LocFedMix-SL: Localize, Federate, and Mix for Improved Scalability, Convergence, and Latency in Split Learning

Seungeun Oh
seoh@ramo.yonsei.ac.kr
Yonsei University
Seoul, Korea

Jihong Park*
jihong.park@deakin.edu.au
Deakin University
Victoria, Australia

Praneeth Vepakomma
vepakom@mit.edu
Massachusetts Institute of Technology
Massachusetts, United States

Sihun Baek
shbaek@ramo.yonsei.ac.kr
Yonsei University
Seoul, Korea

Ramesh Raskar
raskar@media.mit.edu
Massachusetts Institute of Technology
Massachusetts, United States

Mehdi Bennis
mehdi.bennis@oulu.fi
University of Oulu
Oulu, Finland

Seong-Lyun Kim*
slkim@ramo.yonsei.ac.kr
Yonsei University
Seoul, Korea

ABSTRACT

Split learning (SL) is a promising distributed learning framework that enables to utilize the huge data and parallel computing resources of mobile devices. SL is built upon a model-split architecture, wherein a server stores an upper model segment that is shared by different mobile clients storing its lower model segments. Without exchanging raw data, SL achieves high accuracy and fast convergence by only uploading smashed data from clients and downloading global gradients from the server. Nonetheless, the original implementation of SL sequentially serves multiple clients, incurring high latency with many clients. A parallel implementation of SL has great potential in reducing latency, yet existing parallel SL algorithms resort to compromising scalability and/or convergence speed. Motivated by this, the goal of this article is to develop a scalable parallel SL algorithm with fast convergence and low latency. As a first step, we identify that the fundamental bottleneck of existing parallel SL comes from the model-split and parallel computing architectures, under which the server-client model updates are often imbalanced, and the client models are prone to detach from the server's model. To fix this problem, by carefully integrating local parallelism, federated learning, and mixup augmentation techniques, we propose a novel parallel SL framework, coined *LocFedMix-SL*. Simulation results corroborate that LocFedMix-SL achieves improved scalability, convergence speed, and latency, compared to sequential SL as well as the state-of-the-art parallel SL algorithms such as SplitFed and LocSplitFed.

*Corresponding Authors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WWW '22, April 25–29, 2022, Virtual Event, Lyon, France.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512153

CCS CONCEPTS

• Computer systems organization → Client-server architectures; **• Theory of computation → Distributed algorithms**; **• Computing methodologies → Parallel algorithms**.

KEYWORDS

Split Learning, Federated Learning, Local Parallelism, Mixup, Scalability

ACM Reference Format:
Seungeun Oh, Jihong Park, Praneeth Vepakomma, Sihun Baek, Ramesh Raskar, Mehdi Bennis, and Seong-Lyun Kim. 2022. LocFedMix-SL: Localize, Federate, and Mix for Improved Scalability, Convergence, and Latency in Split Learning. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3485447.3512153

1 INTRODUCTION

Big data and huge computing resources are essential in enabling high-quality deep learning. In this respect, the indispensable elements are the utilization of massive amounts of data and the provision of large-scale parallel computing power [5, 12, 20, 28]. The sheer amount of Internet-of-Things (IoT) or Web-of-Things (WoT) clients are great sources of these two elements. They collectively provide a huge volume of data and high parallel-computing power, although each individual client has only a tiny fraction of data with limited computing capabilities [8, 22]. *Federated learning (FL)* [10] is the first of its kind towards exploiting these dispersed data and computing resources. In FL, each client independently trains a local neural network model using its own data, followed by periodically exchanging their model parameters that are aggregated and averaged by a *server* [10, 32]. In doing so, each client can reflect the data owned and processed by other clients without directly exchanging raw data. However, FL is still insufficient to run deep learning on WoT clients particularly for large-sized deep neural network models, since storing and exchanging model parameters impose excessive memory, computing and, communication overhead on the clients.

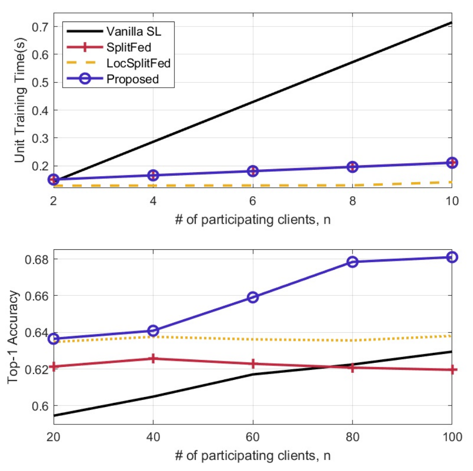

Figure 1: Computational latency per communication round (top) and top-1 accuracy w.r.t. # of clients, i.e., scalability (bottom). A single communication round is defined as the elapsed time until all clients update their model parameters.

In order to cope with the resource efficiency problem in distributed deep neural networks, *split learning (SL)* is an alternative solution [27]. SL divides a deep neural network model into two parts, such that a server stores the *upper model segment* shared by all clients, while each client stores the *lower model segment* [5]. In between the upper and lower model segments, i.e., at the cut layer, SL exchanges the cut-layer representations in the forward propagation (FP), also known as the *smashed data*, and the gradients in the backward propagation (BP). In vanilla SL, the server connects to clients one by one. Under these sequential operations, vanilla SL guarantees its *scalability* in a sense that the accuracy increases with the number of clients. Furthermore, as compared with FL, SL achieves even faster *convergence* in terms of the number of communication rounds until training converges [28]. Notwithstanding, as observed in Fig. 1 (top), the computing and communication *latency* of vanilla SL per communication round keeps increasing with the number of clients due to its sequential operations.

To overcome the sequential processing latency of vanilla SL, in this article, we aim to develop a *parallel SL* framework that ultimately enables the sheer amount of WoT clients to collectively train distributed deep neural networks, while achieving the following three goals: scalability, fast convergence, and low latency. Achieving this tri-fold goal is however nontrivial. Indeed, existing parallel SL algorithms struggle with slow convergence, and what is more, they are not scalable. In particular, to catch up the convergence speed of vanilla SL, *split federated learning (SplitFed)* integrates FL into SL, and exchanges lower model segments across clients after BP [26]. Similarly, inspired by local parallel learning [1, 14], the method in [7], hereafter referred to as *localized SplitFed (LocSplitFed)*, additionally updates each lower model segment using the *local gradients* computed within each client model segment. Unfortunately, both SplitFed and LocSplitFed are not scalable, as illustrated in Fig. 1 (bottom). The fundamental reason for such limited scalability is inherent to the parallel SL model architecture as elaborated below.

1. Server-Client Update Imbalance Problem. In parallel SL, a single upper model segment is concurrently connected with multiple lower segments. Therefore, while each lower model segment is updated once in the BP, the shared upper segment is updated multiple times. In other words, the effective learning rate of the upper model segment is higher than that of each lower model segment. A higher learning rate often requires more training samples, i.e., larger batch sizes [25], yet the upper model segment in parallel SL has no access to client's data. Alternatively, the lower model segment averaging in [7, 26] and local gradients in [7] partly ameliorate the imbalance by additionally training the lower model segments, which may however encounter another issue as detailed below.

2. Client Model Detachment Problem. Separately updating the lower model segments may bring about detaching them from the upper model segment. Lower model segments averaging and local gradients fall into this case, ignoring the updates in the upper model segment. Indeed, layer-wise model parameter averaging does not guarantee high accuracy without sophisticated techniques [2]. Likewise, local gradients make the lower model segments forget about the existence of the upper segment, failing to utilize the full capacity of a deep neural network. The BP from the server, i.e., *global gradients*, partly addresses this issue, yet cannot entirely weave the lower and upper segments due to its ignoring or lagging across the FP's.

To resolve the aforementioned imbalance and detachment problems, in this article we propose a novel parallel SL framework, coined *localized, federated, and mixed SL (LocFedMix-SL)*. The key new elements are locally regularizing the lower model segment at each client, and augmenting smashed data at the server. The *local regularizer* encourages to maximize the mutual information between the raw and smashed data in the FP, while not allowing each lower model segment to extract too much of the original features before reaching the upper segment. Next, the server combinatorially superpositions smashed data uploaded from different clients, producing new augmented smashed data in the FP. This *smashed data mixup* increases the effective batch size at the upper model segment so as to match its higher effective learning rate. Given these modified operations in the FP, lower model segment averaging after BP can finally improve the convergence speed and accuracy while achieving scalability, resulting in LocFedMix-SL.

Contributions. This work has the following contributions.

- We identify that existing parallel SL algorithms achieve neither scalability nor fast convergence, as they fail to balance FP flows and BP updates under the server-client split parallel SL architecture.
- To fix this issue, by leveraging local parallelism, federated learning, and mixup data augmentation techniques, we propose a novel parallel SL framework, *LocFedMix-SL*.
- Simulation results corroborate that LocFedMix-SL achieves higher scalability, faster convergence, and lower latency compared to vanilla SL as well as the state-of-the-art parallel SL algorithms including SplitFed and LocSplitFed.

Related Works. FL and SL are two spearheads in distributed learning with siloed data, and possess their own pros and cons [21]. FL achieves scalable accuracy with many clients [11, 18]. Yet, it is limited to only handling small models due to the constraints

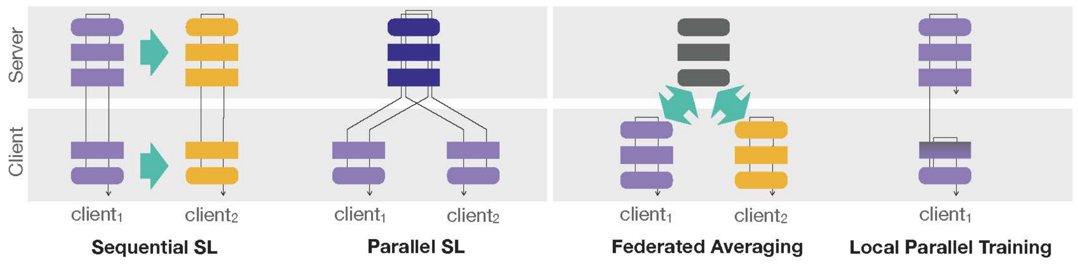

Figure 2: Schematic illustrations of sequential SL (i.e., vanilla SL [28]) and parallel SL, as well as federated learning (i.e., federated averaging [18]) and local parallel learning [14].

of compute, memory, and communication resources available on the clients. SL can instead run large models via model splitting [6, 23, 28, 28], while achieving even faster convergence with lower communication overhead than FL [3, 4, 24]. However, the scalability in its vanilla form is questionable, which is particularly critical for WoT scenarios where the global data and computing power are dispersed across the sheer number of clients.

SplitFed is the first hybrid of its kind, aiming to achieve advantages from both the worlds by applying FL across clients after SL backpropagation completes [9, 26]. LocSplitFed [7] additionally applies local model parallel training [14, 30] at clients after SL backpropagation finishes. In essence, these methods mitigate the server-client update imbalance problem in the BP, thereby improving the convergence speed. Nevertheless, they ignore the client model detachment problem in the FP, and fail to achieve the scalability. By contrast, our proposed LocFedMix-SL aims to address both the imbalance and detachment problem by additionally applying a novel smashed data augmentation method during the FP, which is inspired from the Mixup data augmentation method [34] and its variants [29, 33]. Consequently, LocFedMix-SL is scalable while maintaining all the benefits of SplitFed and LocSplitFed.

Note that the operations of LocFedMix-SL partly coincide with those of LocSplitFed in [7], yet they are still different. The main focus in [7] is to propose LocSplitFed and prove its convergence, ignoring any discussion on whether the accuracy is scalable with many clients. In contrast, we aim to first understand the fundamental principles warranting scalability to develop a scalable SL framework under limited bandwidth and latency constraints. With extensive simulations, we discovered (i) local model averaging and (ii) local model parallel training are essential in achieving scalability by overcoming the server-client update imbalance in the BP. For this reason, the operations of our proposed LocFedMix-SL partly coincide with those of LocSplitFed that is also laid by (i) and (ii).

Compared to LocSplitFed, LocFedMix-SL additionally utilizes the aforementioned (iii) smashed data augmentation and (iv) the global gradient backpropagation from the server to client models. In fact, (iv) is included in vanilla SL, yet is omitted and replaced simply with (ii) in LocSplitFed, due to mathematical amenability for its convergence analysis. We discovered that (iv) is crucial in achieving scalability and fast convergence, by making client model training undetached from server model training. For this reason, we restored this functionality of vanilla SL back to LocFedMix-SL that finally achieves scalability. Note also that our follow-up work tackles the same update imbalance and detachmenet problems but using a different technique that separately adjusts the client and server side learning rates [19]. It achieves scalability without additional server-side computation such as smashed data mixup, but the accuracy is lower than LocFedMix-SL. Integrating these two complementary techniques may yield a more energy-efficient and scalable SL framework, which is deferred to future research.

2 FROM SEQUENTIAL TO PARALLEL SL

In this section, we describe two SL structures, which are classified into sequential SL and parallel SL. Both sequential SL and parallel SL are laid by the same network architecture. Specifically, the network architecture under study consists of a set $\mathbb{C} = \{1, 2, \cdots, n\}$ of multiple clients that are associated with a single server. For each $i \in \mathbb{C}$, there exists a neural network model with weights $\mathbf{w}_i$, which is cut into two segments at the k-th layer such that $\mathbf{w}_i = [\mathbf{w}_{c,i}, \mathbf{w}_s]^{\mathrm{T}}$ where the superscript $(\cdot)^{\mathrm{T}}$ implies the transpose operation, and $\mathbf{w}_i^k$ hereafter denotes the cut-layer weights connecting $\mathbf{w}_{c,i}$ and $\mathbf{w}_s$. The lower segment $\mathbf{w}_{c,i}$ is stored at the i-th client, and the upper segment $\mathbf{w}_s$ is shared by all clients. The i-th client has a local dataset $\mathbb{D}_i$ comprising unlabeled data $\mathbf{x}_{i,j}$ and their one-hot encoded ground-truth labels $\mathbf{y}_{i,j}$ for $j \in \{1, 2, \cdots, m_i\}$ where $m_i = |\mathbb{D}_i|$.

2.1 Sequential SL

In sequential SL which is a basic form of SL, only one client is randomly selected from all clients. Then, the selected i-th client generates a random batch $\mathbb{B}_i \subset \mathbb{D}_i$, consisting of b input-label tuples $(\mathbf{x}_{i,j}, \mathbf{y}_{i,j})$, from its own data set $\mathbb{D}_i$. Here, we define f as a representation for mapping from the input data to the smashed data by the lower model segment. To update the model, the i-th client produces b smashed data $\mathbf{s}_{i,j} := f_{\mathbf{w}_{c,i}}(\mathbf{x}_{i,j})$ by passing the j-th input data $\mathbf{x}_{i,j}$ through its own lower model with weight $\mathbf{w}_{c,i}$ for all $j \in \mathbb{B}_i$. Next, the client uploads the smashed data-label tuples to the server, and the server propagates it through the upper model segment $\mathbf{w}_s$ to produce *softmax output* $f_{\mathbf{w}_s}(\mathbf{s}_{i,j})$. By using the cross-entropy $CE(p, q) = -\sum q \log p$, the loss L_i for the i-th client is given as

$$L_i = \frac{1}{b} \sum_{j \in \mathbb{B}_i} \mathrm{CE}\big(f_{\mathbf{w}_s}(\mathbf{s}_{i,j}), \mathbf{y}_{i,j}\big) . \tag{1}$$

Next, the server generates the gradient of the upper model segment $\nabla_{\mathbf{w}_s} L_i$ through the i-th loss, sends the gradient of the cut-layer $\nabla_{\mathbf{w}_i^k} L_i$ to the i-th client, and the i-th client generates the gradient

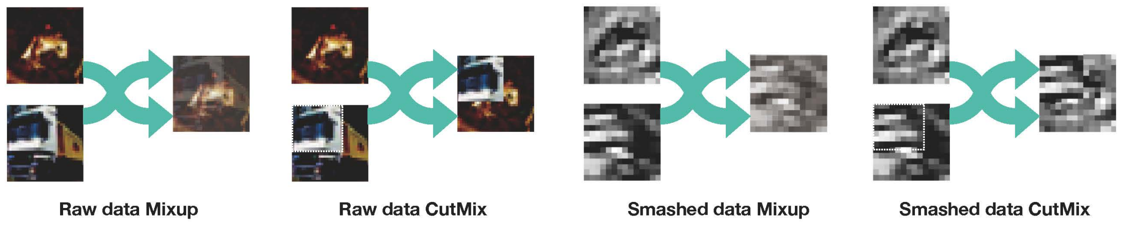

Figure 3: Mixup and CutMix examples for raw and smashed data.

of the lower model segment $\nabla_{\mathbf{w}_{c,i}} L_i$. Finally, the weight update of upper and lower model segment is performed in the server and the i-th client, respectively. After that, the lower model segment $\mathbf{w}_{c,i}$ is sent to the i'-th client, where $i' \in \mathbb{C} - \{i\}$. The update of lower model segments between the i-th client and the $i+1$-th client are as follows:

$$\begin{bmatrix} \mathbf{w}_s \\ \mathbf{w}_{c,i} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{w}_s \\ \mathbf{w}_{c,i} \end{bmatrix} - \eta \begin{bmatrix} \nabla_{\mathbf{w}_s} L_i \\ \nabla_{\mathbf{w}_{c,i}} L_i \end{bmatrix}, \tag{2}$$

where η is a learning rate.

$$\mathbf{w}_{c,i+1} \leftarrow \mathbf{w}_{c,i} \quad \forall i \in \mathbb{C} \backslash \{n\} \tag{3}$$

and $\mathbf{w}_{c,1} \leftarrow \mathbf{w}_{c,n}$.

Such a sequential SL is suitable for generating a high-accuracy model while reducing the client-side computation cost due to the separation of upper and lower model segments. However, due to the sequential nature of the training, where the client and server are connected in a 1-by-1 manner, as the number of clients participating in learning increases, the latency proportionally increases. This motivates parallel SL research that allows simultaneous access of multiple clients.

2.2 Parallel SL

① **Smashed Data FP.** In parallel training as shown in [7, 26], all clients connect to the server at the same time. Therefore, the FP in sequential training is processed in all clients simultaneously. As in (1), the server generates a loss L_i corresponding to the i-th client's smashed data, thereby completing FP in parallel.
② **Global Gradient BP.** For all $i \in \mathbb{C}$ and $j \in \mathbb{B}_i$, the server generates a loss L_i for each i-th client and sends the i-th gradient $\nabla_{\mathbf{w}_i^k} L_i$ to its corresponding client. Then, for all $i \in \mathbb{C}$, the i-th client and the server update the lower model segment $\mathbf{w}_{c,i}$ and the upper model segment $\mathbf{w}_s$ in the following ways:

$$\begin{bmatrix} \mathbf{w}_s \\ \mathbf{w}_{c,i} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{w}_s \\ \mathbf{w}_{c,i} \end{bmatrix} - \eta \begin{bmatrix} \sum_{i \in \mathbb{C}} (\delta_i \nabla_{\mathbf{w}_s} L_i) \\ \nabla_{\mathbf{w}_{c,i}} L_i \end{bmatrix}, \tag{4}$$

where δ_i denotes the weight of the i-th lower model segment determined by its dataset size, and is equal to $\delta_i = \frac{|\mathbb{D}_i|}{\sum_{i \in \mathbb{C}} |\mathbb{D}_i|}$. Note that the cut-layer gradient BP method of sequential SL is a special case in which the i-th weight δ_i is 1 and the remainder is 0 in the global gradient BP method of parallel SL.

2.3 Server-Client Update Imbalance in Parallel SL

By allowing multiple accesses, parallel training solves the latency problem of sequential training. However, in parallel training, while the client's lower model segment calculates gradient with a single batch of input data, the server's upper model segment calculates a global gradient by averaging the gradients using aggregated smashed data from multiple clients. This can be clearly seen when comparing Equations (2) and (4). In (2), $\mathbf{w}_s$ is updated for the loss of a single client, whereas in (4), the loss of multiple clients is reflected to update on $\mathbf{w}_s$. This implies that the *effective learning rate* of upper model segment has the effect of an n-fold increase from its original value, and this gap widens as the number of participating clients n increases. This yields the imbalance on server-client update may induce performance degradation in terms of convergence speed or accuracy and necessitate techniques to overcome them.

In summary, the parallel SL allows to break through the limit of sequential SL with regards to latency. However, parallel SL has a critical 'update imbalance problem' between server and clients as discussed in section 1. This highlights the need for additional techniques to address this problem.

3 LOCFEDMIX-SL: PARALLEL SL WITH LOCAL REGULARIZATION, LOCAL FEDERATED LEARNING, AND SMASHED DATA MIXUP

In this section, we describe the key component techniques used in the existing SL algorithms, and other component techniques in the DNN algorithms to target aforementioned imbalanced update problem with little or no additional cost. The list of component techniques to be covered in this section is as follows: 1) smashed data augmentation, 2) regularized local loss, and 3) lower model aggregation. Detailed design elements of each component technique are provided along with analysis.

3.1 Smashed Data Augmentation

The core idea of this subsection comes from how to improve accuracy and guarantee scalability without additional communication cost in the parallel SL structure. We consider the manifold mixup proposed in [29], using the smashed data aggregated to the server through the existing FP process, as a key to solve this problem.
③ **Smashed Data Mixup.** For the sake of convenience, we first assume a scenario in which only two clients (i-th and i'-th client) exist. When all smashed data and its corresponding ground-truth

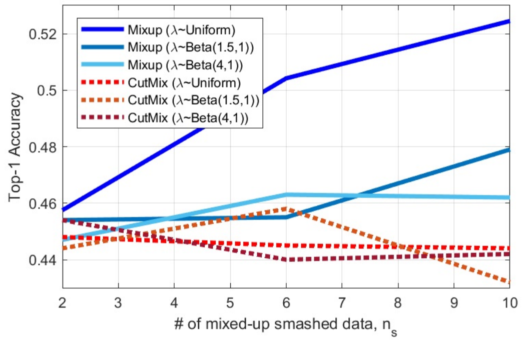

Figure 4: Top-1 accuracy w.r.t the number of mixed-up smashed data n_s and hyperparameters for Beta distribution in Mixup and CutMix algorithms.

Table 1: Top-1 accuracy w.r.t n and n_s in Mixup algorithm with $\lambda \sim [0, 1]$.

Methods	# of mixed-up smashed data, n_s				
	2	6	10	14	18
Mixup (n=10)	0.4575	0.5042	**0.5245**	-	-
Mixup (n=20)	0.4733	0.5335	0.5400	**0.5422**	0.5162

label are uploaded to the server, the server generates *mixed-up smashed data* by mixing the j-th smashed data of the i-th client and the j'-th smashed data of the i'-th client as follows:

$$\mathbf{s}_{i,j}^{i',j'} = \lambda \cdot \mathbf{s}_{i,j} + (1-\lambda) \cdot \mathbf{s}_{i',j'}, \tag{5}$$

where the mixing ratio λ follows the beta distribution ($\lambda \sim B(\alpha, \beta)$), which is equal to a uniform distribution $U[0, 1]$ if $\alpha = \beta = 1$.

The server propagates the mixed-up smashed data through the upper model segment, and produces the loss expressed as $\tilde{L}_{i,j}^{i',j'} = CE(\mathbf{s}_{i,j}^{i',j'}, y_{i,j}^{i',j'})$ where $y_{i,j}^{i',j'} = \lambda \cdot y_{i,j} + (1-\lambda) \cdot y_{i',j'}$. In the scenario with two clients, the i-th loss is expressed as $\tilde{L}_i^{i'} = 1/b \cdot \sum_{j \in \mathbb{B}_i, j' \in \mathbb{B}_{i'}} \tilde{L}_{i,j}^{i',j'}$. Note that the gradient obtained from $\tilde{L}_i^{i'}$ is detached from the lower model segment and thereby only flows to the upper model segment.

Generalizing this, consider $n > 2$ smashed data uploaded to the server. Single smashed data is 1-to-1 mixed-up with the one other smashed data, and the 1-to-1 mixup operation is repeated $n_s \leq n-1$ times per one smashed data. Consequently, each client's smashed data can generate up to n gradients, i.e., $n-1$ from smashed data mixup and 1 from the original smashed data. Let $\mathbb{C}_i^{n_s}$ denote a subset obtained by sampling n_s elements from $\mathbb{C} - \{i\}$ without replacement, in which $\mathbb{C} - \{i\}$ is a special case of $\mathbb{C}_i^{n_s}$ when $n_s = n-1$. When $n_s + 1$ out of n gradients are used to calculate the i-th loss, denoted by $\tilde{L}_i = \sum_{i' \in \mathbb{D}_i^{n_s}} \tilde{L}_i^{i'}$, $\mathbf{w}_s$ and $\mathbf{w}_{c,i}$ are updated by the server and the i-th client, respectively, as follows:

$$\begin{bmatrix} \mathbf{w}_s \\ \mathbf{w}_{c,i} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{w}_s \\ \mathbf{w}_{c,i} \end{bmatrix} - \eta \begin{bmatrix} \sum_{i \in \mathbb{C}} (\delta_i \nabla_{\mathbf{w}_s} (\tilde{L}_i + L_i)) \\ \nabla_{\mathbf{w}_{c,i}} L_i \end{bmatrix}. \tag{6}$$

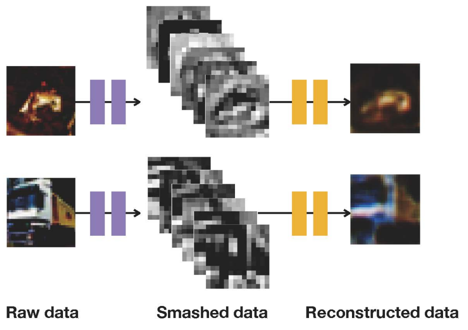

Figure 5: Illustrations of Infopro operations.

1) Full vs. partial mixup: Mixup data augmentation has several variants among which CutMix is one popular method. In CutMix, patches are cut and pasted among training images while the ground truth labels are mixed proportionally to the area of the patches. When CutMix is applied, (5) is changed to the following:

$$\tilde{\mathbf{s}}_{i,j}^{i',j'} = \mathbb{M}_\lambda \odot \mathbf{s}_{i,j} + (\mathbb{I} - \mathbb{M}_\lambda) \odot \mathbf{s}_{i',j'}, \tag{7}$$

where both $\mathbb{M}_\lambda$ and $\mathbb{I}$ are of the same size as $\mathbf{s}_{i,j}$, $\mathbb{M}_\lambda$ is a matrix with $|\mathbf{s}_{i,j}| \cdot b$ random elements 1 and remainder 0, $\mathbb{I}$ is a matrix in which all elements have a value of 1, and $\odot$ denotes element-wise multiplication operation. Fig. 4 compares the top-1 accuracy according to the Mixup and CutMix, along with the distribution of the mixing ratio λ, and the number of mixed-up smashed data n_s. Overall, the accuracy of Mixup is higher than that of CutMix, especially when λ follows a uniform distribution ($\alpha = \beta = 1$).

2) Impact of mixing weights: In Fig. 4, CutMix has fluctuations in the overall accuracy, so it does not show a specific tendency according to the distribution of λ. On the other hand, in Mixup, the performance is high in the order of uniform distribution, beta distribution with $\alpha = 1.5$, and beta distribution with $\alpha = 4$ in terms of accuracy.

3) Impact of n_s: Both Fig. 4 and Table 1 show that the accuracy is maximized when n_s is 10 if $n = 10$, and n_s is 14 if $n = 20$ on a mixup with a uniform λ. That is, accuracy is not always an incremental function for n_s. This is because increasing n_s has a similar effect to increasing batch size. As shown in [16], when the batch size increases, accuracy has either an increasing curve or a concave curve. Nevertheless, thanks to its augmentation effect, the larger n_s is, the higher the probability of achieving higher accuracy. In terms of scalability, larger n means that the range of n_s that can be searched is widened, which leads to a potential improvement in accuracy. From here, the optimal n_s according to n is left as future work, and the mixup technique with uniform λ is used with $n_s \propto n$. Recalling the batch increasing effect of n_s, this may be a key solution to solve the server-client update imbalance problem that occurs in parallel SL by providing an increased batch of smashed data according to the increased effective learning rate of the server. It is also noted that exchanging the smashed data of the cut-layer is heavily dependent on where the cut-layer is located. In general, the closer the cut-layer is to the input layer, the higher the performance gain when exchanging it.

Table 2: Top-1 Accuracy of three local parallel algorithms.

Local Regularization Techniques	Top-1 Accuracy
Infopro Regularization	**0.496**
Norm-based Regularization	0.465
Without Regularization	0.474

3.2 Local Loss with Mutual Information Regularization

The purpose of utilizing local loss for SL can be roughly divided into two categories: 1) (as in LocSplitFed) to update the lower model if the gradient is not transmitted or the transmission fails, and 2) (as in [30]) to control the amount of information contained in smashed data. In this subsection, we focus on existing algorithms centered on the second purpose. Hereafter, let h denote the mapping from the smashed data to the *reconstructed data* by the auxiliary network with weight $\hat{\mathbf{w}}_{c,i}$. For $i \in \mathbb{C}$, the i-th client has local model segment, where weight is represented by $\mathbf{w}_{d,i} = [\mathbf{w}_{c,i}, \hat{\mathbf{w}}_{c,i}]^{\mathrm{T}}$.

④ **Regularized Local Gradient.** The goal of local regularizer is to maximize the information about the input data that can be obtained from given smashed data, that is, the *mutual information* between the smashed data and the input data denoted by $I(\mathbf{x}_{i,j}|\mathbf{s}_{i,j}) = H(\mathbf{x}_{i,j}) - H(\mathbf{x}_{i,j}|\mathbf{s}_{i,j})$ [17]. To maximize the mutual information I, the *residual randonness* $H(\mathbf{x}_{i,j}|\mathbf{s}_{i,j})$ should be minimized. Here, the auxiliary network aims to minimize this residual randomness through its local loss $\hat{\mathbf{L}}_i$. By utilizing both local and global loss, the i-th client updates its local model segment with weight $\mathbf{w}_{d,i}$ as follows:

$$\begin{bmatrix}\hat{\mathbf{w}}_{c,i}\\ \mathbf{w}_{c,i}\end{bmatrix} \leftarrow \begin{bmatrix}\hat{\mathbf{w}}_{c,i}\\ \mathbf{w}_{c,i}\end{bmatrix} - \eta \begin{bmatrix}\nabla_{\hat{\mathbf{w}}_{c,i}}(\hat{L}_i)\\ \nabla_{\mathbf{w}_{c,i}}(\hat{L}_i + L_i)\end{bmatrix}. \tag{8}$$

Then, the i-th client uploads the auxiliary network and the server produces the *global auxiliary network* $\hat{\mathbf{w}}_c$ by taking weighted averaging on the aggregated auxiliary networks to be downloaded to the i-th client, as following formula:

$$\hat{\mathbf{w}}_{c,i} \leftarrow \hat{\mathbf{w}}_c = \sum_{i\in\mathbb{C}} (\delta_i \hat{\mathbf{w}}_{c,i}). \tag{9}$$

1) Infopro vs. norm-based regularization: To maximize of the mutual information, the local regularizers in Infopro and norm-based regularization to be minimized are given as follows:

$$\hat{L}_i = \begin{cases} ||\mathbf{x}_{i,j} - h(\mathbf{s}_{i,j})||^2 : \text{Infopro} \\ \theta \cdot \left| \left\|\mathbf{x}_{i,j}\right\| - \left\|\mathbf{s}_{i,j}\right\| \right| : \text{Norm-based,} \end{cases}$$

where $\|\cdot\|$ denotes the L2-norm function and θ is a hyperparameter controlling a ratio of local gradient. Table 2 compares the accuracy of the two regularization techniques and the standalone technique. Among them, the accuracy of Infopro regularization is the highest, and from this point on, we adopt the Infopro algorithm as a regularizer. In Fig. 4, Examples of representation of input data, smashed data, and reconstructed data are shown when the Infopro regularization is applied.

2) Impact of cut-layer: As the location of the cut-layer changes, the size of the output smashed data also varies, which determines how well Infopro works. Generally, the closer the cut-layer is to the input layer, the larger the size of the smashed data and the smaller the difference with the input data is, so the mutual information with the input data tends to be relatively high.

3) Detachment problem: If a lower model segment is trained only using local gradients instead of global gradients propagated from the server, the lower model can be detached from the upper model segment the server. This isolates the lower model update from the upper model update incurring performance degradation in terms of global model accuracy. This emphasizes the importance of learning in harmony between the server and the client.

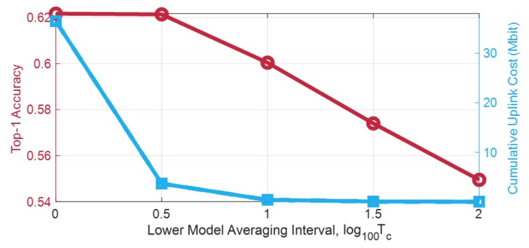

Figure 6: Top-1 accuracy and cumulative uplink cost (in Mbit) of local model federated averaging w.r.t lower model averaging interval T_c. The uplink cost only counts for the communication payload size of model parameters.

3.3 Local Model Federated Averaging

After updating the lower model and upper model through FP and BP, an additional aggregation phase is introduced to supplement the lower model updated with only local gradient compared to the upper model updated with global gradient.

⑤ **Local Federated Averaging.** Each i-th client uploads the updated $\mathbf{w}_{c,i}$ back to the server. The server produces a global lower model $\mathbf{w}_c$ through aggregated lower models and substitutes its own lower model segment with the global lower model with the following expression :

$$\mathbf{w}_{c,i} \leftarrow \mathbf{w}_c = \sum_{i\in\mathbb{C}} (\delta_i \mathbf{w}_{c,i}). \tag{10}$$

1) Period of Local Averaging: It is well known that the more frequent weight averaging is, the higher the test accuracy tends to be [35]. However, by increasing the weight averaging interval, the overhead caused by model aggregation can be reduced. Fig. 6 shows the **accuracy and communication cost trade-off** according to the local model averaging interval T_c. With the auxiliary network of section 3.2, the performance gain in terms of communication cost become larger when increasing the communication period. As with other subsection techniques, the change in communication cost and test accuracy is greatly affected by the location of the cut-layer and furthermore the structure of the entire model. Although its performance enhancement in accuracy is large enough with the help of FL, still this does not solve the detachment problem, since the lower model is updated independently with the server model.

3.4 Proposed: LocFedMix-SL

As depicted in Fig. 7, the operation of our proposed LocFedMix-SL, including all mentioned processes ① to ⑤, can be simplified in the following 3 steps: i) forward propagation with smashed data

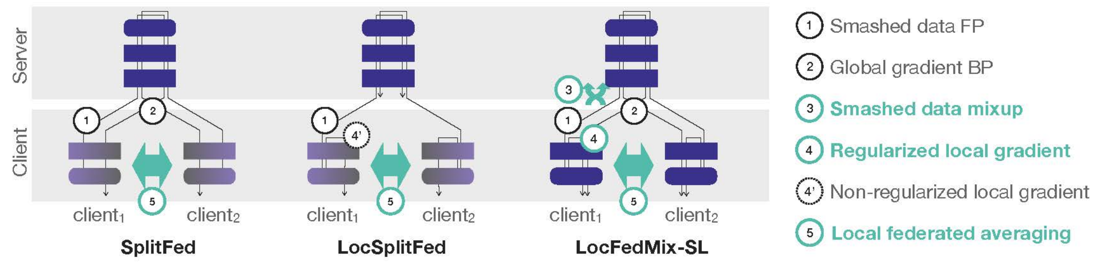

Figure 7: An illustration of the overall operation of LocFedMix-SL.

Table 3: Top-1 accuracy, total communication round of combinations for multiple techniques in SL.

Techniques	Top-1 Accuracy					Total Comm. Round					Remarks
	$n = 20$	40	60	80	100	$n = 20$	40	60	80	100	
1: ①	0.5946	0.6050	0.6171	0.6225	0.6295	9,770	9,680	9,600	9,760	9,390	Vanilla SL
2: ①+②+③	0.4988	0.4971	0.5402	0.5433	0.5551	2,970	8,900	8,890	7,990	7,950	-
3: ①+②+④	0.4751	0.4818	0.4788	0.4961	0.4797	2,190	2,170	2,130	1,940	2,470	-
4: ①+②+⑤	0.6214	0.6257	0.6229	0.6208	0.6196	6,270	6,090	6,360	6,130	6,400	SplitFed
5: ①+②+③+④	0.5152	0.5421	0.5568	0.5592	0.5687	8,940	9,230	8,980	9,090	9,820	-
6: ①+②+③+⑤	0.6292	0.6398	0.6398	0.6688	0.6698	3,750	2,460	2,300	2,020	1,840	-
7: ①+②+④’+⑤	0.6348	0.6376	0.6362	0.6356	0.6381	5,520	5,670	8,160	7,240	6,860	LocSplitFed + ②
8: ①+②+③+④+⑤	0.6365	0.6409	0.6591	0.6784	0.6810	3,590	2,470	2,060	1,340	1,960	LocFedMix-SL

mixup, ii) back propagation with local and global gradients, and iii) periodic local model aggregation.

Step 1 (①+③): For all $i \in \mathbb{C}$, the i-th client propagates b input data $\mathbf{x}_{i,j}$ through its lower model segment with weight $\mathbf{w}_{c,i}$ to produce b smashed data $\mathbf{s}_{i,j}$. Then, the client passes the smashed data through its auxiliary network $\hat{\mathbf{w}}_{c,i}$ to generate $\hat{L}_i$, while sending it to the server at the same time. The server mixes $\mathbf{s}_{i,j}$ with $\mathbf{s}_{i',j'}$ n_s times where $i' \in \mathbb{D}_i^{n_s}$, and obtains global gradient from $\tilde{L}_i + L_i$ using the upper model segment.

Step 2 (②+④): The server updates $\mathbf{w}_s$ using the aggregated global gradient from $\sum_i \delta_i(\tilde{L}_i + L_i)$. Then, the server transmits the cut-layer gradient $\nabla_{\mathbf{w}_i^k} L_i$ to the i-th client, and the client updates $\mathbf{w}_{c,i}$ and $\hat{\mathbf{w}}_{c,i}$ using the gradient flowing from both the global & local gradient $\nabla_{\mathbf{w}_{c,i}}(\hat{L}_i + L_i)$ and the gradient flowing from the local gradient $\nabla_{\hat{\mathbf{w}}_{c,i}}\hat{L}_i$, respectively.

Step 3 (⑤): The i-th client sends its local model $\mathbf{w}_{d,i}$ to the server, for all $i \in \mathbb{C}$. The server takes weighted averaging on the aggregated local model, and produces a global local model $\mathbf{w}_d$ downloaded by all clients. The i-th client replaces $\mathbf{w}_{d,i}$ with $\mathbf{w}_d$. This completes a single communication round of LocFedMix-SL. The detailed operation of our proposed algorithm is described in Appendix F.

4 NUMERICAL EXPERIMENTS

Experimental Settings. By combining ① to ⑤ techniques described in section 3 appropriately, we design 8 SL frameworks shown in Table 3 including our proposed LocFedMix-SL, and other state-of-the-art SL algorithms. In Table 3, we compare the performance of the above algorithms in terms of top-1 accuracy and total communication round. Both Table 4 and Fig. 8 measure the latency of the algorithms while changing the ratio of computing and communication latency. Fig. 9 of Appendix B shows the learning curve w.r.t training time which simultaneously shows the test accuracy, convergence time, and latency.

In Table 3, the LeNet5 [15] is used for the CIFAR-10 dataset [13]. Fig. 9 additionally uses the fashion-MNIST dataset [31]. Regarding the model split, the cut-layer of LeNet-5 is located after the first convolutional layer and max-pooling layer. For data split, in the case of CIFAR-10, each device randomly samples $5,000$ training data out of a total of 50,000 training data sets and, in the case of fashion-MNIST, randomly holds $12,000$ training data out of a total of 60,000 training data sets.

Regarding the structure of the auxiliary network, the interpolate function is first used to compensate for the small size of smashed data compared to input data, and then it passes through a decoder that includes two convolutional layers, one batch normalization layer, one ReLU function, and one sigmoid function. Finally, the mean squared error (MSE) function is used to measure the loss between the reconstructed data and the input data. Other simulation parameters are given as follows: batch size $b = 64$, learning rate $\eta = 0.004$, and # of communication rounds $T = 10,000$.

Scalability & Convergence Speed. Table 3 reports the top-1 accuracy and total communication rounds for several combinations of different techniques. The proposed LocFedMix-SL (algorithm 8) achieves the highest accuracy regardless of n and always records the top performance in terms of convergence speed. Even when $n = 60$ and 80, its convergence speed is the fastest. Next, Table 3 reveals that scalability is guaranteed under the algorithm 1 (i.e., Vanilla SL) as

Table 4: Latency comparison during T communication rounds of split learning algorithms with n clients.

Methods	Latency for T communication rounds
Vanilla SL	$(T_{comp} + nT_{comm}) \cdot nT$
SplitFed	$(T_{comp} + nT_{comm} \cdot \frac{\lvert\mathbf{s}\rvert+\lvert\mathbf{w_c}\rvert}{\lvert\mathbf{s}\rvert}) \cdot T$
LocSplitFed	$(T_{comp} + max(T_{comp}, nT_{comm}) + nT_{comm} \cdot \frac{\lvert\hat{\mathbf{w}}_c\rvert+\lvert\mathbf{w}_c\rvert}{\lvert\mathbf{w}_c\rvert}) \cdot T$
Proposed	$(T_{comp} + nT_{comm}) \cdot T + nT_{comm} \cdot \frac{\lvert\hat{\mathbf{w}}_c\rvert+\lvert\mathbf{w}_c\rvert}{\lvert\mathbf{w}_c\rvert} \cdot \lfloor \frac{T}{T_c} \rfloor$

well as under the new algorithms 2, 5, 6, and 8 (i.e., LocFedMix-SL). The common feature of the algorithms that guarantee scalability except for Vanilla SL is the use of technique ③, smashed data mixup. In the case of techniques such as lower model averaging and local parallel training used by existing parallel SL algorithms, the update imbalance problem between server-client is not resolved, and the detachment problem is rather caused by updating independently of the server's upper model leading to a decrease in accuracy. As n increases, the imbalance and detachment problems become severe, and the accuracy gradually decreases accordingly.

Smashed data mixup resolves the update imbalance problem between server-client by providing a large amount of augmented samples to the server with a large effective learning rate. As n increases and the effective learning rate of the server increases, the number of mixed-up smashed data $n_s < n$ also increases and is supplemented, thereby ensuring scalability. Furthermore, smashed data mixup avoids the detachement problem in a sense that each local model is not separately trained as in local weight averaging or local regularization techniques. The effectiveness of smashed data mixup is bolstered by ④, local loss regularization, as shown by comparing the algorithms 2,5 with 6,8 in Table 3. As mentioned, it is known that the accuracy gain tends to increase as the data mixup is performed in a layer closer to the input layer. Local regularization learns in the direction of maximizing the amount of information about the input data that the smashed data has, which may have a similar effect to making the smashed data mixup closer to the input data mixup. In this context, simultaneous utilization of smashed data mixup and local regularizer leads to greater accuracy gain.

Similarly, from Table 3, the combination of technology ③ and ⑤ has good performance in terms of convergence speed. As mentioned above, smashed data mixup has the same effect as increasing the batch size of the server, which speeds up the convergence speed of the upper model of the server. However, with the smashed data mixup technique alone, the convergence speed of the client's lower model can be bound, and this can be confirmed through the algorithm 2 in the table. Although local weight averaging can reduce scalability by inducing a detachment problem, it is an optimal technique to increase the convergence speed of the lower model. Therefore, as the smashed data mixup and local weight averaging are used together, the convergence speed of the lower model and the upper model is simultaneously improved, which speeds up the convergence speed of the entire model.

Communication-Computing Latency. Table 4 compares SL algorithms' latency for T communication rounds with n clients. The latency measures the time it takes for all n clients to learn for each T communication round. Under the premise that the server's computing power and transmission power are very large, only computing and communication latency occurring at the client-side is considered while ignoring the computing and communication latency occurring on the server-side. Also, it is assumed that all clients share the entire band equally. Within a single communication round, T_{comm} denotes the uplink transmission latency under the assumption that a client utilizes full bandwidth, and T_{comp} denotes the computational latency. In addition, $|\mathbf{s}|$, $|\mathbf{w}_c|$, $|\hat{\mathbf{w}}_c|$ denote the size of b smashed data, lower model segment, auxiliary network, respectively. T_c is the upper layer average interval in our method. Accordingly, in terms of latency, $T_c = 1$ is regarded as the worst case, whereas $T_c = T$ is regarded as the best case.

In Appendix A, Fig. 8 (top) illustrates the latency w.r.t the number of clients n when the ratio of T_{comp} and T_{comm} is 10:1. The best case of our proposed algorithms achieves the lowest latency, and the worst case of our proposed methods is still better than all other baselines except for SplitFed. This latency gain decreases when the communication delays dominate the overall latency (i.e., $T_{comm} \geq T_{comp}$). For such scenarios, the best case of our proposed method still achieves the lowest latency, and our worst case latency is on par with LocSplitFed, as shown in Figs. 8 (middle & bottom).

Impacts of Local Parallelism, Federated Learning, and Mixup. According to Table 3, smashed data mixup contributes to improving convergence speed and to achieving scalability in tandem with lower model averaging. Smashed data mixup also creates a synergetic effect in improving accuracy, in combination with local regularization that does not provide any performance gain alone. Local model averaging contributes to the improved convergence speed and accuracy. In Appendix B, Fig. 9 compares the learning curves of different algorithms over time. The results reveals that the proposed LocFedMix-SL achieves the highest accuracy, fastest convergence, and lowest latency by carefully integrating local parallel training, federated learning, and Mixup.

5 CONCLUSION

Departing from sequential SL, we studied parallel SL having great potential in reducing latency. However, existing parallel SL algorithms are not scalable, and often struggle with slow convergence. We discovered that the reason comes fundamentally from the parallel SL architecture that is inherently prone to incur the problems of server-client update imbalance and client model detachment from the server model. To fix this issue, we carefully integrated local parallelism, federated learning, and mixup data augmentation techniques into parallel SL, so as to keep the FP flows and BP updates balanced. Consequently, we proposed a novel parallel SL framework, named LocFedMix-SL, and validated its achieving high scalability, fast convergence, and low latency by simulation. Extending this work, investigating the effectiveness of LocFedMix-SL in various datasets under imbalanced data distributions and different model architectures could be interesting topics for future work.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00347, 6G Post-MAC (POsitioning- & Spectrum-aware intelligenT MAC for Computing & Communication Convergence)), and by EU-CHISTERA project LeadingEdge and CONNECT.

REFERENCES

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

[2] Anis Elgabli, Jihong Park, Sabbir Ahmed, and Mehdi Bennis. 2020. L-FGADMM: Layer-wise federated group ADMM for communication efficient decentralized deep learning. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 1–6.

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

[4] Yansong Gao, Minki Kim, Chandra Thapa, Sharif Abuadbba, Zhi Zhang, Seyit A Camtepe, Hyoungshick Kim, and Surya Nepal. 2021. Evaluation and Optimization of Distributed Machine Learning Techniques for Internet of Things. *arXiv preprint arXiv:2103.02762* (2021).

[5] Otkrist Gupta and Ramesh Raskar. 2017. Secure Training of Multi-Party Deep Neural Network. US Patent App. 15/630,944.

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

[7] Dong-Jun Han, Hasnain Irshad Bhatti, Jungmoon Lee, and Jaekyun Moon. 2021. Accelerating Federated Learning with Split Learning on Locally Generated Losses. In *ICML 2021 Workshop on Federated Learning for User Privacy and Data Confidentiality*. ICML Board.

[8] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim. 2018. Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data. *presented at NeurIPS Wksp. MLPCD (Montréal, Canada). arXiv preprint arXiv:1811.11479* (Dec. 2018).

[9] Praveen Joshi, Chandra Thapa, Seyit Camtepe, Mohammed Hasanuzzamana, Ted Scully, and Haithem Afli. 2021. Splitfed learning without client-side synchronization: Analyzing client-side split network portion size to overall performance. *arXiv preprint arXiv:2109.09246* (2021).

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

[11] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. 2015. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575* (2015).

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

[13] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

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

[15] Yann LeCun et al. 2015. LeNet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet* 20, 5 (2015), 14.

[16] Linjian Ma, Gabe Montague, Jiayu Ye, Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael Mahoney. 2020. Inefficiency of k-fac for large batch size training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5053–5060.

[17] David JC MacKay and David JC Mac Kay. 2003. *Information theory, inference and learning algorithms*. Cambridge university press.

[18] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[19] Shraman Pal, Mansi Uniyal, Jihong Park, Praneeth Vepakomma, Ramesh Raskar, Mehdi Bennis, Moongu Jeon, and Jinho Choi. 2021. Server-Side Local Gradient Averaging and Learning Rate Acceleration for Scalable Split Learning. *to be presented at 2022 AAAI-FL Wksp. arXiv preprint arXiv:2112.05929* (2021).

[20] Jihong Park, Sumudu Samarakoon, Mehdi Bennis, and Mérouane Debbah. 2019. Wireless network intelligence at the edge. *Proc. IEEE* 107, 11 (2019), 2204–2239.

[21] Jihong Park, Sumudu Samarakoon, Anis Elgabli, Joongheon Kim, Mehdi Bennis, Seong-Lyun Kim, and Mérouane Debbah. 2021. Communication-Efficient and Distributed Learning Over Wireless Networks: Principles and Applications. *Proc. IEEE* 109, 5 (May 2021), 796–819. https://doi.org/10.1109/JPROC.2021.3055679

[22] Jihong Park, Shiqiang Wang, Anis Elgabli, Seungeun Oh, Eunjeong Jeong, Han Cha, Hyesung Kim, Seong-Lyun Kim, and Mehdi Bennis. 2019. Distilling on-device intelligence at the network edge. *arXiv preprint arXiv:1908.05895* (2019).

[23] Maarten G Poirot, Praneeth Vepakomma, Ken Chang, Jayashree Kalpathy-Cramer, Rajiv Gupta, and Ramesh Raskar. 2019. Split learning for collaborative deep learning in healthcare. *arXiv preprint arXiv:1912.12115* (2019).

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

[25] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. 2017. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489* (2017).

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

[27] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, and Seyit Ahmet Çamtepe. 2020. Advancements of federated learning towards privacy preservation: from federated learning to split learning. *CoRR* abs/2011.14818 (2020). arXiv:2011.14818 https://arxiv.org/abs/2011.14818

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

[29] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. 2019. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*. PMLR, 6438–6447.

[30] Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. 2021. Revisiting Locally Supervised Learning: an Alternative to End-to-end Training. *arXiv preprint arXiv:2101.10832* (2021).

[31] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv:cs.LG/1708.07747 [cs.LG]

[32] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.

[33] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. 2019. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6023–6032.

[34] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).

[35] Jian Zhang, Christopher De Sa, Ioannis Mitliagkas, and Christopher Ré. 2016. Parallel SGD: When does averaging help? *arXiv preprint arXiv:1606.07365* (2016).

A LATENCY UNDER DIFFERENT COMPUTING-COMMUNICATION COSTS

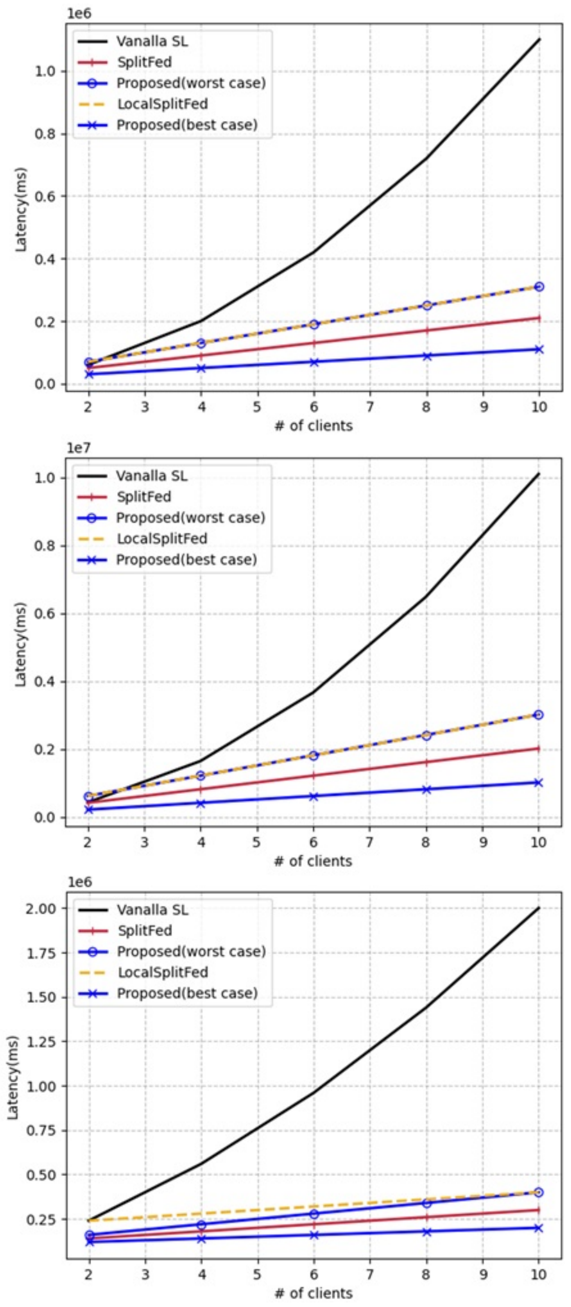

Figure 8: Latency of SL algorithms when $T_{comp} : T_{comm}$ is 10:1 (top), 1:1 (middle), and 1:10 (bottom), respectively.

B LEARNING CURVE UNDER DIFFERENT DATA SETS ($n = 100$)

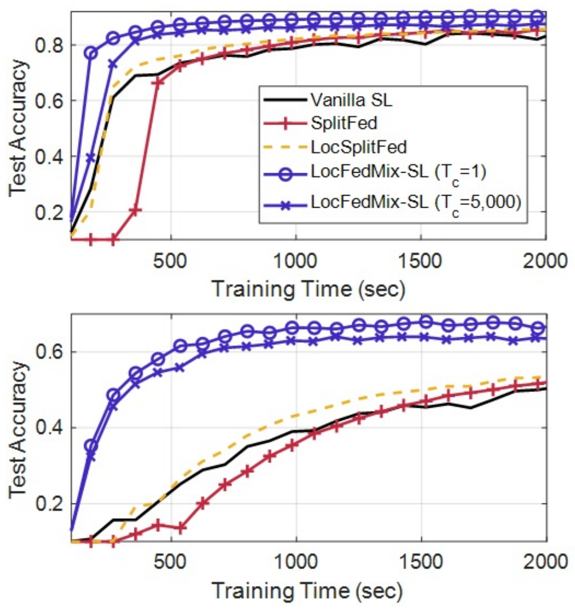

Figure 9: Learning curves of SL algorithms on CIFAR-10 and fashion-MNIST data sets.

C VANILLA SL PSEUDOCODE

Algorithm 1 Vanilla SL

```
1: requirements: $\mathbb{D}_i$ with $i \in \mathbb{C}$, $t = 0$
2: while $t < T$ do
3: $i$-th Client:
4:     generates $s_{i,j}$ by passing $x_{i,j}$ through $w_{c,i}$          ▹ FP
5:     unicasts $(s_{i,j}, y_{i,j})$ to the server          ▹ Uplink
6: Server:
7:     generates $L_i$ by using (1)
8:     updates $w_s$ via (2) with $\nabla_{w_s} L_i$          ▹ BP
9:     unicasts $\nabla_{w_i^k} L_i$ to the $i$-th client          ▹ Downlink
10: $i$-th Client:
11:     updates $w_{c,i}$ via (1) with $\nabla_{w_{c,i}} L_i$
12:     unicasts $w_{c,i}$ to the $i+1$-th client as (3) ▹ Model Transition
13: $t \leftarrow t + 1$
14: end while
```

D SPLITFED PSEUDOCODE

Algorithm 2 SplitFed

```
 1: requirements: $\mathbb{D}_i$ with $i \in \mathbb{C}$, $t = 0$
 2: while $t < T$ do
 3: Client $i \in \mathbb{C}$:
 4:     generates $\mathbf{s}_{i,j}$ by passing $\mathbf{x}_{i,j}$ through $\mathbf{w}_{c,i}$          ▷ FP
 5:     unicasts $(\mathbf{s}_{i,j}, \mathbf{y}_{i,j})$ to the server          ▷ Uplink
 6: Server:
 7:     generates $L_i$ by using (1)
 8:     updates $\mathbf{w}_s$ via (4) with $\sum_i \delta_i \nabla_{\mathbf{w}_s} L_i$          ▷ BP with Global Gradient
 9:     unicasts $\nabla_{\mathbf{w}_i^k} L_i$ to the $i$-th client for all $i \in \mathbb{C}$          ▷ Downlink
10: Client $i \in \mathbb{C}$:
11:     updates $\mathbf{w}_{c,i}$ via (4) with $\nabla_{\mathbf{w}_{c,i}} L_i$
12:     unicasts $\mathbf{w}_{c,i}$ to the server          ▷ Uplink
13: Server:
14:     calculates $\mathbf{w}_c$ via (10)          ▷ Lower Model Averaging
15:     unicasts $\mathbf{w}_c$ to the $i$-th client for all $i \in \mathbb{C}$          ▷ Downlink
16: Client $i \in \mathbb{C}$:
17:     replaces $\mathbf{w}_{c,i}$ with $\mathbf{w}_c$
18: $t \longleftarrow t + 1$
19: end while
```

E LOCSPLITFED PSEUDOCODE

Algorithm 3 LocSplitFed

```
 1: requirements: $\mathbb{D}_i$ with $i \in \mathbb{C}$, $t = 0$
 2: while $t < T$ do
 3: Client $i \in \mathbb{C}$:
 4:     generates $\mathbf{s}_{i,j}$ by passing $\mathbf{x}_{i,j}$ through $\mathbf{w}_{c,i}$          ▷ FP
 5:     unicasts $(\mathbf{s}_{i,j}, \mathbf{y}_{i,j})$ to the server          ▷ Uplink
 6:     generates $\hat{L}_i$ by passing $\mathbf{s}_{i,j}$ through $\hat{\mathbf{w}}_{c,i}$
 7:     updates $\mathbf{w}_{d,i}$ with $\hat{L}_i$          ▷ Local Gradient
 8: Server:
 9:     generates $L_i$ by using (1)
10:     updates $\mathbf{w}_s$ via (4) with $\sum_i \delta_i \nabla_{\mathbf{w}_s} L_i$          ▷ BP with Global Gradient
11: Client $i \in \mathbb{C}$:
12:     unicasts $\mathbf{w}_{d,i}$ to the server          ▷ Uplink
13: Server:
14:     calculates $\mathbf{w}_d$ via (9)&(10)          ▷ Local Model Averaging
15:     unicasts $\mathbf{w}_d$ to the $i$-th client for all $i \in \mathbb{C}$          ▷ Downlink
16: Client $i \in \mathbb{C}$:
17:     replaces $\mathbf{w}_{d,i}$ with $\mathbf{w}_d$
18: $t \longleftarrow t + 1$
19: end while
```

F LOCFEDMIX-SL PSEUDOCODE

Algorithm 4 LocFedMix-SL ($T_c = 1$)

```
 1: requirements: $\mathbb{D}_i$ with $i \in \mathbb{C}$, $t = 0$, $1 \le n_s \le n$
 2: while $t < T$ do
 3: Client $i \in \mathbb{C}$:
 4:     generates $\mathbf{s}_{i,j}$ by passing $\mathbf{x}_{i,j}$ through $\mathbf{w}_{c,i}$          ▷ FP
 5:     unicasts $(\mathbf{s}_{i,j}, \mathbf{y}_{i,j})$ to the server          ▷ Uplink
 6:     generates $\hat{L}_i$ by passing $\mathbf{s}_{i,j}$ though $\hat{\mathbf{w}}_{c,i}$
 7:     updates $\hat{\mathbf{w}}_{c,i}$ via (8) with $\hat{L}_i$          ▷ Local Gradient
 8: Server:
 9:     generates $L_i$ by using (1)
10:     generates $s_{i,j}^{i',j'}$ for all $i' \in \mathbb{D}_i^{n_s}$ via (5)          ▷ Smashed Data Mixup
11:     calculates $\tilde{L}_i$ using all $s_{i,j}^{i',j'}$
12:     updates $\mathbf{w}_s$ via (6) with $\sum_i \delta_i \nabla_{\mathbf{w}_s} (\tilde{L}_i + L_i)$          ▷ BP with Global Gradient
13:     unicasts $\nabla_{w_i^k} (\tilde{L}_i + L_i)$ to $i$-th client for all $i \in \mathbb{C}$          ▷ Downlink
14: Client $i \in \mathbb{C}$:
15:     updates $\mathbf{w}_{c,i}$ via (8) with $\nabla_{\mathbf{w}_{c,i}} (\tilde{L}_i + L_i)$
16:     unicasts $\mathbf{w}_{d,i}$ to the server          ▷ Uplink
17: Server:
18:     updates $\mathbf{w}_d$ via (9)&(10)          ▷ Local Model Averaging
19:     unicasts $\mathbf{w}_d$ to the $i$-th client for all $i \in \mathbb{C}$          ▷ Downlink
20: Client $i \in \mathbb{C}$:
21:     replaces $\mathbf{w}_{d,i}$ with $\mathbf{w}_d$
22: $t \longleftarrow t + 1$
23: end while
```

View publication stats