C^t) + \frac{L}{2}\|\mathbf{w}_C^{t+1}-\mathbf{w}_C^t\|^2. \tag{9}$$

Note that we have

$$\mathbf{w}_C^{t+1} = \frac{1}{K}\sum_{k\in A_t}\left(\mathbf{w}_C^t - \eta_t\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t)\right) \tag{10}$$

$$= \mathbf{w}_C^t - \eta_t\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t) \tag{11}$$

where $\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t) = \frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k}\nabla\ell(x;\mathbf{w}_{C,k})$ for a given mini-batch $\tilde{D}_k \subset D_k$. Hence, we can rewrite equation 9 as follows:

$$F_C(\mathbf{w}_C^{t+1}) \leq F_C(\mathbf{w}_C^t) - \eta_t\nabla F_C(\mathbf{w}_C^t)^T\left(\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t)\right) + \frac{L}{2}\eta_t^2\left\|\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t)\right\|^2. \tag{12}$$

Now by taking the expectations at both sides of equation 12, we have

$$\mathbb{E}[F_C(\mathbf{w}_C^{t+1})] \leq \mathbb{E}[F_C(\mathbf{w}_C^t)] - \eta_t\underbrace{\mathbb{E}\left[\nabla F_C(\mathbf{w}_C^t)^T\left(\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t)\right)\right]}_{B_1} \tag{13}$$

$$+\frac{L}{2}\eta_t^2\underbrace{\mathbb{E}\left[\left\|\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t)\right\|^2\right]}_{B_2} \tag{14}$$

In the following, we will bound B_1 and B_2, respectively.

We first consider B_1. By defining X as

$$X = \frac{1}{K}\sum_{k\in A_t}\left(\tilde{\nabla}f_{C,k}(\mathbf{w}_C^t) - \nabla f_{C,k}(\mathbf{w}_C^t)\right), \tag{15}$$

we can find the lower bound of B_1 as

$$B_1 = \mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\Big)\Big] \tag{16}$$

$$= \mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(X + \frac{1}{K}\sum_{k\in A_t} \nabla f_{C,k}(\mathbf{w}_C^t)\Big)\Big] \tag{17}$$

$$\geq \underbrace{\mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(\frac{1}{K}\sum_{k\in A_t} \nabla f_{C,k}(\mathbf{w}_C^t)\Big)\Big]}_{C_1} - \underbrace{\|\mathbb{E}[\nabla F_C(\mathbf{w}_C^t)^T X]\|}_{C_2}. \tag{18}$$

Since $f_{C,k}(\mathbf{w}_C^t) := F_{C,k}(\mathbf{w}_C^t, \mathbf{a}_C^t)$ and A_t is chosen uniformly at random among N clients in the system, by the law of total expectation, we can write

$$C_1 = \mathbb{E}[\|\nabla F_C(\mathbf{w}_C^t)\|^2]. \tag{19}$$

Now we consider C_2. Note that since $\tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)$ is an unbiased estimator of $\nabla f_{C,k}(\mathbf{w}_C^t)$, we have

$$\|\mathbb{E}[\tilde{\nabla} f_{C,k}(\mathbf{w}_C^t) - \nabla f_{C,k}(\mathbf{w}_C^t)]\| = 0. \tag{20}$$

We also note that $\mathbb{E}[U^T V] \leq \frac{1}{4}\mathbb{E}[\|U\|^2] + \mathbb{E}[\|V\|^2]$ holds for any vectors U and V. Hence, we have

$$C_2 = \big\|\mathbb{E}[\nabla F_C(\mathbf{w}_C^t)^T X]\big\| \tag{21}$$

$$= \big\|\mathbb{E}\left[\mathbb{E}\left[\nabla F_C(\mathbf{w}_C^t)^T X|\Omega\right]\right]\big\| \tag{22}$$

$$= \big\|\mathbb{E}\left[\nabla F_C(\mathbf{w}_C^t)^T \mathbb{E}\left[X|\Omega\right]\right]\big\| \tag{23}$$

$$\leq \frac{1}{4}\mathbb{E}[\|\nabla F_C(\mathbf{w}_C^t)\|^2] + \mathbb{E}\left[\|\mathbb{E}\left[X|\Omega\right]\|^2\right] \tag{24}$$

$$= \frac{1}{4}\mathbb{E}[\|\nabla F_C(\mathbf{w}_C^t)\|^2]. \tag{25}$$

where the last equality comes from equation 20.

By inserting equation 19 and equation 25 to equation 18, we obtain

$$B_1 = \mathbb{E}\Big[\nabla F_C(\mathbf{w}_C^t)^T \Big(\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\Big)\Big] \geq \frac{3}{4}\mathbb{E}[\|\nabla F_C(\mathbf{w}_C^t)\|^2]. \tag{26}$$

Now we consider B_2 in equation 13. We can bound B_2 as

$$B_2 = \|\frac{1}{K}\sum_{k\in A_t} \tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\|^2 \tag{27}$$

$$\underset{(a)}{\leq} \frac{1}{K}\sum_{k\in A_t} \|\tilde{\nabla} f_{C,k}(\mathbf{w}_C^t)\|^2 \tag{28}$$

$$= \frac{1}{K}\sum_{k\in A_t} \left\|\frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k} \nabla \ell(x; \mathbf{w}_C^t)\right\|^2 \tag{29}$$

$$\underset{(b)}{\leq} \frac{1}{K}\sum_{k\in A_t} \frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k} \left\|\nabla \ell(x; \mathbf{w}_C^t)\right\|^2 \tag{30}$$

$$\underset{(c)}{\leq} G_1 \tag{31}$$

where (a) and (b) comes from the Cauchy-Schwarz inequality and (c) comes from Assumption 2.

By inserting equation 26 and equation 31 to equation 13, we obtain

$$\mathbb{E}[F_C(\mathbf{w}_C^{t+1})] \leq \mathbb{E}[F_C(\mathbf{w}_C^t)] - \frac{3}{4}\eta_t \mathbb{E}[||\nabla F_C(\mathbf{w}_C^t)||^2] + \frac{G_1 L}{2}\eta_t^2. \tag{32}$$

Now by summing up for all global rounds $t = 0, 1, ...T-1$, we have

$$\mathbb{E}[F_C(\mathbf{w}_C^T)] \leq \mathbb{E}[F_C(\mathbf{w}_C^0)] - \frac{3}{4}\sum_{t=0}^{T-1}\eta_t \mathbb{E}[||\nabla F_C(\mathbf{w}_C^t)||^2] + \frac{G_1 L}{2}\sum_{t=0}^{T-1}\eta_t^2. \tag{33}$$

Finally from $F_C(\mathbf{w}_C^*) \leq \mathbb{E}[F_C(\mathbf{w}_C^T)]$, we can write

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \mathbb{E}[||\nabla F_C(\mathbf{w}_C^t)||^2] \leq \frac{4(F_C(\mathbf{w}_C^0) - F_C(\mathbf{w}_C^*))}{3\Gamma_T} + \frac{G_1 L}{2}\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t^2 \tag{34}$$

which completes the proof for the client-side model.

C.2 Convergence of server-side model

Due to the L-smoothness of the server-side loss function, following the same procedure of the client-side model, we have

$$\mathbb{E}[F_S(\mathbf{w}_S^{t+1})] \leq \mathbb{E}[F_S(\mathbf{w}_S^t)] - \eta_t \underbrace{\mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t)\Big)\Big]}_{B_3} \tag{35}$$

$$+ \frac{L}{2}\eta_t^2 \underbrace{\mathbb{E}[\|\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t)\|^2]}_{B_4} \tag{36}$$

where $\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t) = \frac{1}{|\tilde{D}_k|}\sum_{x\in\tilde{D}_k}\nabla\ell(g_{\mathbf{w}_{C,k}^t}(x); \mathbf{w}_S^t)$ for a given mini-batch $\tilde{D}_k \subset D_k$. We will derive the bounds of B_3 and B_4.

We first consider B_3. By defining X as

$$X = \frac{1}{K}\sum_{k\in A_t}\Big(\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t) - \nabla f_{S,k}(\mathbf{w}_S^t)\Big), \tag{37}$$

we can write

$$\mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\tilde{\nabla} f_{S,k}(\mathbf{w}_S^t)\Big)\Big] \tag{38}$$

$$= \mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(X + \frac{1}{K}\sum_{k\in A_t}\nabla f_{S,k}(\mathbf{w}_S^t)\Big)\Big] \tag{39}$$

$$\geq \mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\nabla f_{S,k}(\mathbf{w}_S^t)\Big)\Big] - \|\mathbb{E}[\nabla F_S(\mathbf{w}_S^t)^T X]\| \tag{40}$$

$$\geq \underbrace{\mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\nabla F_{S,k}(\mathbf{w}_S^t)\Big)\Big]}_{C_1} + \underbrace{\mathbb{E}\Big[\nabla F_S(\mathbf{w}_S^t)^T\Big(\frac{1}{K}\sum_{k\in A_t}\big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t)\big)\Big)\Big]}_{C_2} \tag{41}$$

$$- \underbrace{\|\mathbb{E}[\nabla F_S(\mathbf{w}_S^t)^T X]\|}_{C_3} \tag{42}$$

As can be seen in the derivation for the client-side model, we have

$$C_1 = \mathbb{E}[\|\nabla F_S(\mathbf{w}_S^t)\|^2] \tag{43}$$

and

$$C_3 \leq \frac{1}{4}\mathbb{E}[\|\nabla F_S(\mathbf{w}_S^t)\|^2]. \tag{44}$$

Now we analyze C_2. We have

$$\nabla F_S(\mathbf{w}_S^t)^T \Big(\frac{1}{K} \sum_{k \in A_t} \big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t) \big) \Big) \tag{45}$$

$$\geq - \left\| \nabla F_S(\mathbf{w}_S^t)^T \Big(\frac{1}{K} \sum_{k \in A_t} \Big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t) \Big) \Big) \right\| \tag{46}$$

$$\geq - \big\| \nabla F_S(\mathbf{w}_S^t) \big\| \left\| \frac{1}{K} \sum_{k \in A_t} \big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t) \big) \right\| \tag{47}$$

$$\geq - \sqrt{G_2} \left\| \frac{1}{K} \sum_{k \in A_t} \big(\nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t) \big) \right\| \tag{48}$$

$$\geq - \sqrt{G_2} \frac{1}{K} \sum_{k \in A_t} \big\| \nabla f_{S,k}(\mathbf{w}_S^t) - \nabla F_{S,k}(\mathbf{w}_S^t) \big\| \tag{49}$$

$$= - \sqrt{G_2} \frac{1}{K} \sum_{k \in A_t} \left\| \frac{1}{|D_k|} \sum_{x \in D_k} \nabla \ell(g_{\mathbf{w}_C^t}(x); \mathbf{w}_S) - \frac{1}{|D_k|} \sum_{x \in D_k} \nabla \ell(g_{\mathbf{w}_C^*}(x); \mathbf{w}_S) \right\| \tag{50}$$

$$= - \sqrt{G_2} \frac{1}{K} \sum_{k \in A_t} \left\| \int \nabla \ell(z; \mathbf{w}_S^t) p_{C,k}^t(z) dz - \int \nabla \ell(z; \mathbf{w}_S^t) p_{C,k}^*(z) dz \right\| \tag{51}$$

$$\geq - \sqrt{G_2} \frac{1}{K} \sum_{k \in A_t} \int \big\| \nabla \ell(z; \mathbf{w}_S^t) \big\| \big\| p_{C,k}^t(z) - p_{C,k}^*(z) \big\| \, dz \tag{52}$$

$$\geq - G_2 \frac{1}{K} \sum_{k \in A_t} d_{C,k}^t. \tag{53}$$

Now we can write

$$C_2 \geq - \mathbb{E} \left[G_2 \frac{1}{K} \sum_{k \in A_t} d_{C,K}^t \right] \tag{54}$$

$$= - G_2 \frac{1}{N} \sum_{k=1}^{N} d_{C,k}^t \tag{55}$$

since A_t chosen uniformly at random among N clients in the system.

By utilizing the results of C_1, C_2, C_3, we have

$$B_3 = \mathbb{E} \Big[\nabla F_S(\mathbf{w}_S^t)^T \Big(\frac{1}{K} \sum_{k \in A_t} \tilde{\nabla} f_{S,k}(\mathbf{w}_S^t) \Big) \Big] \geq \frac{3}{4} \mathbb{E}[\|\nabla F_S(\mathbf{w}_S^t)\|^2] - G_2 \frac{1}{N} \sum_{k=1}^{N} d_{C,k}^t \tag{56}$$

Following the same procedure of the client-side model, for B_4, we have

$$B_4 = \| \frac{1}{K} \sum_{k \in A_t} \tilde{\nabla} f_{S,k}(\mathbf{w}_S^t) \|^2 \leq G_2. \tag{57}$$

Now by inserting the results of equation 56 and equation 57 to equation 35, we have

$$\mathbb{E}[F_S(\mathbf{w}_S^{t+1})] \leq \mathbb{E}[F_S(\mathbf{w}_S^t)] - \frac{3}{4} \eta_t \mathbb{E}[||\nabla F_S(\mathbf{w}_S^t)||^2] + \eta_t G_2 \frac{1}{N} \sum_{k=1}^{N} d_{C,k}^t + \frac{L G_2}{2} \eta_t^2. \tag{58}$$

Summing up for all global rounds $t = 0, 1, ...T-1$, we have

$$\mathbb{E}[F_S(\mathbf{w}_C^T)] \leq \mathbb{E}[F_S(\mathbf{w}_C^0)] - \frac{3}{4} \sum_{t=0}^{T-1} \eta_t \mathbb{E}[||\nabla F_S(\mathbf{w}_C^t)||^2] + G_2 \sum_{t=0}^{T-1} \left(\eta_t \frac{1}{N} \sum_{k=1}^{N} d_{C,k}^t + \frac{L}{2} \eta_t^2 \right). \tag{59}$$

Finally from $F_S(\mathbf{w}_S^*) \leq \mathbb{E}[F_S(\mathbf{w}_S^T)]$, we can write

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t\mathbb{E}[||\nabla F_S(\mathbf{w}_S^t)||^2] \leq \frac{4(F_S(\mathbf{w}_S^0)-F_S(\mathbf{w}_S^*))}{3\Gamma_T} + G_2\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\left(\eta_t\frac{1}{N}\sum_{k=1}^{N}d_{C,k}^t + \frac{L}{2}\eta_t^2\right) \tag{60}$$

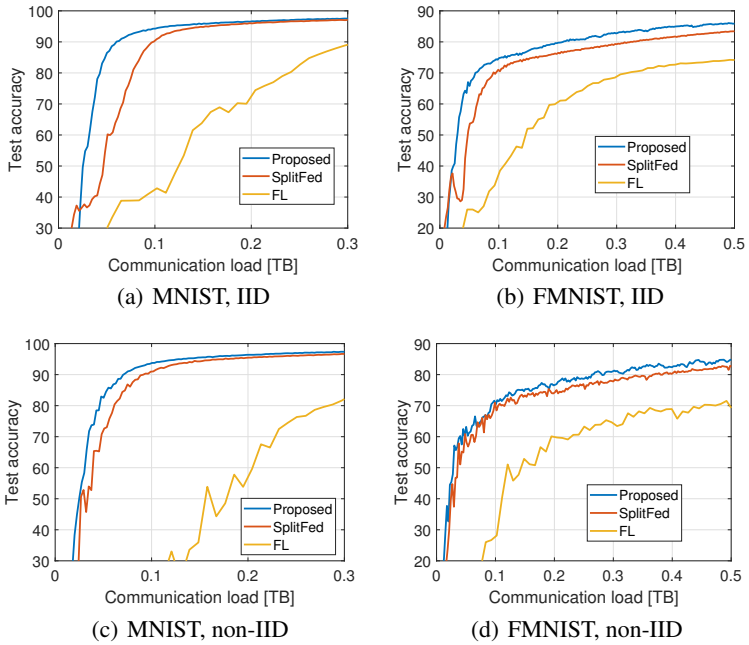
which completes the proof for the server-side model.

D Additional Experimental Results

D.1 Results with different model splitting

To confirm the advantage of the proposed idea further, we performed additional experiments with different model splitting. Here, we allocate more layers to the clients compared to the setup in the main manuscript: for training CNN with MNIST and FMNIST, we allocate 5 convolutional layers to the client and the remaining 3 fully connected (FC) layers to the server: we have $|\mathbf{w}_C| = 997,920$, $|\mathbf{w}_S| = 2,890,250$, i.e., $\frac{|\mathbf{w}_C|}{|\mathbf{w}|} = 0.26$. A single FC layer with $23,050$ parameters is utilized as the auxiliary network at the end of $\mathbf{w}_C$. Hence, the size of the auxiliary network is 0.60% of the full model $\mathbf{w}$.

In Fig. 6, we plot the test accuracy versus communication load for all schemes. Since more layers are allocated to the clients compared to the setup in the main manuscript, the communication load is increased for both our scheme and SplitFed. Hence, the gap between the SL-based methods (including our scheme) and FL is reduced. We also note that the gap between our scheme and SplitFed is reduced, since the size of the client-side model $|\mathbf{w}_C|$ becomes dominant compared to the size of the cut layer. Our scheme still performs the best, confirming the advantage of our method that enables the clients to update their models without receiving the backpropagated gradients from the server.

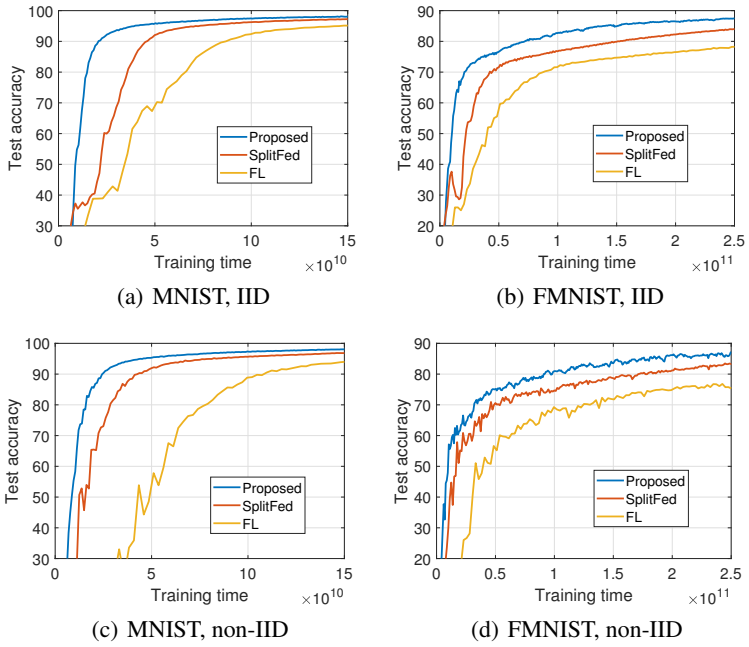


(a) MNIST, IID (b) FMNIST, IID

(c) MNIST, non-IID (d) FMNIST, non-IID

Figure 6: Test accuracy versus communication load. Here, we split the model in a different way compared to the setup in the main manuscript: more layers are allocated to the clients compared to the plots the main manuscript.

Fig. 7 shows the test accuracy of different schemes as a function of training time. The parameters are set to $P_C = 1$, $P_S = 100$, $R = 1$, $\beta = 0.2$. In Table 3, we also compare the test accuracy of each



(a) MNIST, IID (b) FMNIST, IID

(c) MNIST, non-IID (d) FMNIST, non-IID

Figure 7: Test accuracy versus training time. Here, we split the model in a different way compared to the setup in the main manuscript: more layers are allocated to the clients compared to the plots the main manuscript.

Table 3: Performance of different schemes at a specific time in Fig. 7.

	MNIST		FMNIST	
Methods	IID	Non-IID	IID	Non-IID
FL	92.80%	89.02%	78.21%	75.77%
SplitFed	97.22%	96.83%	83.92%	83.31%
Proposed	**98.07%**	**98.05%**	**87.48%**	**87.11%**

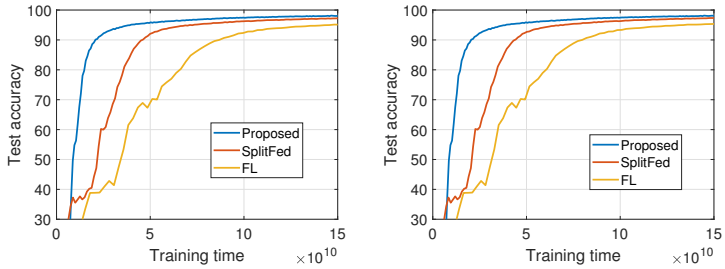
scheme at a specific time (1.5×10^{11} for MNIST and 2.5×10^{11} for FMNIST). The overall results confirm the advantage of our local-loss-based training tailored to the split learning setup.

D.2 Results with different system parameters

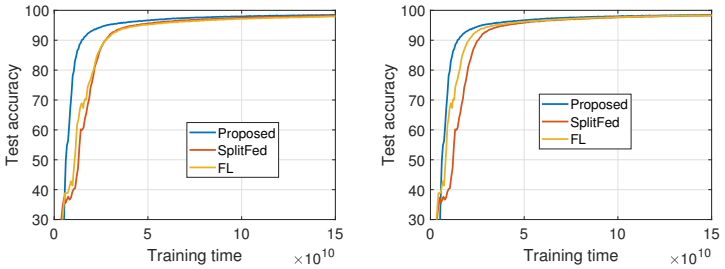
In Fig. 8, we provide additional results with varying client-side computing power P_C and transmission rate R. The model is splitted as in subsection D.1, and other system parameters are the same as in Fig. 7. As in the results in the main manuscript, it can be seen that our scheme is always better than SplitFed since each client can update the model directly by its local loss function. Moreover, as P_C and R increase, FL can have better performance compared to SplitFed. In practical settings where the clients have relatively low computing powers and transmission rates (e.g., mobile/IoT devices), our scheme outperforms existing FL and SL-based ideas.

D.3 Results with different number of local updates

To see the effect of number of local udpates, we performed additional experiments with CIFAR-10 by increasing the number of local epochs from 1 to 5. Hence, the number of local updates is increased from 5 to 25 compared to the setup in the main manuscript. We consider a non-IID setup where other setups are exactly the same as in the main manuscript. After consuming 2TB of communication load, the accuracies are 73.47%, 68.85%, 41.95% for our scheme, SplitFed, FL with 25 local updates. In contrast, the accuracies are 79.01%, 75.06%, 57.34% with only 5 local updates. It can be seen that the performance of all schemes (including our method) are degraded with more local updates, which



(a) $P_C = 1, R = 1$ (b) $P_C = 10, R = 1$



(c) $P_C = 1, R = 4$ (d) $P_C = 10, R = 4$

Figure 8: Test accuracy with varying system parameters. Here, we split the model in a different way compared to the setup in the main manuscript: more layers are allocated to the clients compared to the plots the main manuscript. MNIST is utilized for training CNN in an IID setup. Our scheme is beneficial especially when the computing powers and and the transmission rates of the clients are low (e.g., mobile/IoT devices).

leads to a slower convergence. This is because the local models become more biased in a non-IID setup when the number of local updates is too large. Hence, setting the number of local updates too large should be avoided. The trend is consistent with the results in the main manuscript, confirming the advantage of the proposed approach.