Harnessing the Power of Local Supervision in Federated Learning

Fei Wang, *Student Member, IEEE,* Baochun Li, *Fellow, IEEE*

Department of Electrical and Computer Engineering
University of Toronto

***Abstract*—Federated learning is widely accepted as a privacy-preserving paradigm for training a shared global model across multiple client devices in a collaborative fashion. However, in practice, the significantly limited computational power on client devices has been a major barrier when we wish to train large models with potentially hundreds of millions of parameters. In this paper, we propose a new architecture, referred to as INFOCOMM, that incorporates locally supervised learning in federated learning. With locally supervised learning, the disadvantages of split learning can be avoided by using a more flexible way to offload training from resource constrained clients to a more capable server. INFOCOMM enables parallel training of different modules of the neural network in both the server and clients in a gradient-isolated fashion. The efficacy in reducing both training time and communication time is supported by our theoretical analysis and empirical results. In the scenario involving larger models and fewer available local data, INFOCOMM has been observed to reduce the elapsed time per round by over 37% without sacrificing accuracy compared to both conventional federated learning or directly combining federated learning and split learning, which showcases the advantages of INFOCOMM under power-constrained IoT scenarios.**

***Index Terms*—Federated learning, locally supervised learning, split learning, large models, resource-constrained IoT devices**

I. INTRODUCTION

As ubiquitous Internet of Things (IoT) devices allow for the tremendous and continuous growth of data at the network edge, it becomes very appealing to integrate federated learning (FL) to utilize such data, while preserving data privacy and avoiding the transfer of large amounts of data [1]. As an emerging distributed learning paradigm, federated learning is a well-explored practice of performing machine learning model training on decentralized client devices without offloading data to centralized servers [2], [3].

Despite the privacy advantages of federated learning over centralized machine learning mechanisms, the integration of federated learning and edge computing is not as practical as many anticipated, due to extremely limited amounts of computational resources available on IoT devices [4]. Recent work has made significant strides towards mitigating this issue, by developing new weighted global aggregation mechanisms [5], new device sampling mechanisms [6], or by updating the global model asynchronously [7], [8]. These approaches attempted to avoid slow devices with limited computational resources, as low-quality updates from these slow devices may affect global convergence negatively. However, by restricting the participation of slow devices, such approaches also limit the utilization of data on a diverse range of IoT devices, and may result in wasted communication and computation efforts on these devices if they are excluded from participation.

Furthermore, the scarcity of computational power becomes more pronounced in IoT devices equipped with CPUs when compared to cloud servers that typically have multiple GPUs. An IoT device may even fail to accommodate a large neural network with the associated memory footprint during training, let alone training a model within an expected amount of time that can contribute local updates for the global model's convergence. Consequently, it becomes unrealistic to expect edge clients, with their limited computational capabilities, to handle all the intensive training work, while the more powerful server merely focuses on model aggregation in federated learning. Thus, a more equitable approach is required, with the hope of distributing the training workload to leverage the limited resources of edge clients and the abundant resources of the server.

As a promising way to tackle this issue, an intuitive idea is to split the full learning model into two modules and allow the client to train the first smaller module based on local data, while leaving the training of the second module to the server. Split learning [9] utilized this approach to perform distributed deep learning without sharing raw data. Further, SplitFed [10] combined the strengths of both federated learning and split learning so that multiple clients can engage with the server in parallel in resource-constrained environments. SplitFed involved separate training at both clients and the server, as well as model aggregation at the server. However, these mechanisms can incur significant delay and communication overhead compared to conventional federated learning. As clients must forward intermediate outputs (i.e., features) at the cut layer to the server, and then wait for the server to send its gradients back to the clients, an excessive amount of extra communication becomes unavoidable.

In this paper, we propose a novel architecture, referred to as INFOCOMM[1], that makes federated learning more applicable at resource-constrained client devices, and provides a more balanced approach to training machine learning models on these client devices. To leverage the benefits of model splitting without adding communication overhead and delays like SplitFed, INFOCOMM allows for more independent training

[1]INFOCOMM involves the integration of ***Info***Pro [11] for independent training in federated learning while minimizing the ***comm***unication overhead introduced compared to split learning.

of clients and the server using separated models. It involves generating backpropagation signals within each module of the neural network, rather than passing them through the entire neural network via a global loss.

More specifically, we incorporate a state-of-the-art locally supervised learning mechanism, called InfoPro [11], which allows clients to perform module training via locally generated losses without receiving gradients from the server for further backward passes. Such locally generated loss is critical for independent module training, enabling the learning of intermediate features that benefit both the local module at the clients and the latter module at the server. With this approach, the training workload can be offloaded from resource-constrained client devices to the server with more computational power in a highly independent manner, avoiding extra communication between clients and the server for network backpropagation, while ensuring the convergence of the global model.

Combining federated learning and locally supervised learning offers another key advantage: training at the server and clients is separate and can be concurrently executed, as the server can keep training upon receiving features from clients while clients perform a new round of local training. In INFOCOMM, we propose a unique feature aggregation algorithm specifically tailored for our architecture, which is not applicable to SplitFed. This algorithm allows the server to have access to a larger dataset and provides greater flexibility in training its module.

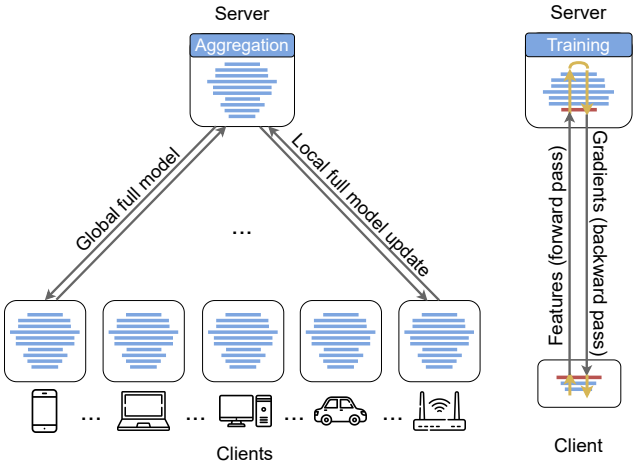
By utilizing model splitting and InfoPro loss, INFOCOMM addresses the computational power deficiency of IoT devices in conventional federated learning. With reduced costs of communication, INFOCOMM is more efficient than both conventional federated learning and SplitFed, which we show using both theoretical analysis and empirical results. We compare INFOCOMM with federated learning algorithms such as Federated Averaging (FedAvg) [3], Stochastic Controlled Averaging (SCAFFOLD) [12], and federated split learning (SplitFed) [10]. Our extensive experimental results in various scenarios with different datasets and neural networks demonstrate that our approach can significantly improve the speed of convergence, especially for the cases of non-IID data distribution, or small clients with fewer data samples. The advantage of INFOCOMM in terms of reducing both training time and communication time is more prominent when training with large models and limited amount of local data. In certain scenarios, INFOCOMM can achieve time savings of 28%, 54%, and 37% compared to FedAvg, SCAFFOLD, and SplitFed, respectively. When comparing the test accuracy of the global model at the point where INFOCOMM already converges, it can outperform FedAvg and SplitFed by achieving approximately $6.5\times$ and $1.2\times$ higher accuracy, respectively.

II. PRELIMINARIES AND MOTIVATIONS

In this section, we first introduce the basic workflows of conventional federated learning and split learning, and then highlight several important points that motivate our work of incorporating both model splitting and locally supervised learning into federated learning.

A. Preliminaries

Federated learning. Federated learning involves multiple rounds of interaction between clients and a central server to train a global model until convergence. Clients can be a wide range of devices, including laptops, smartphones, and IoT devices, which have different capabilities to participate in the training process. In each round of federated learning, as illustrated in Fig. 1a, the server sends the current global model to a subset of clients selected from a pool of participants. These selected clients then perform several epochs of training on the model with their local data and send the model updates back to the server. The server aggregates the model updates received from these clients and updates the global model accordingly. FedAvg [3] is considered a pioneer and standard federated learning algorithm, in which clients are randomly selected and their local updates are aggregated, with aggregation weights computed based on the number of local data samples used for local training. For our work, we focus solely on the FedAvg algorithm when comparing federated learning and other distributed learning approaches.



(a) Federated Learning (b) Split Learning

Fig. 1: Illustration of the interactions between clients and the server in federated learning and split learning.

Split learning. Split learning [9], [13] is another approach to training a neural network in a distributed manner across multiple clients without sharing raw data. Unlike federated learning, the server and clients have distinct roles, as demonstrated in Fig. 1b. In a typical split learning architecture, instead of training the entire model, a client only trains a portion of the network, up to a designated cut layer, based on its local data. The client sends the cut layer's output, *i.e.,* features, to the server, which then completes the forward propagation as well as the backward propagation from the last layer to the cut layer. After updating the weights, the server sends the gradients till the cut layer back to the client, who then completes the backpropagation.

Despite the separation of network modules between the server and clients, split learning still employs end-to-end training. The objective or loss function, which evaluates the

overall model's performance on the task, is calculated at the output layer of the network residing on the server. The gradients of the loss function need to be transmitted from the server to the client for backpropagation from the output layer to the input layer, in order to update the weights for the entire model.

When multiple clients from different entities are involved in split learning, a round-robin approach is employed, where each client engages in alternating epochs while collaborating with the server. Before starting the next training epoch, the next client in the training order is required to update its model weights, either from the server in a centralized mode or from the last trained client in a peer-to-peer mode, to synchronize the sub-model.

B. Motivation

Researchers investigating the application of federated learning for edge devices have underestimated the extent of the severe computational constraints associated with edge devices [1], [4], [8], [14]. Training a substantial language model, for example, on CPUs could take days, or it might not be feasible to run in practice at all. At first glance, split learning [9] seems to be a promising solution for reducing the training workload on the clients by transferring it to the server using model splitting [15], [16]. A recent work proposed SplitFed [10] that incorporated federated learning into split learning.

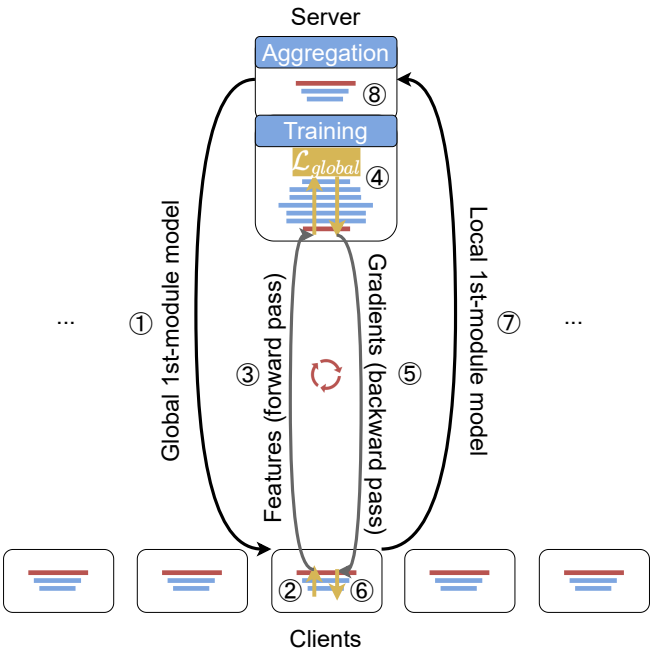


Fig. 2: SplitFed (SFLV2)

In SplitFed, all clients perform forward propagation on their 1st-module models in parallel and send their features at the cut layer to the server. The server is responsible for aggregating and updating the global 1st-module model using FedAvg aggregation, as well as training the 2nd-module model. When it comes to 2nd-module training, SplitFed offers two different approaches: SFLV1 and SFLV2. In SFLV1, the server maintains a separate 2nd-module model for each participating client and trains them in parallel using the corresponding client's features during each communication round. Once all the training is completed, the SFLV1 server performs the FedAvg aggregation to obtain a global 2nd-module model. In contrast, in SFLV2, the server maintains a single 2nd-module model and performs sequential forward and backward propagations with features from different clients. The 2nd-module model in SFLV2 is basically trained in a centralized manner with no aggregation and may achieve higher model accuracy compared with SFLV1. In our study, we do not consider SFLV1 due to its high computational requirements and lack of scalability, especially when dealing with a large number of selected clients and when a large portion of a complex model resides on the server. Interactions between clients and servers in SFLV2 are illustrated in Fig. 2.

Other potential hybrid formats that combine federated learning and split learning have been explored in previous research. In the architecture proposed by [17], multiple servers are employed, and each client-server pair conducts parallel training of their 1st-module and 2nd-module models. A more generalized approach, presented in [18], introduces a hierarchical architecture where multiple servers are organized into different levels at the edge. Comparative studies were carried out in [19], [20] to assess the learning performance and training costs of split learning, federated learning, and the hybrid frameworks in diverse IoT scenarios.

While SplitFed offers potential benefits for resource-constrained clients where full model training and deployment may not be feasible, it's important to note that there are several fundamental drawbacks in combining federated learning and split learning that may render it impractical.

SplitFed introduces a substantial amount of communication overhead into federated learning. At first glance, sending features extracted from the cut layer instead of entire model updates from clients to the server seems to have reduced the upload communication overhead. However, model updates are sent only after each round of local training, whereas features are sent in every iteration of stochastic gradient descent. This implies that SplitFed involves substantially more communication rounds between clients and the server in comparison to federated learning. Furthermore, the total size of the features is directly affected by the number of data samples in the batch. With a large batch size, the size of features can become significant and potentially exceed that of the entire model. In addition to the upload communication overhead, the server also needs to send gradients back to the clients in every iteration of training. This requirement significantly amplifies the download communication overhead.

Sequential training at the SplitFed server introduces inherent delays that cannot be avoided. In SplitFed (SFLV2), the features and gradients are tied to each batch of local data on a specific client. The server is required to wait until it completes the backpropagation process for the current set of features before it can proceed with forwarding the next set of features. Essentially, the server updates the 2nd-

module parameters *sequentially* with respect to the features from different clients rather than performing aggregation. In conventional federated learning, one communication round typically consists of multiple epochs and batches of local training. However, when split learning is combined, clients and the server must exchange features and gradients for forward and backward propagations to proceed *within each iteration of training*. Consequently, delays arise as each party waits repeatedly for the other to complete local forward or backward propagation and transmit the required information over the network.

These limitations have motivated us to search for a more appropriate solution to accommodate split models for federated learning, leading us to locally supervised learning. Locally supervised learning involves training a network model that is split into multiple modules. Unlike in traditional end-to-end training, these successive modules are not associated with gradients from each other and are trained with local supervision only. To train each module, features extracted from the last layer of the previous module are taken as input (raw data samples are still used for the first module), and a locally generated loss function (task-based loss for the last module still) is used for local backpropagation. The locally generated loss is responsible for backpropagating gradients layer-by-layer in the current module and updating the weights accordingly.

In this sense, locally supervised learning can substantially mitigate the limitations discussed previously. The gradient isolation between the modules at the two parties eliminates the need for the server to send additional data to the clients, and for clients to wait for gradients from the server to complete the backpropagation in each iteration of training. Moreover, the server can aggregate features from selected clients before conducting mini-batch training, which can mitigate the impact of heterogeneous data distribution among clients.

III. INFOCOMM: ARCHITECTURAL DESIGN

In this section, we introduce INFOCOMM, our proposed new federated learning architecture. We will begin by giving an overview of its design, followed by an explanation of its key characteristics, including locally generated loss, auxiliary networks, feature aggregation, and total cost analysis.

A. Workflow Overview

INFOCOMM leverages the benefits of federated learning and split learning, allowing multiple clients to train sub-models in parallel and offloading the bulk of the training process to a more powerful server, all without the need of transferring raw data. INFOCOMM empowers clients by allowing them to utilize independent local training strategies, resulting in greater flexibility and customization. Furthermore, it reduces both the frequency and size of information exchanges between clients and the server, optimizing the efficiency of the overall system. Moreover, the server employs an innovative approach to acquire mini-batches of features from various clients for training its sub-model, as opposed to sequentially using features.

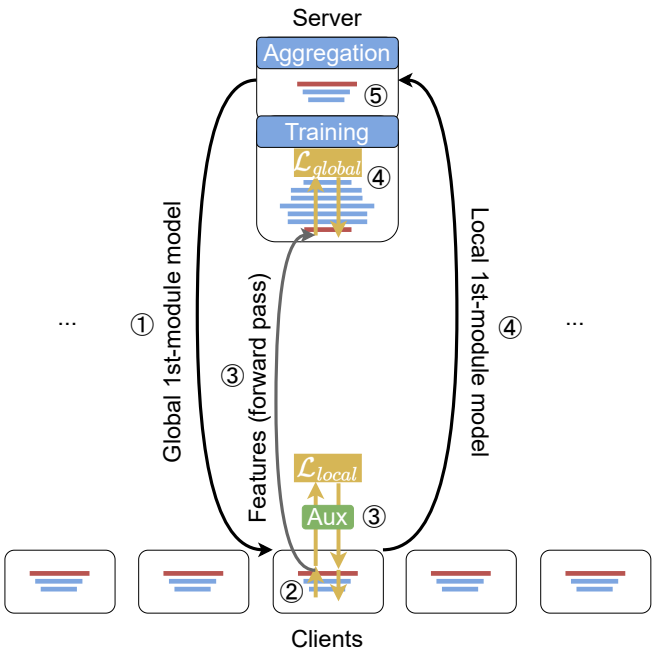


Fig. 3: INFOCOMM: an overview of its design.

Fig. 3 illustrates our proposed framework and distinguishes it from the standard approaches of federated learning, split learning, and SplitFed shown in Fig. 1 and Fig. 2. In each round t, the server and clients engage in communication and training according to the following steps:

Step 1: At the beginning, each client k, randomly selected by the server, downloads the current weights $\mathbf{w}_c^t$ of the global 1st-module model $\boldsymbol{\theta}_c^t$ from the server.

Step 2: Each selected client k applies forward propagation to its local data $\boldsymbol{D}_k$ through multiple epochs of mini-batch training. Each iteration generates a feature extracted from the cut layer.

Step 3: After completing the forward propagation process, client k uploads the collected features $\boldsymbol{h}_k^t$, along with the corresponding true labels $\hat{\boldsymbol{y}}_k^t$ for all iterations, to the server. Meanwhile, the client generates its local loss by the auxiliary networks and performs backpropagation to update its local 1st-module model $\boldsymbol{\theta}_{c_k}^t$, as well as the auxiliary networks.

Step 4: The server performs forward propagation through mini-batch training on the received features, and backpropagation based on the task-related cross-entropy loss. This process updates the weights $\mathbf{w}_s^{t+1}$ of its 2nd-module model $\boldsymbol{\theta}_s^{t+1}$. Concurrently, client k uploads its updated weights $\mathbf{w}_{c_k}^t$ to the server.

Step 5: The server then aggregates the 1st-module models from the selected clients to obtain a new set of global weights $\mathbf{w}_c^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{w}_{c_k}^t$. Once the update of the 2nd-module model and aggregation of the 1st-module models are complete, the server proceeds to the next round $t+1$ by selecting a new subset of K clients.

Similarly to SplitFed (SFLV2), the server in INFOCOMM trains the 2nd-module model in a centralized manner using the features generated by the clients, which is more effi-

cient compared to the decentralized aggregation of model parameters in conventional federated learning. However, it is important to note that the SplitFed approach is expected to repeat steps 3-6 in a single global communication round, as illustrated in Fig. 2, as each client typically performs multiple training iterations and exchanges features and gradients with the server for each batch. In contrast, INFOCOMM significantly reduces the number of steps required in a single global round. Additionally, in INFOCOMM, the model update at the clients and the training at the server can overlap in time. This is because clients can complete their local training using locally generated loss without waiting for any actions or information from the server. Furthermore, the server's aggregation and training processes can proceed in parallel. In this sense, locally supervised learning eliminates the need for additional steps of communication and the delays associated with sequential learning in SplitFed.

B. Locally Generated Loss

Locally supervised learning tries to optimize individual components separately based on local loss, which may deviate from the big picture of optimizing the overall model's performance on the task [21]. Therefore, it is vital to design a proper local loss function that is not only informative for local training, but also aligns with the overall objective of optimizing the full model's performance. Limitations of traditional locally supervised learning methods can lead to suboptimal results due to short-sighted local loss functions (such as local classification loss), as they collapse task-relevant information at early layers that may be useless for short-term performance but is actually useful for the full model.

To address these limitations, we have selected to employ the information propagation (InfoPro) loss introduced in recent research [11]. The InfoPro loss has proven to be highly effective in retaining important information of the input while discarding task-irrelevant information, ultimately leading to enhanced performance of the overall model. The InfoPro loss function is written as:

$$\begin{aligned} \mathcal{L}_{\text{InfoPro}}\left(\boldsymbol{h}\right) &= \alpha\left[-I(\boldsymbol{h},\boldsymbol{x}) + \beta I(r^*,\boldsymbol{h})\right], \alpha,\beta \geq 0, \\ \text{s.t. } r^* &= \underset{r, I(r,\boldsymbol{x})>0, I(r,y)=0}{\operatorname{argmax}} I(r,\boldsymbol{h}), \end{aligned} \tag{1}$$

where x, y, and h denote the input data, the label, and the features, respectively. The nuisance r^* [22] captures as much task-irrelevant information in the features $\boldsymbol{h}$, which are the intermediate outputs of the local module, as possible. The first term of the loss function aims to retain as much information of the input as possible, while the second term aims to maximally discard the task-irrelevant information. The coefficient β controls the effects of these two goals, while α balances the loss of the 1st-module at clients and the loss of the 2nd-module at the server in our application. Subsequently, an upper bound of $\mathcal{L}_{\text{InfoPro}}$ can be derived as an surrogate objective [11] that is easier to optimize compared to Eq. (1):

$$\mathcal{L}_{\text{InfoPro}} \leq -\alpha(1-\beta)I(\boldsymbol{h},\boldsymbol{x}) - \alpha\beta I(\boldsymbol{h},y) \triangleq \bar{\mathcal{L}}_{\text{InfoPro}}. \tag{2}$$

In line with the approach described in [11], we utilize two small auxiliary networks to estimate the mutual information $I(\boldsymbol{h},\boldsymbol{x})$ and $I(\boldsymbol{h},y)$ in $\bar{\mathcal{L}}_{\text{InfoPro}}$, respectively. The estimation works as follows.

$I(\boldsymbol{h},\boldsymbol{x}) \leftarrow$ **decoder** $\boldsymbol{w}$**:** According to [23], [24], the relationship $I(\boldsymbol{h},\boldsymbol{x}) = H(\boldsymbol{x}) - H(\boldsymbol{x}|\boldsymbol{h}) \geq H(\boldsymbol{x}) - \mathcal{R}(\boldsymbol{x}|\boldsymbol{h})$ holds for $I(\boldsymbol{h},\boldsymbol{x})$ and $\mathcal{R}(\boldsymbol{x}|\boldsymbol{h})$, the expected error for reconstructing $\boldsymbol{x}$ from $\boldsymbol{h}$. This implies that $I(\boldsymbol{h},\boldsymbol{x})$ can be approximated by $I(\boldsymbol{h},\boldsymbol{x}) \approx \max_{\boldsymbol{w}}\left[H(\boldsymbol{x}) - \mathcal{R}_{\boldsymbol{w}}(\boldsymbol{x}|\boldsymbol{h})\right]$, where $H(\boldsymbol{x})$ is a constant and $\mathcal{R}_{\boldsymbol{w}}(\boldsymbol{x}|\boldsymbol{h})$ is the reconstruction loss obtained by an auxiliary decoder parameterized by $\boldsymbol{w}$. In this case, the auxiliary decoder takes the feature at the cut layer as input and produces a reconstructed input. The reconstruction loss is calculated as the binary cross-entropy loss between the reconstructed input and the original input.

$I(\boldsymbol{h},y) \leftarrow$ **classifier** $\boldsymbol{\psi}$**:** $I(\boldsymbol{h},y)$ can be expanded as $I(\boldsymbol{h},y) = H(y) - H(y|\boldsymbol{h}) = H(y) - \mathbb{E}_{(\boldsymbol{h},y)}[-\log p(y|\boldsymbol{h})]$. Training an auxiliary classifier parameterized by $\boldsymbol{\psi}$ to approximate the conditional distribution $p(y|\boldsymbol{h})$ leads to $I(\boldsymbol{h},y) \approx \max_{\boldsymbol{\psi}}\{H(y) - \mathbb{E}_{\boldsymbol{h}}[\sum_y -p(y|\boldsymbol{h})\log q_{\boldsymbol{\psi}}(y|\boldsymbol{h})]\}$. The auxiliary classifier $q_{\boldsymbol{\psi}}(y_i|\boldsymbol{h}_i)$ can be trained using the cross-entropy loss between the network output and the true labels.

As a result, the auxiliary networks $\boldsymbol{w}$ and $\boldsymbol{\psi}$ are trained collaboratively with the 1st-module model $\boldsymbol{\theta_c}$ at the clients with an optimization objective:

$$\begin{aligned} \underset{\boldsymbol{\theta_c},\boldsymbol{w},\boldsymbol{\psi}}{\text{minimize}}\ & \alpha(1-\beta)\,\mathcal{R}_{\boldsymbol{w}}(\boldsymbol{x}|\boldsymbol{h}) \\ & + \alpha\beta\frac{1}{N}\sum_{i=1}^{N} -\log q_{\boldsymbol{\psi}}(y_i|\boldsymbol{h}_i). \end{aligned} \tag{3}$$

The architecture of the auxiliary networks. Based on the findings of the study by [11], including at least one convolutional layer in the auxiliary network architecture is important. However, the study also demonstrates that using larger networks does not offer a significant improvement in performance and instead increases computational overhead. Therefore, as presented in Table I, we designed the auxiliary decoder $\boldsymbol{w}$ and the auxiliary classifier $\boldsymbol{\psi}$ for generating the InfoPro loss. The two auxiliary networks are small compared to the entire primary network, which allows for efficient computation at the client side. As our subsequent experimental results will show, the design of the auxiliary networks we employed is generalizable and effective for both LeNet-5 and ResNets. Adjustments, such as including additional convolutional layers in the auxiliary network, may be necessary for other models. These adjustments can enhance the decoding and classifying capabilities of the auxiliary networks, ensuring that the locally generated loss facilitates independent model training.

Aggregation for auxiliary networks. When employing locally supervised learning in federated learning, we face a new issue: auxiliary networks at different clients are trained in different directions. Will the discrepancy between these auxiliary networks affect the global learning performance or facilitate clients' personalized local losses? Intuitively, we can either let clients train their auxiliary networks completely locally or allow them to upload their auxiliary networks to the server along with their 1st-module main network, then download the global auxiliary network after aggregation. We empirically investigate the effects of aggregating auxiliary networks across different

TABLE I: The architecture of the auxiliary decoder and the auxiliary classifier.

Network structure of w	Network structure of ψ
nn.Conv2d	nn.Conv2d
nn.BatchNorm2d	nn.BatchNorm2d
nn.ReLU	nn.ReLU
nn.Conv2d	nn.AdaptiveAvgPool2d
nn.Sigmoid	nn.Flatten
	nn.Linear
	nn.ReLU
	nn.Linear

clients, along with the 1st-module networks at the server, in various datasets with different learning models, as summarized in Table II. In the same settings, we noticed that aggregating auxiliary networks leads to improvements in global accuracy upon convergence. Specifically, for LeNet-5 on MNIST and LeNet-5 on FashionMNIST, the global accuracy increased by 6.86% and 14.37% respectively. However, in the case of ResNet-18 on CIFAR10, the accuracy decreased by 9.23%. The results indicate that aggregating the auxiliary networks at the server does not always facilitate learning, as each client might have quite divergent auxiliary networks, especially for more complicated primary networks like ResNet. Keeping the auxiliary networks locally, without aggregating them at the server side, can be beneficial for generating personalized local loss for different clients. This approach also avoids the need for additional communication between clients and the server. We will further compare the training and communication costs associated with both auxiliary network aggregation and non-aggregation methods in Section III-D.

TABLE II: Comparison of global test accuracy upon convergence with and without aggregation for auxiliary networks at the server.

Scenario	w/ aux aggregation	w/o aux aggregation
MNIST, LeNet-5	98.1%	91.8%
FashionMNIST, LeNet-5	87.23%	76.27%
CIFAR10, ResNet-18	76.7%	84.5%

C. Feature Aggregation

As previously discussed in Section II, SplitFed requires the server to process features from clients sequentially so that it can send back the corresponding gradients of loss for each batch of features to complete the backpropagation process at clients. However, by combining federated learning with locally supervised learning, our proposed architecture provides an advantage on this issue. As the server no longer needs to send back gradients, it can aggregate the features $\boldsymbol{h}_k$ received from the selected K clients to form a larger feature dataset denoted as $\boldsymbol{D_h}$. This dataset consists of features generated from each raw data sample through the 1st-module model at a client, along with the corresponding true labels. $\boldsymbol{D_h}$ then can be used to sample data for training the 2nd-module model in a flexible and isolated manner, using different epoch and batch sizes for mini-batch training.

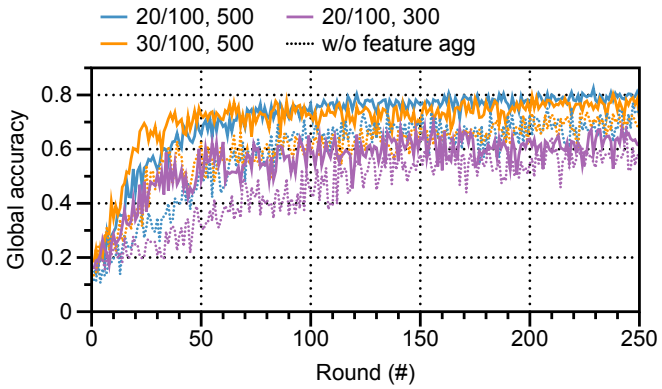


Fig. 4: Comparison of test accuracy across rounds with and without aggregation for features.

Algorithm 1 INFOCOMM: Server Side

Initialize: Global 1st-module model weights $\mathbf{w}_c^0$; 2nd-module model weights $\mathbf{w}_s^0$

2nd-Module Model Update:
Send $\mathbf{w}_c^t$ to all K clients for $ClientUpdate(\mathbf{w}_{c_k}^t)$
for each client $k \in S_t$ in parallel **do**
  for each gradient descent iteration **do**
    Receive features $\mathbf{h}_k^t$ and labels $\mathbf{Y}_k^t$ from $ClientUpdate(\mathbf{w}_{c_k}^t)$
    Aggregate features $\mathbf{h}_k^t$ for $k \in S_t$ into feature dataset $\boldsymbol{D}_{\boldsymbol{h}}^t$
  end for
  for each mini-batch b **do**
    Perform forward propagation with a batch of features $\mathbf{h}_b^t$ from $\boldsymbol{D}_{\boldsymbol{h}}^t$ on the 2nd-module model $\mathbf{w}_s^t$ and get the corresponding predictions $\hat{\mathbf{Y}}_b$
    Calculate loss with $\mathbf{Y}_b$ and $\hat{\mathbf{Y}}_b$ and perform back-propagation
    Update 2nd-module model weights $\mathbf{w}_s^{t+1} \leftarrow \mathbf{w}_s^t - \eta\nabla\ell(\hat{\mathbf{Y}}_b; \mathbf{Y}_b)$
  end for
end for

1st-Module Model Aggregation:
for each client $k \in S_t$ in parallel **do**
  Receive $\mathbf{w}_{c_k}^{t+1}$ from each selected client k
end for
Update global 1nd-module model weights $\mathbf{w}_c^{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{w}_{c_k}^{t+1}$

We show the benefits of aggregating features at each round by presenting empirical results in Fig. 4. We conducted experiments on the CIFAR10 dataset with ResNet-18 using federated learning with InfoPro, where the server selects 20 or 30 clients from a pool of 100 clients and each client has 500 or 300 data samples. The results we obtained in both scenarios strongly support our statement that aggregating features from different clients into a larger dataset for training the 2nd-module model at the server leads to a significant improvement in convergence

TABLE III: Total cost analysis of different schemes in one global round.

Scheme	Downlink comm time	Uplink comm time	Client training time	Server training time
FedAvg	$\frac{\lvert\mathbf{w}\rvert}{R_s}$	$\frac{\lvert\mathbf{w}\rvert}{R_c}$	$\frac{(E\lceil P/B\rceil)L}{r_c}$	0
SplitFed	$\frac{\eta\lvert\mathbf{w}\rvert}{R_s} + (E\lceil P/B\rceil)\frac{Bq}{R_s}$	$\frac{\eta\lvert\mathbf{w}\rvert}{R_c} + (E\lceil P/B\rceil)\frac{Bq}{R_c}$	$\frac{(E\lceil P/B\rceil)\eta L}{r_c}$	$K\frac{(E\lceil P/B\rceil)(1-\eta)L}{r_s}$
INFOCOMM (w/ aux aggregation)	$\frac{\eta\lvert\mathbf{w}\rvert+\lvert\mathbf{w}_a\rvert}{R_s}$	$\frac{\eta\lvert\mathbf{w}\rvert+\lvert\mathbf{w}_a\rvert}{R_c} + \frac{EPq}{R_c}$	$\frac{(E\lceil P/B\rceil)(\eta L+L_a)}{r_c}$	$\frac{(E\lceil KP/B\rceil)(1-\eta)L}{r_s}$
INFOCOMM (w/o aux aggregation)	$\frac{\eta\lvert\mathbf{w}\rvert}{R_s}$	$\frac{\eta\lvert\mathbf{w}\rvert}{R_c} + \frac{EPq}{R_c}$		

speed.

The algorithms conducted at the server and client sides are depicted in Algorithms 1 and 2.

Algorithm 2 INFOCOMM: Client Side

ClientUpdate $(\mathbf{w}_{c_k}^t)$:
Update local 1nd-module model weights from the current global one $\mathbf{w}_{c_k}^t \leftarrow \mathbf{w}_c^t$
for each local epoch e from 1 to E **do**
  Perform forward propagation with raw data X_k up to the final layer in $\mathbf{w}_{c_k}^t$
  Obtain features $\mathbf{h}_k^t$, *i.e.*, the representations of the final layer
  Send features $\mathbf{h}_k^t$ and corresponding labels $\mathbf{Y}_k^t$ to the server
  Calculate locally generated loss ℓ_k with the auxiliary networks $\boldsymbol{w}$ and $\boldsymbol{\psi}$ according to Eq. (3) and perform backpropagation
  Update 1nd-module model weights $\mathbf{w}_{c_k}^{t+1} \leftarrow \mathbf{w}_{c_k}^t - \eta\nabla\ell_k$
end for
Send $\mathbf{w}_{c_k}^{t+1}$ to the server

D. Total Cost Analysis

In the analysis of SplitFed [10], both the communication time and training time in one global round for different learning schemes have been studied. However, such analysis only considered the simplest case, where there is only one batch of training in each global round. This contradicts the realistic settings in conventional federated learning where clients perform multiple iterations of mini-batch training, and in practice, SplitFed may result in significant communication overhead and delays in training as its actual cost is much larger than FedAvg in one global round.

To address the limitations of such analysis, we provide a more accurate and elaborated analysis of the communication time and training time at the server and client in each global round for FedAvg, SplitFed, and INFOCOMM. We assume that in each round, the server selects K clients, each client has P data samples and conducts E local epochs of training with batch size B. We use $|\mathbf{w}|$ to represent the corresponding actual size of model weights, and use q to represent the size of features or gradients at the cut layer per data sample. We denote η as the fraction of the size of the 1st-module model at the client of the full model, i.e., $|\mathbf{w}_c| = \eta|\mathbf{w}|$. We also denote R_s and R_c as the downlink and uplink data transmission rate, respectively, L as the workload for a forward and backward propagation through the full model for one batch of data, and L_a as the workload for training the auxiliary network for one batch of data. Note that, we consider the case where the server and clients have different computational power due to the use of GPUs or CPUs, and thus, the server's training rate is r_s while the client's training rate is r_c, which is smaller than r_s.

The complete breakdown of the total cost in terms of communication and training time is presented in Table III. The communication time is calculated per client, and the training time does not include the time taken for aggregation or other processing time. In our proposed scheme, INFOCOMM, the communication time is always lower than SplitFed by at least $(E\lceil P/B\rceil)\frac{Bq}{R_s}$, as INFOCOMM does not require the server to send back gradients at the cut layer per batch to each client. Increasing the depth of the auxiliary network with additional convolutional or fully connected layers can potentially improve the learning performance as it can generate a more informative local loss. However, a stronger auxiliary network can also introduce extra training time at the client.

Compared to FedAvg, INFOCOMM can significantly reduce the client training time by offloading the workload to the server. Although the server needs to train the 2nd-module model with the feature data from all selected clients, it has much more powerful computational capacity to do so than clients, which may result in a total training time that is smaller than FedAvg's. Additionally, INFOCOMM can actually reduce the communication time when P is small enough, which is a common scenario for federated learning in edge devices. We have validated our analysis with additional empirical results, as described in Section IV-B.

IV. PERFORMANCE EVALUATION

To evaluate the performance of INFOCOMM experimentally, we have implemented both SplitFed [10] and INFOCOMM in the open-source federated learning framework, PLATO [25], which is designed to support real-world federated learning experiments using a limited amount of computing resources. Our implementation will be released as open-source as well to support the best possible reproducibility of this work. With our implementation, we conducted a diverse array of experiments comparing our proposed method, INFOCOMM, with FedAvg [3], SCAFFOLD [12], and SplitFed [10]. Through our experiments, we investigated and validated the learning and

communication efficiency of these approaches from various perspectives, including non-IID data, large neural networks, small clients, and server-side training flexibility. All of our experiments were conducted on an Mac Studio with an Apple M1 Max processor, featuring a 10-core CPU, a 24-core GPU, and 64GB of unified memory.

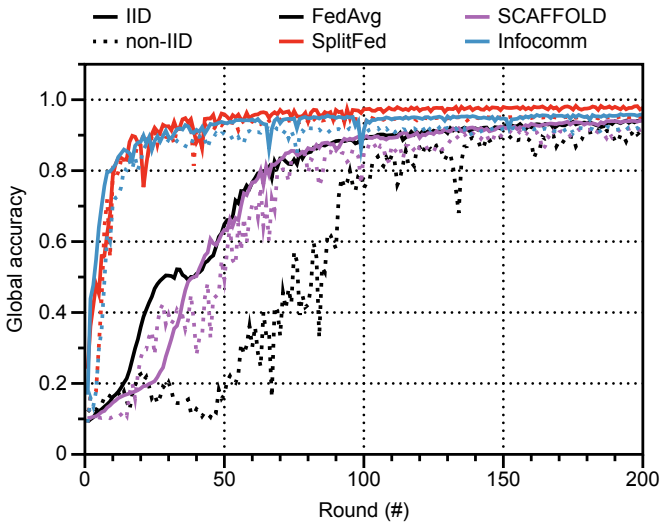
Datasets and models. We conducted our experiments on various datasets including MNIST [26], FashionMNIST [27], and CIFAR10 [28], with LeNet-5 [29] and several variants of ResNet [30] neural network architectures including ResNet-18, ResNet-101, ResNet-152, which have approximately 11, 42, 58 million trainable parameters, respectively. For model splitting in SplitFed and INFOCOMM, we set the cut layer at `ReLU2` for LeNet-5 and at `Layer1` for all variants of ResNets. The decoder network and auxiliary classifier network have 2k and 10k trainable parameters, respectively. The total size of them is only 0.49% of the entire ResNet-18. We choose to aggregate the auxiliary networks at the server for LeNet-5 and not for ResNets to achieve optimal performance.

Setups. In our experiments, we follow a standard federated learning setting where the server randomly selects K clients in each round. The entire dataset is evenly distributed among a pool of clients, where each client even has P data samples. For example, if we consider the entire MNIST dataset, which consists of $50,000$ samples, and we have a total of 100 clients, then each client would hold 500 samples. Unless otherwise specified in experiment scenarios such as Section IV-A, the data is independent and identically distributed (IID) across clients, and the server in SplitFed and INFOCOMM has the same training settings as the clients. We train each client's local model with a mini-batch size of B and epoch number E ($E = 1$ as default). To train LeNet-5, we use the SGD optimizer with a learning rate of 0.01 and momentum of 0.9; while for ResNets, we use the same optimizer with a learning rate, momentum, and weight decay of 0.01, 0.9, and 0.0001, respectively. Additionally, we bind the optimizer with a PyTorch LambdaLR learning rate scheduler for training ResNets. We use the default hyperparameters used in [11] for the auxiliary networks in INFOCOMM, such as $\alpha(1-\beta) = 5$ and $\alpha\beta = 1$. Settings and configurations, such as the number of clients selected K, the number of samples on each client P, and the batch size B, will be specified in different experiment scenarios below.

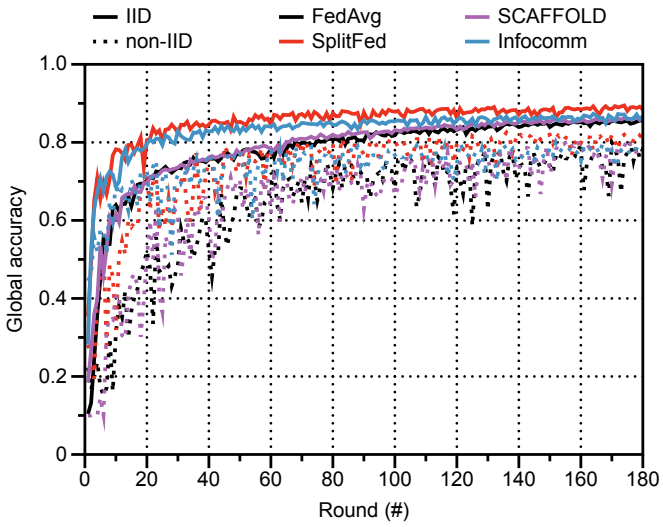
A. Convergence Performance

The convergence performance of different schemes when applied to IID or non-IID data distributions is demonstrated in Fig. 5. We tested these schemes on MNIST and FashionMNIST datasets with the LeNet-5 model, and set the parameters $K = 10, P = 300, B = 128$ and $K = 10, P = 300, B = 32$ for them, respectively. To simulate non-IID conditions, we employed a Dirichlet distribution with a concentration parameter of 0.5 and 0.1 for the two scenarios, respectively.

For the MNIST dataset, the test accuracies of SplitFed and INFOCOMM ramp up very quickly and steadily in the first 20 rounds compared to FedAvg and SCAFFOLD, even for the non-IID setting, as shown in Fig. 5a. Although the gap is



(a) MNIST & LeNet-5

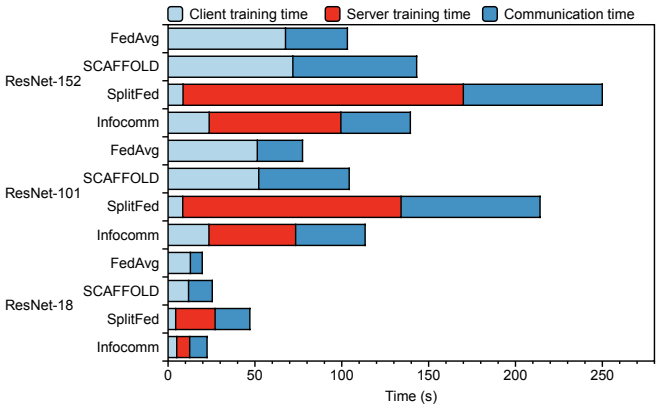


(b) FashionMNIST & LeNet-5

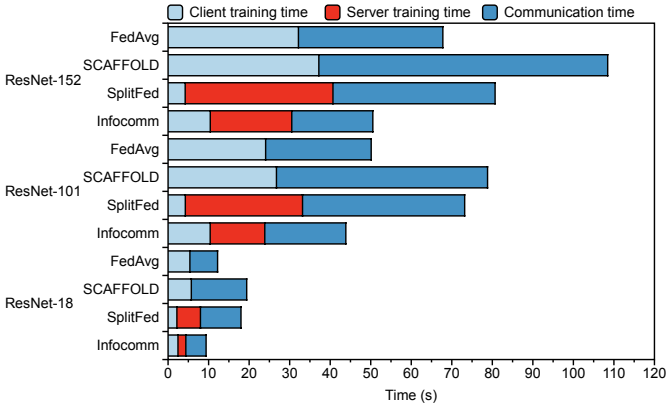
Fig. 5: Global model test accuracy over rounds on IID or non-IID data.

not as significant with the more complicated FashionMNIST dataset, SplitFed and INFOCOMM still outperform FedAvg and SCAFFOLD in terms of the speed of convergence in both IID and non-IID settings. SCAFFOLD excels over FedAvg, particularly in the non-IID setting. At round 150 in Fig. 5a, INFOCOMM has already converged with an accuracy 2.65% and 4.96% higher than FedAvg in IID setting and non-IID setting, respectively. The primary reason for this improvement is that SplitFed (SFLV2) and INFOCOMM use a single 2nd-module model at the server that is trained consistently with batches of feature data collected from different clients, instead of having multiple full models trained separately at clients and then aggregated with weights as in standard federated learning.

We can also notice that INFOCOMM is marginally inferior to SplitFed in terms of model accuracy per round, which can be attributed to the fact that locally generated loss in INFOCOMM does not capture the same information as the global classi-

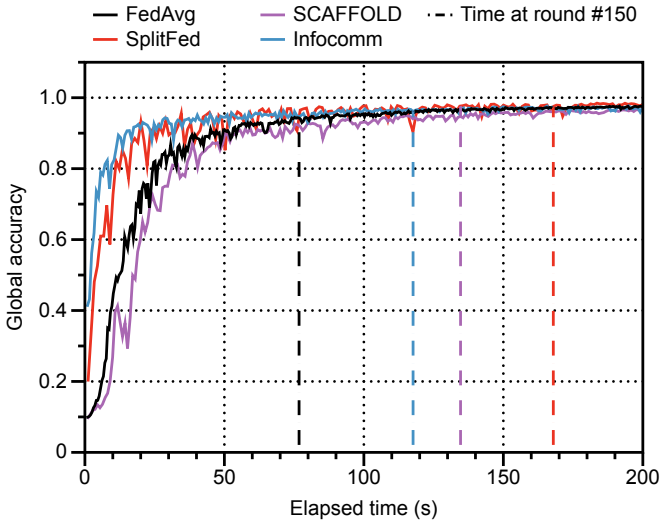


(a) $K = 20, P = 500$

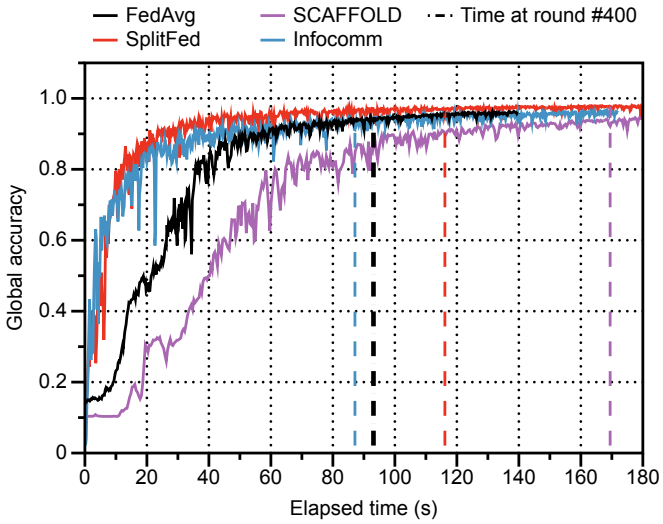


(b) $K = 10, P = 250$

Fig. 6: A breakdown of elapsed time in each round of training different model with different methods.



(a) MNIST & LeNet-5; $K = 10$, $P = 100$



(b) MNIST & LeNet-5; $K = 5$, $P = 50$

Fig. 7: Global model test accuracy over elapsed time of different number of local data samples.

fication loss. Nevertheless, the reduction of communication overhead and delay brought by locally generated loss can make up for this drawback, as we will show in the following results.

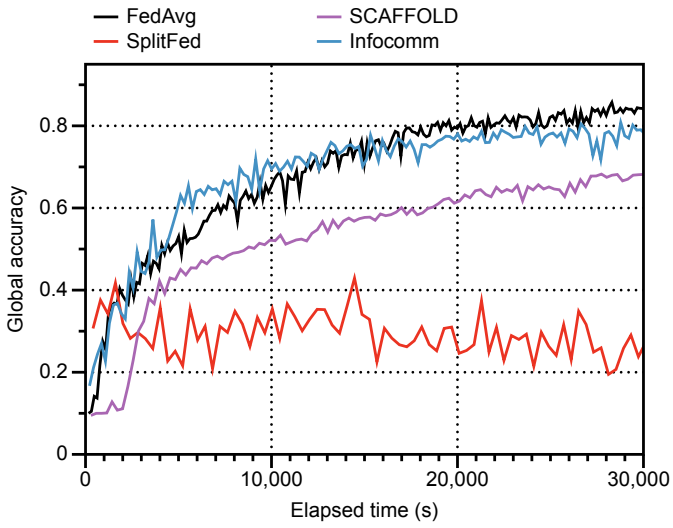
B. Elapsed Time Breakdown when Training Different Models

To assess the impact of different schemes on delays (measured with the elapsed time), we measured the average training time at the clients, the communication time between the server and one client, and the server training time, when using various ResNet models with different numbers of layers.

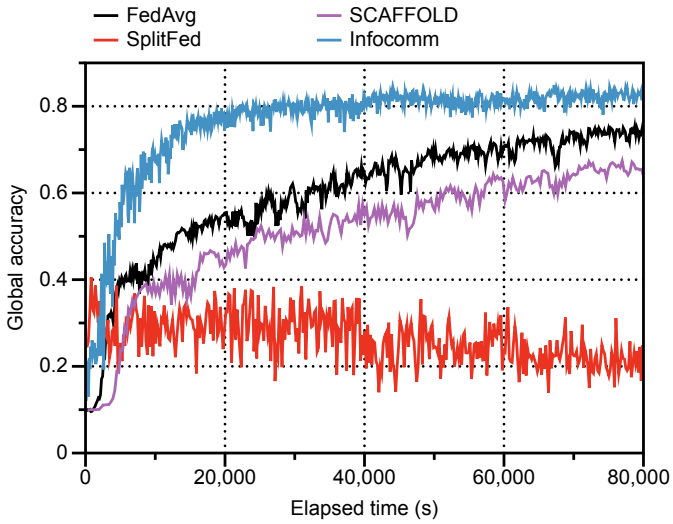
To reflect real-world scenarios, all client training was performed on CPUs while the server was equipped with GPUs. The breakdown of the average time taken during a single training round for each scheme is displayed in Fig. 6. We evaluated different values for K and P, as they are the major factors that affect the time in SplitFed and INFOCOMM with model splitting and server-side training. The batch size is set to $B = 128$ for both cases.

We can observe that INFOCOMM consistently has a shorter client training time compared to FedAvg and SCAFFOLD due to the model split, including the training time for auxiliary networks, as they are relatively small compared to the main network. As the model size increases and P decreases, the communication time of INFOCOMM can get closer to, and even become smaller than, that of FedAvg or SCAFFOLD, since the addition of $\frac{EPq}{R_c}$ can be compensated by the reduction of $\frac{(1-\eta)|\mathbf{w}|}{R_c} + \frac{(1-\eta)|\mathbf{w}|}{R_s}$ by INFOCOMM. SCAFFOLD requires slightly more client training time and nearly double the communication time compared to FedAvg because the server and clients need to update and send the client control variates.

Compared with SplitFed, INFOCOMM has a slightly longer client training time due to the auxiliary network training, but it significantly reduces communication overhead and delay introduced by downloading gradients in each batch of training. The server training time in SplitFed is longer compared to that of INFOCOMM, which is somewhat inconsistent with our theoretical analysis. The reason for this discrepancy is that during implementation, SplitFed needs to prepare all the model and optimizer states every time for propagating each batch of



(a) CIFAR10 & ResNet-18; $K = 20$, $P = 500$



(b) CIFAR10 & ResNet-18; $K = 20$, $P = 250$

Fig. 8: Global model test accuracy over elapsed time of different number of local data samples.

feature data, whereas INFOCOMM only requires preparation once with a single large feature dataset.

In the most favorable scenario among these experiments, where the model is relatively the largest, INFOCOMM can achieve a time reduction of 28%, 54%, and 37%, compared to FedAvg, SCAFFOLD, and SplitFed, respectively. It is worth emphasizing that in INFOCOMM, clients do not need to wait for the server to send back gradients or other information during their local training, so the client training and server training time can overlap. In this sense, INFOCOMM can achieve more time-efficient training than its alternatives.

C. Better Efficiency for Less Capable Clients

Compared to standard federated learning approaches like FedAvg and SCAFFOLD, INFOCOMM is particularly efficient for less capable clients such as edge devices, which often have a limited amount of data samples and computational power. To validate this claim, we conducted experiments on MNIST with LeNet-5 and on CIFAR10 with ResNet-18, varying the number of data samples per client P. The results are shown in Fig. 7 and Fig. 8.

In the scenario of MNIST with LeNet-5 as shown in Figs. 7a and 7b, both SplitFed and INFOCOMM still demonstrated faster convergence speed and higher ultimate model accuracy than FedAvg. Furthermore, each round of INFOCOMM saved about $25\% - 30\%$ of the time compared to SplitFed. For instance, as illustrated by the dashed lines in Fig. 7a, SplitFed requires approximately $1.18\times$ more elapsed time than FedAvg, whereas INFOCOMM only incurs a $0.53\times$ increase in elapsed time compared to FedAvg. In Fig. 7b, SplitFed experiences an increase of $0.26\times$ in elapsed time compared to FedAvg, while INFOCOMM demonstrates a reduction of 5.4% in elapsed time compared to FedAvg. These results also align with our analysis in Section III-D and Section IV-B that, with smaller values of P, the total elapsed time per round of INFOCOMM can be even smaller than that of FedAvg.

In the scenario of CIFAR10 with ResNet-18, we observed that INFOCOMM achieved similar learning progress over the elapsed time as FedAvg as seen in Fig. 8a. However, in a more challenging scenario with a limited number of data samples, as depicted in Fig. 8b, INFOCOMM demonstrated superior performance compared to FedAvg, resulting in a 15.3% higher accuracy at time $60,000$s when reaching convergence. In contrast, SplitFed failed to converge in both cases. SplitFed only achieved approximately 30% accuracy in average for both cases. The authors of SplitFed also mentioned this similar phenomenon in their paper [10]. We suspect that one of the reasons for SplitFed's failure to converge could be that communication between the server and clients can result in losing important information in the optimizer states and other factors. This might lead to inconsistent settings of the 1st- and 2nd-module networks for backpropagation with the same global loss. In contrast, INFOCOMM is not affected by the separate training for the two modules because they use local losses to update the model weights. This ensures that the optimizer states and other factors of each module during back-propagation are consistent with corresponding loss. In all the four scenarios shown in Figs. 7 and 8, SCAFFOLD is unable to match FedAvg in terms of test accuracy versus elapsed time. This observation aligns with our analysis in Section IV-B that SCAFFOLD requires more training and communication time due to the updating and sharing of client control variates.

D. Server- and Client-specific Epoch and Batch Sizes

Here we show another bonus of INFOCOMM that FedAvg, and SplitFed cannot benefit from. When the server and clients have a consistent batch size of B and the number of epochs E, we refer to the method as vanilla INFOCOMM. Since the training at the server and clients is separated and does not require each batch to be the same between them, the server can use a smaller batch size B and a larger number of epochs E to train the 2nd-module model, which can further enhance

TABLE IV: Comparison of global model test accuracy at certain rounds, when the server uses different epoch sizes or batch sizes than the clients.

Scenario	Client-side	Server-side	Accuracy					
			FedAvg	SplitFed	INFOCOMM	FedAvg	SplitFed	INFOCOMM
				Round #15			Round #150	
MNIST & LeNet-5	$E = 1, B = 32$	$E = 1, B = 32$	15.38%	43.55%	58.43%	73.07%	93.77%	86.39%
$K = 5, P = 50$		$E = 5, B = 32$			87.80%			94.37%
				Round #10			Round #50	
FashionMNIST & LeNet-5	$E = 1, B = 128$	$E = 1, B = 128$	10%	55.72%	60.06%	62.02%	78.1%	76.02%
$K = 10, P = 300$		$E = 1, B = 5$			75.49%			80.77%
				Round #15			Round #100	
CIFAR10 & ResNet-18	$E = 1, B = 128$	$E = 1, B = 128$	36.65%	27.83%	44.04%	75.86%	25.1%	78.68%
$K = 20, P = 500$		$E = 5, B = 32$			60.86%			79.55%

the learning efficiency. In this case, we refer to the method as advanced INFOCOMM. We show the global model accuracy at different specific rounds during training in different settings in Table IV, comparing FedAvg, SplitFed, vanilla INFOCOMM, and advanced INFOCOMM.

As shown in the results, regardless of the specific setting, the advanced INFOCOMM can achieve higher global model accuracy than FedAvg and vanilla INFOCOMM by using a smaller batch size B and a larger number of epochs E at the server to train the 2nd-module model. In the best-case scenario, when comparing the test accuracy of the global model during the early stage of training, the advanced INFOCOMM outperforms FedAvg and SplitFed by achieving $6.5\times$ and $1.2\times$ higher accuracy, respectively. During the later stage of training, the advanced INFOCOMM consistently achieves an average accuracy that is 0.21% higher than FedAvg across all three different scenarios.

In our previous experiments, the vanilla INFOCOMM may be outperformed by SplitFed on MNIST and FashionMNIST in terms of accuracy versus round (instead of accuracy versus elapsed time shown in Section IV-C) when approaching convergence, due to the fact that its locally generated loss potentially might lose specific task-relevant information compared to the global classification loss used in SplitFed. Fortunately, however, adjusting the epoch and batch sizes at the server can more than make up for this small negative impact, fully utilizing the abundant computational power available at the server. For example, in the first scenario, when the vanilla INFOCOMM approached convergence, its accuracy was 7.87% lower than SplitFed. However, the advanced INFOCOMM exhibited a notable improvement, with an accuracy that was 3.4% higher than SplitFed.

V. FURTHER DISCUSSIONS AND CONCLUDING REMARKS

In this paper, we proposed INFOCOMM, a novel architecture that addresses significant obstacles due to severely limited computational power on IoT devices in federated learning. The upshot of our original contributions lies in the utilization of model splitting and InfoPro loss to enable locally supervised learning, as well as a unique feature aggregation algorithm designed for our architectural design. We have presented a theoretical analysis of the total cost in comparison with SplitFed, a state-of-the-art alternative, as well as an extensive array of experimental results that demonstrate INFOCOMM's efficiency in reducing both the training and communication times. Our approach is particularly suitable for large learning models, non-IID data distribution, and limited local data. Overall, our work offers a promising direction for improving the efficiency and scalability of federated learning in edge computing scenarios.

It is worth pointing out that our proposed architecture for federated learning offers a much greater degree of flexibility compared to traditional federated learning in terms of client selection and workload distribution. As training can occur concurrently at both the client and server, clients can proactively upload their local features instead of being passively selected by the server at the beginning of each round. Specifically, clients can download the global 1st-module model weights at any time and perform local training to generate new features, while the server is training the 2nd-module model with the updated feature datasets. Clients only need to train a minimum number of necessary layers, reducing their training workload by a substantial margin. The remaining layers can be further separated and trained by a group of servers with more powerful computing capacities, while still preserving data privacy and security.

So far, there has been limited research conducted on the privacy and security of the hybrid approach that combines split learning and federated learning. Such a learning framework may introduce extra vulnerabilities due to the exposure of intermediate outputs and gradients related to the cut layer. Instead, by incorporating locally supervised learning into federated learning in our approach, we can avoid the exposure of gradients and enhance the security of the learning framework. Nevertheless, it is still important to protect the features and labels transmitted from clients to the server. We can leverage existing methods from secure split learning to prevent information leakage in our work. For example, we can extend techniques such as incorporating the distance correlation between the raw data and the features at the cut layer into the loss function [31]. Additionally, we can enhance privacy by adding Laplacian noise to the features before transmitting them to the server [32]. To protect labels, we may also adopt the U-shaped configuration [9] where the clients retain the end layers and generate the gradients from them, which are then transmitted to the server for subsequent backpropagation.

REFERENCES

[1] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. Vincent Poor, "Federated Learning for Internet of Things: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[2] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.

[4] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A Survey on Federated Learning for Resource-Constrained IoT Devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, 2022.

[5] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization," *Proc. 34th Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 7611–7623, 2020.

[6] X. Xu, S. Duan, J. Zhang, Y. Luo, and D. Zhang, "Optimizing Federated Learning on Device Heterogeneity with a Sampling Strategy," in *Proc. IEEE/ACM International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1–10.

[7] Z. Chen, W. Liao, K. Hua, C. Lu, and W. Yu, "Towards Asynchronous Federated Learning for Heterogeneous Edge-Powered Internet of Things," *Digital Communications and Networks*, vol. 7, no. 3, pp. 317–326, 2021.

[8] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous Online Federated Learning for Edge Devices with Non-IID Data," in *Proc. IEEE International Conference on Big Data (Big Data)*, 2020, pp. 15–24.

[9] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split Learning for Health: Distributed Deep Learning without Sharing Raw Patient Data," *arXiv preprint arXiv:1812.00564*, 2018.

[10] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "SplitFed: When Federated Learning Meets Split Learning," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[11] Y. Wang, Z. Ni, S. Song, L. Yang, and G. Huang, "Revisiting Locally Supervised Learning: an Alternative to End-to-End Training," in *Proc. International Conference on Learning Representations (ICLR)*, 2021.

[12] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning," in *Proc. International conference on machine learning (ICML)*, 2020, pp. 5132–5143.

[13] C. Thapa, M. A. P. Chamikara, and S. A. Camtepe, "Advancements of Federated Learning towards Privacy Preservation: from Federated Learning to Split Learning," *Federated Learning Systems: Towards Next-Generation AI*, pp. 79–109, 2021.

[14] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated Learning for Internet of Things: Recent Advances, Taxonomy, and Open Challenges," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1759–1799, 2021.

[15] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed Comparison of Communication Efficiency of Split Learning and Federated Learning," *arXiv preprint arXiv:1909.09145*, 2019.

[16] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtepe, H. Kim, and S. Nepal, "End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things," *arXiv preprint arXiv:2003.13376*, 2020.

[17] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Combining Split and Federated Architectures for Efficiency and Privacy in Deep Learning," in *Proc. 16th International Conference on Emerging Networking EXperiments and Technologies*, 2020, pp. 562–563.

[18] Y. Gao, M. Kim, C. Thapa, S. Abuadbba, Z. Zhang, S. A. Çamtepe, H. Kim, and S. Nepal, "Evaluation and Optimization of Distributed Machine Learning Techniques for Internet of Things," *IEEE Transactions on Computers*, vol. 71, pp. 2538–2552, 2021.

[19] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Federated or Split? A Performance and Privacy Analysis of Hybrid Split and Federated Learning Architectures," *Proc. IEEE 14th International Conference on Cloud Computing*, pp. 250–260, 2021.

[20] Q. Duan, S. Hu, R. Deng, and Z. Lu, "Combined Federated and Split Learning in Edge Computing for Ubiquitous Intelligence in Internet of Things: State-of-the-Art and Future Directions," *Sensors*, vol. 22, no. 16, p. 5983, 2022.

[21] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Decoupled Greedy Learning of CNNs," in *Proc. International Conference on Machine Learning (ICML)*, 2020, pp. 736–745.

[22] A. Achille and S. Soatto, "Emergence of Invariance and Disentanglement in Deep Representations," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 1947–1980, 2018.

[23] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and Composing Robust Features with Denoising Autoencoders," in *Proc. International Conference on Machine Learning (ICML)*, 2008, pp. 1096–1103.

[24] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning Deep Representations by Mutual Information Estimation and Maximization," in *Proc. International Conference on Learning Representations (ICLR)*, 2019.

[25] "Plato: A New Framework for Scalable Federated Learning Research," https://github.com/TL-System/plato, accessed: 2023-09-01.

[26] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[27] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[28] A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-10 (Canadian Institute for Advanced Research). [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html.

[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

[31] P. Vepakomma, A. Singh, O. Gupta, and R. Raskar, "NoPeek: Information Leakage Reduction to Share Activations in Distributed Deep Learning," in *Proc. the 20th International Conference on Data Mining Workshops*, 2020, pp. 933–942.

[32] T. Titcombe, A. J. Hall, P. Papadopoulos, and D. Romanini, "Practical Defences against Model Inversion Attacks for Split Neural Networks," *arXiv preprint arXiv:2104.05743*, 2021.

Fei Wang (Student Member, IEEE) received her B.Eng. (Hons) degree from Hongyi Honor College at Wuhan University, China, in 2020. She is currently pursuing her Ph.D. degree in the Department of Electrical and Computer Engineering, University of Toronto, Canada. Her research interests lie at both efficiency improvement and privacy leakage in distributed machine learning. She was a recipient of the Best Paper Award with IEEE INFOCOM 2023.

Baochun Li (Fellow, IEEE) received his B.Eng. degree from Tsinghua University in 1995 and his M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign in 1997 and 2000, respectively. He is currently a Professor in the Department of Electrical and Computer Engineering at University of Toronto, and holds the Bell Canada Endowed Chair in Computer Engineering. His current research interests include cloud computing, security and privacy, distributed machine learning, federated learning, and networking. He is a Fellow of the Canadian Academy of Engineering.