Federated Split Learning With Joint Personalization-Generalization for Inference-Stage Optimization in Wireless Edge Networks

Dong-Jun Han, *Member, IEEE*, Do-Yeon Kim, Minseok Choi, *Member, IEEE*, David Nickel, *Graduate Student Member, IEEE*, Jaekyun Moon, *Fellow, IEEE*, Mung Chiang, *Fellow, IEEE*, and Christopher G. Brinton, *Senior Member, IEEE*

***Abstract*—The demand for intelligent services at the network edge has introduced several research challenges. One is the need for a machine learning architecture that achieves personalization (to individual clients) and generalization (to unseen data) properties concurrently across different applications. Another is the need for an inference strategy that can satisfy network resource and latency constraints during testing-time. Existing techniques in federated learning have encountered a steep trade-off between personalization and generalization, and have not explicitly considered the resource requirements during the inference-stage. In this paper, we propose SplitGP, a joint edge-AI training and inference strategy that simultaneously captures generalization/personalization for efficient inference across resource-constrained clients. The training process of SplitGP is based on federated split learning, with the key idea of optimizing the client-side model to have personalization capability tailored to its main task, while training the server-side model to have generalization capability for handling out-of-distribution tasks. During testing-time, each client selectively offloads inference tasks to the server based on the uncertainty threshold tunable based on network resource availability. Through formal convergence analysis and inference time analysis, we provide guidelines on the selection of key meta-parameters in SplitGP. Experimental results confirm the advantage of SplitGP over existing baselines.**

***Index Terms*—Federated learning, split learning, edge-AI, inference, personalization, wireless edge network.**

Manuscript received 1 May 2023; revised 4 September 2023; accepted 25 October 2023. Date of publication 10 November 2023; date of current version 7 May 2024. This work was supported in part by NRF grant funded by the MSIT of Korea under Grants NRF-2019R1I1A2A02061135 and NRF-2022R1C1C1010766, in part by IITP grant funded by MSIT of Korea under Grant 2021-0-02201, in part by the Center for Applied Research in Artificial Intelligence (CARAI) grant funded by DAPA and ADD under Grant UD230017TD, in part by ARL under Grant W911NF-2020-221, in part by ONR under Grant N000142212305, in part by the NSF under Grants CNS-2212565 and CNS-2146171, and in part by DARPA under Grant D22AP00168-00. This work was partially presented at IEEE INFOCOM 2023 [DOI: 10.1109/INFOCOM53939.2023.10229027]. Recommended for acceptance by J. Ren. *(Corresponding author: Minseok Choi.)*

Dong-Jun Han, David Nickel, Mung Chiang, and Christopher G. Brinton are with Purdue University, West Lafayette, IN 47907 USA (e-mail: han762@purdue.edu; dnickel@purdue.edu; chiang@purdue.edu; cgb@purdue.edu).

Do-Yeon Kim and Jaekyun Moon are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea (e-mail: dy.kim@kaist.ac.kr; jmoon@kaist.edu).

Minseok Choi is with the School of Electronics Engineering, Kyung Hee University, Yongin 17104, South Korea (e-mail: choims@khu.ac.kr).

Digital Object Identifier 10.1109/TMC.2023.3331690

I. Introduction

With the increasing prevalence of mobile and Internet-of Things (IoT) devices, there is an explosion in demand for machine learning (ML) functionality across the intelligent network edge. From the service provider's perspective, providing a high-quality edge-AI service to individual clients, from smartphones to smart cars, is of paramount importance: given newly collected data, the goal of each client is to apply the provided ML model for inference/decisioning. However, during the inference stage (i.e., when training is completed), there are two key requirements that need to be fulfilled to satisfy the client needs in practical edge-AI settings.

A. Key Requirements for Edge-AI

1) Personalization versus generalization: First, during inference, each client should be able to make reliable predictions not only for dominant data classes which have been observed locally, but also occasionally for the classes that have not previously appeared in its local data. We refer to these as a client's main classes and out-of-distribution classes, respectively. Federated learning (FL) [2], [3], [4], the most recently popularized technique for distributing ML across edge devices, has demonstrated a sharp tradeoff between these objectives. In particular, existing works have aimed to create either a *generalized global model* [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] that is tuned to the data distribution across all clients, or *personalized local models* [19], [20], [21], [22], [23] that work case-by-case on each client's individual data. For example, a global activity recognition classifier for wearables learned via FL would be optimized for classes of activities observed over all users. We term the capability to classify all classes as "generalization". The generalized global model is a good option when the input data distribution appearing at each client during inference resembles the global training distribution. However, when the data distributions across clients are significantly non-IID (independent and identically distributed), the globally aggregated FL model may not be the best option for many clients (e.g., consider activity sensors for individuals playing different types of sports). Personalized FL approaches tackle this problem by providing a customized local model to each client based on

their individual local data distributions (e.g., a basketball versus football player). We term this capability to classify the local classes as "personalization".

However, when a client needs to make predictions for classes that are not in its local data (e.g., due to data distribution shift between training and inference), the personalized FL model shows much lower performance than the generalized model, as shown later in Section VI. Hence, it is important to capture both personalization (for handling local classes) and generalization (for handling out-of-distribution classes) in practice where not only the main classes but also the out-of-distribution classes appear occasionally during the inference stage, due to the time-varying testing environment at individual clients.

2) Resource and latency constraints during inference: When training is finished, the trained model should be deployed over the network for efficient inference/decisioning. Mobile edge and IoT devices suffer from limited storage and computation resources. As a result, it is challenging to deploy large-scale models (e.g., neural networks with millions of parameters) at individual clients for inference tasks without incurring significant costs. Deploying the full model at a nearby edge server can be another option, but this approach requires direct transmissions of raw data from the client during inference, which incur noticeable latency depending on the wireless channel condition. Moreover, under this framework, when client models are personalized, the server would need to store all of these variations, which presents scalability challenges. Hence, effectively deploying the trained model over the network (i.e., to satisfy the resource and latency requirements) for efficient inference is of paramount importance.

These two issues are thus significant obstacles to high quality edge-AI services, with existing approaches falling short of addressing them simultaneously. Motivated by this, we pose the following research questions: *How can we achieve both personalization and generalization across resource-constrained edge devices for high-quality inference? Moreover, what is the best inference strategy to satisfy the resource and latency constraints in wireless networks?*

B. Overview of Approach

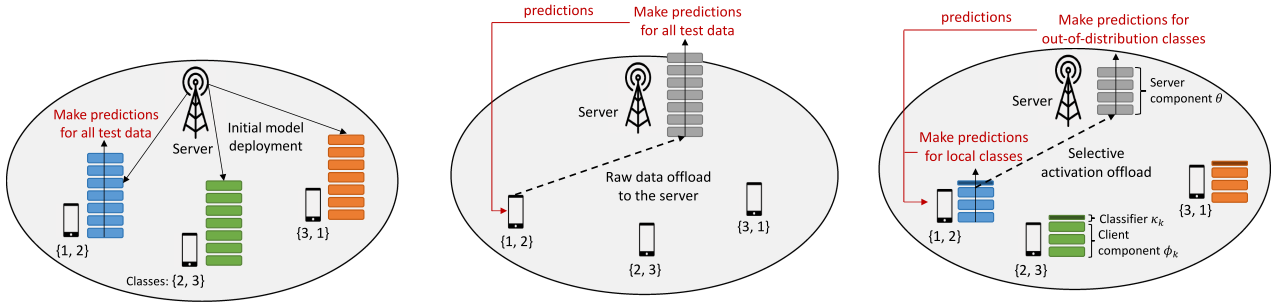
To address these questions, we propose SplitGP, a joint training and inference methodology for generalization and personalization in FL settings. Our key idea is to split the full model into two parts, client-side and server-side, and impose different roles to them. The client-side model should have strong *personalization capability*, where the goal is to work well on each user's local distribution. On the other hand, the server-side model, shared by all clients in the system, should have strong *generalization capability* across the tasks of all users. We achieve this goal based on training with federated split learning (SL). During inference, each client solves its personalized task, i.e., for its main classes, using the client-side model. When the client has to make a prediction that is not related to its personalized task, i.e., for out-of-distribution classes, they can send the output feature from the client-side model to the server-side model, and receive the predicted result back from the edge server. In other words, each client can selectively offload the inference task to the server as in edge computing systems. We will show that this combination of model splitting and edge computing captures both personalization and generalization while reducing the storage, computational load and latency during inference compared to existing methods where the full model is provided to individual clients. SplitGP also has significant advantages in terms of privacy and latency compared to schemes where the full model is deployed at the server-side. Fig. 1 compares the inference stage of SplitGP with these existing frameworks; note that the models in Fig. 1(a)&(b) can be realized through existing FL and SL methodologies and then deployed either at the client-side or at the server-side.

Training in SplitGP proceeds through a series of global rounds, as in FL. In each round, each client first utilizes its local data to compute the client-side loss and the server-side loss. By viewing the full model as a multi-exit neural network, each client conducts model updates to minimize the weighted sum of the client-side and server-side losses. This enables the client-side models not only to perform well on the local task but also to reduce the loss at the server-side. After this model update process, the server-side model components are aggregated as in FL to improve generalization capability. Here we also introduce the notion of a client-side aggregation, where each client augments its local model with information from other client-side models, to converge on meaningful output features for the server-side model while retaining personalization capability. Inference in SplitGP is based on selective inference task offloading to the edge server, which depends on the Shannon entropy value computed at the client-side and the current channel condition. More specifically, each client offloads the task that are uncertain for making the decision using the client-side model, when the channel condition is acceptable.

C. Summary of Contributions

To the best of our knowledge, simultaneous training of client/server-side models with different roles (personalization and generalization) has not been considered before. Existing works in distributed ML also tend to only focus on the training process, without considering the inference at the clients with small storage space and small computing powers. Overall, our contributions are summarized as follows:

- We propose a *joint training and inference* strategy for distributed ML, termed SplitGP, with the following two components: (i) hybrid federated/split learning (i.e., a *training* strategy) for capturing both generalization and personalization, and (ii) selective inference task offloading (i.e., an *inference* strategy) to satisfy the resource and latency constraints at testing-time.
- We analytically characterize the convergence behavior of SplitGP for both strongly convex and non-convex loss functions, showing that training at each client will converge to a stationary or optimal point asymptotically under common assumptions in distributed ML. The result provides insights into selecting the design parameter for training, to control the trade-off between personalization and generalization.
- We conduct a inference time analysis of SplitGP and provide guidelines on selecting the design parameter for inference, to control the amount of the task to be offloaded



(a) Full model at the client-side (b) Full model at the server-side (c) Proposed framework with split models

Fig. 1. *Comparison of inference procedures:* Consider the inference stage when training is completed. Deploying the full model at individual clients [as in Fig. 1(a)] is challenging in resource-constrained edge-AI settings (e.g., mobile/IoT devices) as it induces significant storage and computational burden during inference. When the model is implemented at the server [as in Fig. 1(b)], raw data should be directly transmitted from the client to the server during inference, which incurs various privacy, communication and latency issues. The proposed framework based on model splitting and edge computing in Fig. 1(c) captures both personalization and generalization requirements in resource-constrained scenarios while retaining desirable privacy and latency properties.

to the edge server depending on the network resource availability.

- Experimental results on three real-world datasets show that SplitGP outperforms existing approaches with wide margins of improvement in testing accuracy and inference time. We also implemented SplitGP on Raspberry Pi testbed and confirmed the effectiveness of our algorithm in real-world resource-constrained scenarios.

This paper is an extension of our previous conference paper [1]. Compared with [1], this paper makes the following additional contributions: (i) We provide theoretical guidelines on designing the key parameters of SplitGP, using the convergence analysis and latency analysis results. (ii) We analyze the convergence behavior of SplitGP on strongly convex loss function. The convergence analysis now covers not only the non-convex case but also the strongly-convex case, which holds for many applications such as SVM models. (iii) We propose a U-shaped version of SplitGP to handle the privacy issue that arises from transmitting the label information from each client to the server, which affects neither the test accuracy nor the latency during the inference stage. (iv) Finally, we implemented our algorithm on Raspberry Pi testbed to further validate the effectiveness using real-world resource-constrained devices.

D. Organization

The rest of this paper is organized as follows. We provide the related works in Section II, and describe our SplitGP training and inference strategies in Section III. In Section IV, we analyze the convergence behavior of SplitGP. Inference time analysis in a wireless setup and experimental results are provided in Sections V and VI, respectively. Finally, we draw conclusion in Section VII.

II. Related Works

A. Federated Learning

A large number of FL techniques [4], [5], [6], [7], [8], [9], [15], [16], [17], [18], [24], [25] including FedAvg [4], [26], [27], [28], [29], FedProx [5], FedMA [6], FedDyn [7] and SCAFFOLD [8] have been proposed to construct a shared generalized global model that works well on average for all clients in the system. Personalized FL [19], [20], [21], [22], [23] has been studied more recently tailored to individual clients, through techniques such as multi-task learning [19], interpolation and finetuning [20], meta-learning [21], regularization [23], or personalizing only a specific portion of the layers [30], [31]. A recent work [32] constructs both the global model and the personalized models but use them separately during inference. FL has been also actively studied in wireless networks [10], [11], [33], [34], [35], generally focusing on constructing a shared global model. However, all of these strategies (e.g., personalized FL) do not achieve generalization and personalization simultaneously, and thus are not the best options in practice where not only the main classes and but also the out-of-distribution classes appear occasionally during inference; the global FL model is not personalized to each client's main classes, while personalized FL models are not able to effectively handle the out-of-distribution classes.

B. Split Learning

Recently, SL schemes have been proposed [36], [37], [38], [39], [40], [41], [42] to reduce client-side storage and computation requirements during training compared to FL. The training process of our approach draws from concepts in SL in that we divide the full model into client-side and server-side components. However, existing works on SL including [38], [39], [40] do not focus on capturing both generalization and personalization simultaneously. Compared to existing works, we consider a new inference scenario where the clients should make predictions frequently for the main classes but also occasionally for the out-of-distribution classes, and design a solution tailored to this setup. Prior works also do not provide guidelines on how to perform inference during testing. Compared to existing SL methodologies considering a single model output during inference, we take advantage of multi-exit neural networks and let clients to frequently make predictions using only the client-side model (instead of the full model) while also letting

them to occasionally offload the task to the server, which results in reduced inference time with better test accuracy.

C. Efficient Edge-AI Inference

Several works have focused on the inference stage of the clients at the edge. [43] proposed to deploy distributed deep neural networks on the server and devices during inference, using multi-exit neural networks [44], [45], [46], [47], [48]. Our approach also borrows the concept of multi-exit neural networks with two exits to make predictions both at the client-side and at the server-side. However, these previous works have assumed that the model training phase occurs in a centralized manner, which does not consider the important challenge of non-IID local datasets in FL/SL setups where raw data remains at the devices. In practical settings where each client observes main and out-of-distribution classes during inference, incorporating personalization and generalization results in significant performance enhancements during inference, as we will see in Section VI.

Overall, compared to all previous literatures, the key contribution of this work is to focus on *both training and inference* of FL/SL with distributed clients. In terms of training, we focus on capturing both personalization and generalization to handle not only the main classes but also the out-of-distribution classes, which cannot be achieved via existing personalized FL/SL methods. In terms of inference, we propose to selectively offload the inference task to the server for efficient inference at low-cost client devices.

III. Proposed SplitGP Training & Inference

Let K be the number of clients in the system and D_k be the local dataset of client $k = 1, 2, \ldots, K$ to be used for training an ML model. We consider a setup with resource-constrained clients (e.g., mobile/IoT devices) having limited storage space and computing powers. We denote the *full model* as a parameter vector w, which is split into *client-side* and *server-side* model components ϕ_k and θ, respectively. This model splitting can be performed depending on the client-side storage space. Considering resource-constrained edge devices, we have $|\phi| \ll |\theta|$ in practice. Each client also maintains an *auxiliary classifier* κ_k, with output dimension equal to the number of classes, which enables each client k to make predictions using only ϕ_k and κ_k; as shown in Fig. 1(c), the output of ϕ_k becomes the input of κ_k, and prediction can be made at the output of κ_k. As in FL, model training will proceed in a series of training rounds, which we index $t = 0, 1 \ldots, T-1$.

Before training begins, we split the initialized full model w^0 into $w^0 = [\phi^0, \theta^0]$, and also initialize κ^0. Each client k receives ϕ^0, κ^0 and sets $\phi_k^0 = \phi^0$, $\kappa_k^0 = \kappa^0$, whereas θ^0 is deployed at the server-side. After T global rounds of training, each client k obtains ϕ_k^T and κ_k^T, while the server obtains θ^T. We let

$$v_k^t = \left[\phi_k^t, \kappa_k^t, \theta^t\right] \tag{1}$$

be the model components obtained at client k and the server when global round t is finished.

TABLE I
Summary of Notations

Symbol	Definition
K	The number of clients in the network
T	Total number of global rounds
ϕ_k^t	Model component of client k at round t
κ_k^t	Auxiliary network of client k at round t
θ^t	Server-side model component at round t
v_k^t	Concatenation of ϕ_k^t, κ_k^t and θ^t
$\ell_{C,k}$	Client-side loss computed with client k's local data
$\ell_{S,k}$	Server-side loss computed with client k's local data
D_k	Local dataset of client k
η_t	Learning rate at round t
R	Uplink communication rate during inference
P_C, P_S	Computation powers at the client and the server
$p_k^{(k)}(z)$	Softmax output corresponding to class q on sample z
$E_k(z)$	Shannon entropy computed at client k with test sample z
λ	Our design parameter that controls personalization & generalization
E_{th}	Our design parameter that controls task offloading over the network

Inference Scenario: We consider a scenario having data distribution shift between training and inference in each client: each client should make predictions mainly for the local classes that were already observed in the client's training dataset (personalization) but also occasionally for the out-of-distribution classes (generalization) due to distribution shift. Note that existing works on personalized FL consider the inference scenario with only the local classes, while FL with global model focuses on the case where each class appears uniformly at random during testing. Compared to existing works, we consider a practical setup with frequent local classes and occasional out-of-distribution classes at testing, which is caused by data distribution shift between training and inference.

Goal: As depicted in Fig. 1(c), under this scenario, the goal of the k-th client's model, ϕ_k combined with κ_k, is to make a reliable prediction for local classes in D_k. The goal of each full model, ϕ_k combined with θ, is to make a reliable prediction for all classes in the network-wide dataset, $D = \cup_{k=1}^K D_k$, to handle the out-of-distribution classes of each client. In the following, we describe our SplitGP training strategy (Sections III-A & III-B) and inference strategy (Section III-C) tailored to this setup. The notations used in this paper is summarized in Table I.

A. Multi-Exit Objective Function

Based on the three model components $v = [\phi_k, \kappa_k, \theta]$, we first define the following two losses computed based on D_k.

Client-Side Loss. Given k-th client's local data D_k and $v = [\phi, \kappa, \theta]$, the client-side loss $\ell_{C,k}(v)$ is defined as

$$\ell_{C,k}(v) = \frac{1}{|D_k|} \sum_{x \in D_k} \ell(x; \phi_k, \kappa_k), \tag{2}$$

where $\ell(x; \phi_k, \kappa_k)$ is the loss (e.g., cross-entropy loss) computed with the client model (ϕ_k combined with κ_k) using input data x. The loss in (2) is computed by client k.

Server-Side Loss: We also define the server-side loss $\ell_{S,k}(v)$ computed with the k-th client's local data D_k, as follows:

$$\ell_{S,k}(v) = \frac{1}{|D_k|} \sum_{x \in D_k} \ell(x; \phi_k, \theta), \tag{3}$$

where $\ell(x;\phi_k,\theta)$ is the loss computed at the output of the full model (ϕ_k combined with θ) based on input x. As in existing SL schemes, (3) is computed by the server in SplitGP. To facilitate this, for each $x \in D_k$, the client transmits the output features it computes from ϕ_k along with the label to the server[1].

Proposed Objective Function: In this way, the model ϕ_k maintained at the client-side affects both the client-side loss $\ell_{C,k}(v)$ and the server-side loss $\ell_{S,k}(v)$. By viewing the model $v=[\phi_k,\kappa_k,\theta]$ as a multi-exit neural network [44], [45], [46], [47], [48] with two exits ($\ell_{C,k}$ and $\ell_{S,k}$), we update ϕ_k, κ_k, θ to minimize the weighted sum of client/server-side losses computed with D_k:

$$F_k(v)=\gamma\ell_{C,k}(v)+(1-\gamma)\ell_{S,k}(v). \tag{4}$$

Here, γ and $1-\gamma$ correspond to the weights of the client-side loss and the server-side loss, respectively. If $\gamma=0$, the client-side model is updated only considering the server-side loss, which corresponds to the objective function of SplitFed proposed in [38]. Personalization capability is not guaranteed at the client-side in this case. If $\gamma=1$, the client-side model does not consider the server-side loss at all, which does not guarantee generalization capability at the server-side. In multi-exit network literatures [46], [47], [48], a common choice is to give equal weights to both exits with $\gamma=0.5$. This choice enables the models to provide reliable predictions not only with the client-side model (ϕ_k combined with κ_k) but also with the full model (ϕ_k combined with θ). multi-exist lambda selection discussion

B. SplitGP Training: Personalization & Generalization

Model Update: In the beginning of global round t, we have $v_k^t=[\phi_k^t,\kappa_k^t,\theta^t]$, where ϕ_k^t and κ_k^t are implemented at client k while θ^t is deployed at the server. Based on the proposed objective function (4), the models of client k (ϕ_k^t and κ_k^t) and the shared server-side model θ^t are updated according to

$$\phi_k^{t+1}=\phi_k^t-\eta_t\tilde{\nabla}_\phi F_k(v_k^t), \tag{5}$$

$$\kappa_k^{t+1}=\kappa_k^t-\eta_t\tilde{\nabla}_\kappa F_k(v_k^t), \tag{6}$$

$$\theta_k^{t+1}=\theta^t-\eta_t\tilde{\nabla}_\theta F_k(v_k^t), \tag{7}$$

where η_t is the learning rate at global round t, and

$$\tilde{\nabla}F_k(v_k^t)=\frac{1}{|\tilde{D}_k^t|}\sum_{x\in\tilde{D}_k^t}\left(\gamma\nabla\ell_C(v;x)+(1-\gamma)\nabla\ell_S(v;x)\right) \tag{8}$$

is the stochastic gradient computed with a specific mini-batch $\tilde{D}_k^t\subset D_k$.

Server-Side Model Aggregation: The updated server-side models based on (7) are aggregated according to

$$\theta^{t+1}\leftarrow\sum_{i=1}^{K}\alpha_i\theta_i^{t+1}, \tag{9}$$

to construct a single server-side model, where $\alpha_i=\frac{|D_i|}{\sum_{k=1}^K|D_k|}$ is the relative dataset size. This aggregation process is a natural choice to capture generalization capability at the server using a single model.

Client-Specific Model Aggregation: For client k, ϕ_k combined with κ_k should work well on its local classes (personalized task), while ϕ_k combined with θ should work well on all classes in the system. While the updated ϕ_k using (4) enables the client-side model (ϕ_k combined with κ_k) to have strong personalization capability, it does not guarantee the generalization performance of the full model (ϕ_k combined with θ_k). In particular, since ϕ_k is updated mainly with the local data D_k of client k, the output of ϕ_k (which becomes the input of θ in the full model) does not provide meaningful output features for the classes outside of client k's local dataset D_k.

A natural way to resolve this issue would be to aggregate ϕ_k^{t+1} for all k as $\sum_{i=1}^K\alpha_i\phi_i^{t+1}$, and deploy this aggregated model at each client. However, this can reduce the personalization capability at each client. In order to capture personalization while providing a meaningful result to the server-side model θ, in SplitGP, each client k computes the weighted sum of ϕ_k^{t+1} and the average of ϕ_k^{t+1} for all $k=1,2,\ldots,K$, as follows:

$$\phi_k^{t+1}\leftarrow\lambda\phi_k^{t+1}+(1-\lambda)\sum_{i=1}^{K}\alpha_i\phi_i^{t+1}. \tag{10}$$

Here, $\lambda\in[0,1]$ is a parameter that controls the weights for personalization and generalization. If $\lambda=1$, the client-side model has a strong personalization capability but does not provide a meaningful output feature to the server-side model since the client-side models are updated independently without aggregation. If $\lambda=0$, the client-side model provides a generalizable feature to the server but lacks personalization capability, since all clients share the same components in this case. Using λ, the auxiliary classifiers $\{\kappa_k\}_{k=1}^K$ are also aggregated at each client k according to

$$\kappa_k^{t+1}\leftarrow\lambda\kappa_k^{t+1}+(1-\lambda)\sum_{i=1}^{K}\alpha_i\kappa_i^{t+1}, \tag{11}$$

which enables the client-side model to make reliable predictions on the out-of-distribution classes. Although generalization is not the main goal of the client model, conducting inference for out-of-distribution classes at the client when possible will further reduce communication cost and latency. Moreover, during inference, the client does not automatically know whether a datapoint is from one of its main classes or not. We therefore introduce a confidence threshold for test-time in Section III-C which chooses between client and server-side inference.

The left part of Fig. 2 summarizes the loss computation process, the model update procedure, and the model aggregation step during SplitGP training. Note that, for simplicity of presentation, we have presented the model updates in (5), (6), (7) assuming a single gradient step at each time t. In practice, these can be repeated multiple times in-between each model aggregation process.

[1]Potential privacy issues can be handled by adding a noise layer [49] at the client as in [38], to construct private/noisy versions of output features. More detailed discussions regarding the privacy issue of SL can be found in [38].

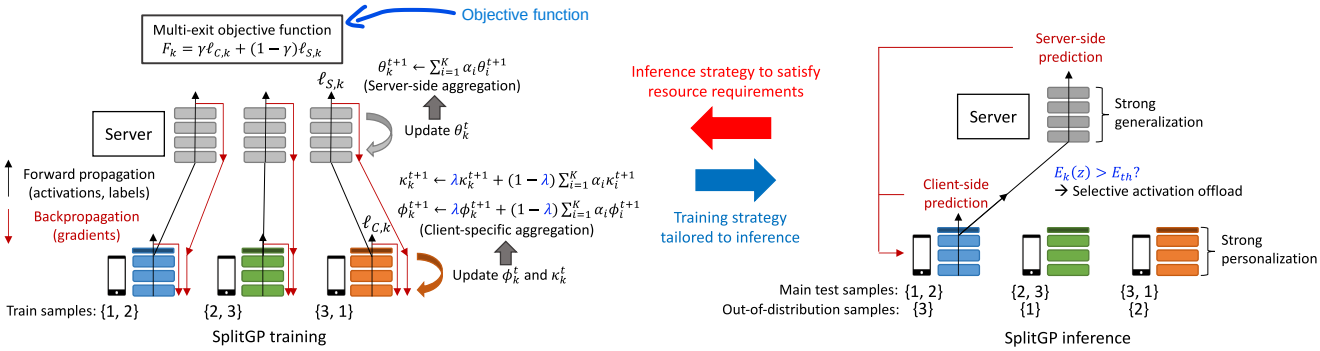


Fig. 2. *Overall procedure of SplitGP training (left) and inference (right):* Training and inference strategies are jointly designed for efficient edge-AI over the network. To satisfy the resource and latency constraints during inference, SplitGP controls the amount of offloaded inference task over the network. Tailored to this inference strategy, the training strategy of SplitGP is designed to capture personalization capability at the client-side and generalization capability at the server-side.

After repeating the overall process for T global rounds, we obtain K different personalized models $\{\phi_k^T\}_{k=1}^K$ and auxiliary classifiers $\{\kappa_k^T\}_{k=1}^K$, and one server model θ^T which are deployed over the network at testing time; ϕ_k^T and κ_k^T are deployed at client k while θ^T is implemented at the edge server for efficient inference. The overall training process of SplitGP is also summarized in Algorithm 1.

C. SplitGP Inference: Selective Task Offloading

During inference, given the trained models ϕ_k^T, κ_k^T and θ^T, each client k must determine whether to rely on the client-side (ϕ_k^T, κ_k^T) or server-side (ϕ_k^T, θ^T) model. Given a test sample z at client k, the Shannon entropy is first computed using the client-side model (ϕ_k^T, κ_k^T) as

$$E_k(z) = -\sum_{q=1}^{Q} p_k^{(q)}(z) \log p_k^{(q)}(z), \tag{12}$$

where Q is the total number of classes in the system and $p_k^{(q)}(z)$ is the softmax output for class q on sample z, using the model deployed at client k. This value shows how much the client-side model is certain about the prediction of z. If

$$E_k(z) \leq E_{th} \tag{13}$$

holds for a desired entropy threshold E_{th}, the inference is made at the client-side. Otherwise, if the prediction is not certain enough (i.e., if $E_k(z) > E_{th}$), the output feature of ϕ_k^T computed on sample z is sent to the server and the output of the server model θ^T is used for inference, as in Fig. 1(c). The value of E_{th} in (13) is therefore a control parameter for the amount of communication over the network during inference. The inference stage of our SplitGP is described in the right part of Fig. 2 and in Algorithm 1.

Remark 1: If the client-side component ϕ_k is fully personalized (e.g., by setting $\lambda = 1$ or via fine-tuning), ϕ_k does not provide meaningful features to the server-side model θ_k for the out-of-distribution samples. The aggregation in (10) without further fine-tuning enables the full model (ϕ_k combined with θ_k) to make reliable predictions for the out-of-distributions samples.

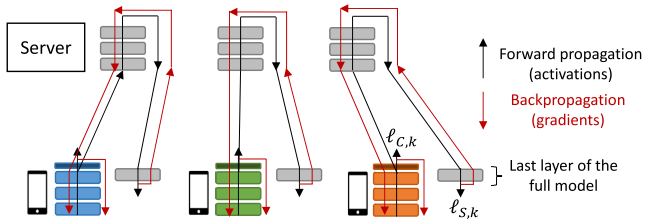


Fig. 3. U-shaped version of SplitGP training process to avoid label transmission to the server. By letting each client to have the last layer of the full model, the clients can directly compute the loss without sending the labels to the server. This training process can replace the left part of Fig. 2 when label information turns out to be critical.

Remark 2: We reiterate that model splitting $w = [\phi, \theta]$ can be done depending on the storage space of the clients. In resource-constrained scenarios with mobile/IoT devices, the size of the client-side component ϕ is significantly smaller than that of θ. Considering this practical setup, we set ϕ to have 10% of model parameters compared to the full model w when performing experiments in Section VI.

D. Label Transmission to the Server

We note that existing works on SL and our SplitGP require each client to transmit its label information to the server during training, which can cause privacy issue in specific applications. This can be handled by adopting U-shaped SL [36] where the last layer of the overall model is also trained at the client-side. Fig. 3 describes how model update can be conducted in SplitGP by adopting U-shaped SL, to replace the process in the left part of Fig. 2. Since the last layer of the full model is at the client-side, each client can compute the loss without sending the labels to the server. Here, the server is still responsible for training the majority of the layers (except the first few layers and the last layer), preserving the advantage of SL. After the model update process, the models are aggregated according to Lines 12, 13, 14 in Algorithm 1. Here, the last layer as well as the client-side components are broadcasted to the client for the next round.

Algorithm 1: SplitGP: Training and Inference.

Training Phase

Input: Initialized models $v^0 = [\phi^0, \kappa^0, \theta^0]$
Output: $v_k^T = [\phi_k^T, \kappa_k^T, \theta^T]$ for each client $k = 1, 2, \ldots, K$
for each global round $t = 0, 1, \ldots, T-1$ **do**
  for $k \in \{1, 2, \ldots, K\}$ **in parallel do**
    $\ell_{C,k}(v_k^t) = \frac{1}{|\tilde{D}_k^t|} \sum_{x \in \tilde{D}_k^t} \ell(x; \phi_k^t, \kappa_k^t)$
    // Client-side loss computed with $[\phi_k^t, \kappa_k^t]$
    $\ell_{S,k}(v_k^t) = \frac{1}{|\tilde{D}_k^t|} \sum_{x \in \tilde{D}_k^t} \ell(x; \phi_k^r, \theta^t)$
    // Server-side loss computed with $[\phi_k^t, \phi^t]$
    $F_k(v_k^t) = \gamma \ell_{C,k}(v_k^t) + (1-\gamma)\ell_{S,k}(v_k^t)$
    // Multi-exit objective function
    $\phi_k^{t+1} = \phi_k^t - \eta_t \tilde{\nabla}_\phi F_k(v_k^t)$
    $\kappa_k^{t+1} = \kappa_k^t - \eta_t \tilde{\nabla} F_k(v_k^t)$
    $\theta_k^{t+1} = \theta_k^t - \eta_t \tilde{\nabla}_\theta F_k(v_k^t)$ // Model update
  end for
  $\theta^{t+1} \leftarrow \sum_{i=1}^K \alpha_i \theta_i^{t+1}$
  // Server model aggregation
  $\phi_k^{t+1} \leftarrow \lambda \phi_k^{t+1} + (1-\lambda) \sum_{i=1}^K \alpha_i \phi_i^{t+1}$
  $\kappa_k^{t+1} \leftarrow \lambda \kappa_k^{t+1} + (1-\lambda) \sum_{i=1}^K \alpha_i \kappa_i^{t+1}$
  // Client-side model aggregations; λ controls the weights for personalization and generalization
end for
$v_k^T = [\phi_k^T, \kappa_k^T, \theta^T]$

Inference Phase

Input: Test sample z at client k with $v_k^T = [\phi_k^T, \kappa_k^T, \theta^T]$
Output: Prediction result for test sample z
$E_k(z) = -\sum_{q=1}^Q p_k^{(q)}(z) \log p_k^{(q)}(z)$
if $E_k(z) < E_{th}$ **then**
  Make prediction with ϕ_k^T combined with κ_k^T
else
  Make prediction with ϕ_k^T combined with θ^T
end if

This training process requires additional communication load between the server and the clients during training, which is the cost for handling the label transmission issue.

Remark 3: The U-shaped version of SplitGP training produces exactly the same model as in the original process in the left-side of Fig. 2. Moreover, U-shaped approach only needs to be applied during training. When training is finished, the model is deployed as in the our original inference strategy shown in the right part of Fig. 2. Hence, the U-shaped approach affects neither the test accuracy nor the latency during the inference stage, which is a significant advantage.

E. Design Parameters in SplitGP

Overall, SplitGP is a solution that tackles both training and inference stages, with two design parameters: λ in (10) and (11) is the hyperparameter that controls the amount of personalization and generalization during training, while E_{th} in (13) is the hyperparameter that manages the amount of communication over the network during inference. In the following, we first analyze the convergence behavior of SplitGP and provide insights into the effect of λ, in Section IV. Then in Section V, we provide guidelines on selecting E_{th} to satisfy the inference time constraint depending on the current channel condition and other system parameters. The effects of λ and E_{th} are also analyzed via experiments in Section VI.

IV. SplitGP Convergence Analysis

In this section, we analyze the convergence behavior of SplitGP based on some standard assumptions in FL [50], [51], [52].

Assumption 1: For each k, $F_k(v)$ is L-smooth, i.e., $\|\nabla F_k(u) - \nabla F_k(v)\| \leq L\|u - v\|$ for any u and v.

Assumption 2: For each k, the expected squared norm of stochastic gradient is bounded, i.e., $\mathbb{E}[\|\tilde{\nabla} F_k(v)\|^2] \leq G$.

Assumption 3: The variance of the stochastic gradient of D_k is bounded, i.e., $\mathbb{E}[\|\nabla F_k(v) - \tilde{\nabla} F_k(v)\|^2] \leq \sigma_k^2$.

Assumption 1 is one of the most common assumptions in FL [50], [51] that holds for linear/logistic regression models and neural networks with sigmoid activations. Assumptions 2 and 3 are also adopted in various FL literatures [10], [50]. We define the global loss function $F(v)$ as

$$F(v) = \frac{1}{K} \sum_{k=1}^K F_k(v), \tag{14}$$

which is the average of the losses defined in (4). We show that our algorithm converges to a stationary point of (14), which guarantees the generalization capability of SplitGP while including personalization through λ for any non-convex ML loss function $F(v)$.

A. Main Theorem and Discussions

The following theorem gives the convergence behavior of our SplitGP training process for non-convex loss functions.

Theorem 1: (SplitGP Convergence: Non-Convex Case) Let $\eta_t = \frac{\eta_0}{a+t}$, where $a = \frac{c+4}{1-\lambda^2}$ for some constant $c > 0$. Suppose that η_0 is chosen to satisfy $\eta_t \leq \frac{1}{2L}$. Under Assumptions 1, 2, 3, SplitGP model training converges to the stationary point as

$$\frac{1}{\Gamma_T} \sum_{t=0}^{T-1} \sum_{k=1}^K \frac{\eta_t}{4K} \mathbb{E}\left[\|\nabla F(v_k^t)\|^2\right] \leq \frac{F(v^0) - F^*}{\Gamma_T} + \frac{L \sum_{k=1}^K \sigma_k^2}{K} \left(\frac{1}{\Gamma_T} \sum_{t=0}^{T-1} \eta_t^2\right) + \epsilon(\lambda) \left(\frac{1}{\Gamma_T} \sum_{t=0}^{T-1} \eta_t^3\right), \tag{15}$$

where

$$\epsilon(\lambda) = \frac{16(c+4)G^2 L^2 \lambda^2 (2-\lambda^2)}{c(1-\lambda^2)^2}, \tag{16}$$

$\Gamma_T = \sum_{t=0}^{T-1} \eta_t$ and F^* is the minimum value of $F(v)$ in (14).

Proof: See Section IV-B.

Here, $\epsilon(\lambda)$ is the term specific to our work, arising from the joint consideration of generalization and personalization. By setting $\eta_t = \frac{\eta_0}{a+t}$, we have $\Gamma_T = \sum_{t=0}^{T-1} \eta_t \to \infty$ as T grows,

and $\sum_{t=0}^{\infty}\eta_t^2<\infty$, $\sum_{t=0}^{\infty}\eta_t^3<\infty$. Hence, for any $\lambda\in[0,1)$, the upper bound in (15) goes to 0 as T grows. Thus, we have $\min_{t\in\{0,1,\ldots,T-1\}}\mathbb{E}[\|\nabla F(v_k^t)\|]\overset{T\to\infty}{\longrightarrow}0$ for all $k=1,\ldots,K$, which guarantees convergence to a stationary point of (14).

Theorem 1 indicates that $v_k^t=[\phi_k^t,\kappa_k^t,\theta^t]$, which has a certain amount of personalization capability from λ, also obtains the generalization capability of (14). In other words, both personalization and generalization are achieved. Here, λ controls the trade-off between personalization and generalization; as λ grows, a larger number of global rounds is required to reduce the upper bound in (15). This is the cost for achieving a stronger personalization at the client-side. Given the number of global rounds T and learning rates η_t, the first two terms of (16) are fixed values, while the third term $\epsilon(\lambda)(\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t^3)$ is controllable by λ. Let δ be the constraint that we expect $\epsilon(\lambda)(\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t^3)$ to be smaller than, i.e.,

$$\epsilon(\lambda)\left(\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t^3\right)\le\delta, \tag{17}$$

to guarantee a certain amount of generalization capability. It can be easily seen that $\epsilon(\lambda)$ is an increasing function of λ within the range [0,1). Hence, we can obtain the condition

$$\lambda\le\lambda^{(\delta)}, \tag{18}$$

where $\lambda^{(\delta)}$ is the solution of $\epsilon(\lambda)(\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t^3)=\delta$ which can be obtained via a bisection search method. In other words, we must select λ within the modified range $[0,\min\{\lambda^{(\delta)},1\})$ to satisfy the generalization constraint.

Note that the case with $\lambda=1$ does not guarantee convergence, since the client-side models are constructed independently without aggregation. On the other hand, the case with $\lambda=0$ reduces to the bound of conventional FL with generalization since all the clients share the same ϕ and κ. Later in Section VI, we also observe via experiments that λ controls the trade-off between personalization and generalization with varying portion of out-of-distribution test samples relative to the main test samples.

B. Convergence Proof

Using $v_k^t=[\phi_k^t,\kappa_k^t,\theta^t]$, we first define v^t as:

$$v^t=\frac{1}{K}\sum_{k=1}^{K}v_k^t. \tag{19}$$

By the L-smoothness of $F(v)$ and taking the expectation of both sides, we have

$$\mathbb{E}\left[F\left(v^{t+1}\right)\right]-\mathbb{E}\left[F\left(v^t\right)\right]\le\underbrace{\mathbb{E}\left[\left\langle\nabla F(v^t),v^{t+1}-v^t\right\rangle\right]}_{A}+\underbrace{\frac{L}{2}\mathbb{E}\left[\|v^{t+1}-v^t\|^2\right]}_{B}. \tag{20}$$

In the following, we first bound A, and then bound B.

Step 1. Bounding A: We start by rewriting A as

$$A\underset{(a)}{=}-\eta_t\mathbb{E}\left[\left\langle\nabla F(v^t),\frac{1}{K}\sum_{k=1}^{K}\tilde{\nabla}F_k(v_k^t)\right\rangle\right] \tag{21}$$

$$\underset{(b)}{=}-\eta_t\mathbb{E}\left[\left\langle\nabla F(v^t),\frac{1}{K}\sum_{k=1}^{K}\nabla F_k(v_k^t)\right\rangle\right] \tag{22}$$

$$\underset{(c)}{=}\underbrace{-\frac{\eta_t}{2}\mathbb{E}\left[\|\nabla F(v^t)\|^2\right]}_{A_1}-\frac{\eta_t}{2}\mathbb{E}\left[\left\|\frac{1}{K}\sum_{k=1}^{K}\nabla F_k(v_k^t)\right\|^2-\underbrace{\left\|\nabla F(v^t)-\frac{1}{K}\sum_{k=1}^{K}\nabla F_k(v_k^t)\right\|^2}_{A_2}\right], \tag{23}$$

where (a) comes from $v^{t+1}-v^t=-\eta_t\frac{1}{K}\sum_{k=1}^{K}\tilde{\nabla}F_k(v_k^t)$, (b) follows from taking the expectation for the mini-batch, and (c) is obtained by utilizing $\|z_1-z_2\|^2=\|z_1\|^2+\|z_2\|^2-2\langle z_1,z_2\rangle$.

We now focus on A_1. We can write

$$\|\nabla F(v^t)\|^2\underset{(d)}{\ge}\frac{1}{2}\|\nabla F(v_k^t)\|^2-\|\nabla F(v_k^t)-\nabla F(v^t)\|^2$$

$$=\frac{1}{2}\|\nabla F(v_k^t)\|^2-\left\|\frac{1}{K}\sum_{i=1}^{K}(\nabla F_i(v_k^t)-\nabla F_i(v^t))\right\|^2$$

$$\underset{(e)}{\ge}\frac{1}{2}\|\nabla F(v_k^t)\|^2-L^2\|v_k^t-v^t\|^2 \tag{24}$$

for any k. Here, (d) comes from using $\|a+b\|^2\le2\|a\|^2+2\|b\|^2$ and (e) comes from L-smoothness. Thus, we can bound A_1 as follows:

$$A_1=-\frac{\eta_t}{2}\mathbb{E}\left[\|\nabla F(v^t)\|^2\right]$$

$$=-\frac{\eta_t}{2K}\sum_{k=1}^{K}\mathbb{E}\left[\|\nabla F(v^t)\|^2\right]$$

$$\le-\frac{\eta_t}{4K}\sum_{k=1}^{K}\mathbb{E}\left[\|\nabla F(v_k^t)\|^2\right]+\frac{\eta_t L^2}{2K}\sum_{k=1}^{K}\mathbb{E}[\|v_k^t-v^t\|^2]. \tag{25}$$

For A_2, we have

$$\frac{\eta_t}{2}\mathbb{E}[A_2]=\frac{\eta_t}{2}\mathbb{E}\left[\left\|\frac{1}{K}\sum_{k=1}^{K}(\nabla F_k(v^t)-\nabla F_k(v_k^t))\right\|^2\right]$$

$$\underset{(f)}{\le}\frac{\eta_t}{2K}\sum_{k=1}^{K}\mathbb{E}[\|\nabla F_k(v^t)-\nabla F_k(v_k^t)\|^2]$$

$$\underset{(g)}{\le}\frac{\eta_t L^2}{2K}\sum_{k=1}^{K}\mathbb{E}[\|v^t-v_k^t\|^2], \tag{26}$$

where (f) holds due to the convexity of $\|\cdot\|^2$ and (g) holds due to the L-smoothness assumption.

Step 2. Bounding B: Now we bound the term B. Recall that from (14), the following holds:

$$v^{t+1} - v^t = -\eta_t \frac{1}{K} \sum_{k=1}^{K} \tilde{\nabla} F_k(v_k^t). \tag{27}$$

Based on this result, we can write

$$B \leq \eta_t^2 L \left(\mathbb{E}\left[\| \frac{1}{K} \sum_{k=1}^{K} \nabla F_k(v_k^t) \|^2 \right] \right. \tag{28}$$

$$\left. + \mathbb{E}\left[\| \frac{1}{K} \sum_{k=1}^{K} \nabla F_k(v_k^t) - \frac{1}{K} \sum_{k=1}^{K} \tilde{\nabla} F_k(v_k^t) \|^2 \right] \right), \tag{29}$$

where

$$\mathbb{E}\left[\left\| \frac{1}{K} \sum_{k=1}^{K} \nabla F_k(v_k^t) - \frac{1}{K} \sum_{k=1}^{K} \tilde{\nabla} F_k(v_k^t) \right\|^2 \right] \tag{30}$$

$$\leq \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}[\| \nabla F_k(v_k^t) - \tilde{\nabla} F_k(v_k^t) \|^2]$$

$$\underset{(h)}{\leq} \frac{1}{K} \sum_{k=1}^{K} \sigma_k^2 \tag{31}$$

and (h) results from Assumption 3.

By inserting the bounds of A and B to (20), and by employing a learning rate that satisfies $\eta_t \leq \frac{1}{2L}$, we obtain

$$\frac{\eta_t}{4K} \sum_{k=1}^{K} \mathbb{E}\left[\| \nabla F(v_k^t) \|^2 \right] \leq \mathbb{E}[F(v^t)] - \mathbb{E}[F(v^{t+1})] \tag{32}$$

$$+ \frac{\eta_t^2 L}{K} \sum_{k=1}^{K} \sigma_k^2 + \underbrace{\frac{\eta_t L^2}{K} \sum_{k=1}^{K} \mathbb{E}[\| v_k^t - v^t \|^2]}_{C}. \tag{33}$$

Step 3. Bounding C: To bound C, we define

$$\psi_k^t = \left[\phi_k^t, \kappa_k^t \right] \tag{34}$$

$$\hat{\psi}_k^{t+1} = \psi_k^t - \eta_t \tilde{\nabla}_\psi F_k(v_k^t), \tag{35}$$

$\psi^t = \frac{1}{K} \sum_{k=1}^{K} \psi_k^t$ and $\hat{\psi}^t = \frac{1}{K} \sum_{k=1}^{K} \hat{\psi}_k^t$. Then, we have

$$\psi_k^{t+1} = \lambda \hat{\psi}_k^{t+1} + (1-\lambda) \hat{\psi}^{t+1} \tag{36}$$

and $\frac{1}{K} \sum_{k=1}^{K} \| v_k^t - v^t \|^2 = \frac{1}{K} \sum_{k=1}^{K} \| \psi_k^t - \psi^t \|^2$. We can write

$$\frac{1}{K} \sum_{k=1}^{K} \| \psi_k^t - \psi^t \|^2 = \frac{1}{K} \sum_{k=1}^{K} \| (\psi_k^t - \hat{\psi}^t) - (\psi^t - \hat{\psi}^t) \|^2$$

$$\underset{(i)}{\leq} \frac{1}{K} \sum_{k=1}^{K} \| \psi_k^t - \hat{\psi}^t \|^2 \underset{(j)}{=} \frac{1}{K} \sum_{k=1}^{K} \| \lambda (\hat{\psi}_k^t - \hat{\psi}^t) \|^2$$

$$= \frac{\lambda^2}{K} \sum_{k=1}^{K} \| (\hat{\psi}_k^t - \psi^{t-1}) - (\hat{\psi}^t - \psi^{t-1}) \|^2$$

$$\underset{(k)}{\leq} \frac{\lambda^2}{K} \sum_{k=1}^{K} \| \hat{\psi}_k^t - \psi^{t-1} \|^2$$

$$\underset{(l)}{=} \frac{\lambda^2}{K} \sum_{k=1}^{K} \| - \eta_{t-1} \tilde{\nabla}_\psi F_k(v_k^{t-1}) + (\psi_k^{t-1} - \psi^{t-1}) \|^2 \tag{37}$$

where (i) and (k) come from $\mathbb{E}[\|z - \mathbb{E}[z]\|^2] \leq \mathbb{E}[\|z\|^2]$, (j) follows from (36) and (l) results from $\hat{\psi}_k^{t+1} = \psi_k^t - \eta_t \tilde{\nabla}_\psi F_k(v_k^t)$. Now following the proof of Lemma 4 of [52] and utilizing Assumption 2, when $\eta_t = \frac{\eta_0}{a+t}$ and $a = \frac{c+4}{1-\lambda^2}$ for some constant $c > 0$, we obtain the following result:

$$\frac{1}{K} \sum_{k=1}^{K} \mathbb{E}\left[\| v_k^t - v^t \|^2 \right] \leq \frac{16(c+4) G^2 \lambda^2 (2-\lambda^2)}{c(1-\lambda^2)^2}. \tag{38}$$

Step 4. Telescoping sum: Finally, after inserting the result of Step 3 into (32), we have

$$\frac{\eta_t}{4K} \sum_{k=1}^{K} \mathbb{E}\left[\| \nabla F(v_k^t) \|^2 \right] \leq \mathbb{E}[F(v^t)] - \mathbb{E}[F(v^{t+1})] \tag{39}$$

$$+ \frac{\eta_t^2 L}{K} \sum_{k=1}^{K} \sigma_k^2 + \frac{16 \eta_t^3 (c+4) G^2 \lambda^2 (2-\lambda^2)}{c(1-\lambda^2)^2}. \tag{40}$$

After summing up for $t = 0, 1 \ldots, T-1$ and dividing both sides by $\Gamma_T = \sum_{t=0}^{T-1} \eta_t$, with some manipulations, we obtain (15). This completes the proof of Theorem 1.

C. Strongly Convex Case

In this subsection, we analyze the convergence behavior of the strongly convex loss function (e.g., logistic regression and SVM models). We consider the following assumptions for the proof as in [9], [21].

Assumption 4: For each k, $F_k(v)$ is μ strongly convex i.e., $\langle \nabla F_k(u) - \nabla F_k(v), u - v \rangle \geq \mu \| u - v \|$ holds for any u and v.

Assumption 5: For each k, the diversity of gradient is bounded, i.e., $[\| \nabla \bar{F}(v) - \nabla F_k(v) \|] \leq \varsigma^2$.

Assumption 4 is about the strong convexity, while Assumption 5 is about the bounded gradient diversity, which is more practical than Assumption 2 on the bounded gradient. Now we state the following theorem.

Theorem 2: (SplitGP Convergence: Strongly Convex Case) Define $v^t = \frac{1}{K} \sum_{k=1}^{K} v_k^t$. Let $\eta_t = \frac{\eta_0}{\sqrt{1+t}}$ and $\sigma = \max_{1 \leq k \leq K} \sigma_k$. Under Assumptions 1, 3, 4, 5 SplitGP model training satisfies

$$\begin{bmatrix} \mathbb{E}\left[\left\| v^T - v^* \right\|^2 \right] \\ \mathbb{E}\left[\left\| V^T - \mathbf{1}_K \cdot v^T \right\|^2 \right] \end{bmatrix} \leq \frac{1}{2\mu\eta_0} \mathcal{O}\left(\frac{1}{\sqrt{T}} \right)$$

$$\times \begin{bmatrix} \mathbb{E}\left[\left\| v^0 - v^* \right\|^2 \right] \\ \mathbb{E}\left[\left\| V^0 - \mathbf{1}_K \cdot v^0 \right\|^2 \right] \end{bmatrix}$$

$$+ \left\{ \frac{1}{2\mu\eta_0} \mathcal{O}\left(\frac{1}{\sqrt{T}} \right) + \frac{\eta_0}{2} \mathcal{O}\left(\frac{\ln T}{\sqrt{T}} \right) + \eta_0^2 \mathcal{O}\left(\frac{1}{T} \right) \right\}$$

$$\times \begin{bmatrix} \frac{\sigma^2}{K} \\ 4E\mathcal{O}(\varsigma^2) + 2K\sigma^2 \end{bmatrix}, \tag{41}$$

where $V^t = [(v_1^t)^\top \quad \cdots \quad (v_K^t)^\top]^\top$ is a matrix with stacked row vectors $\{v_k^t\}_{k=1}^K$, and v^* is the model that minimizes (14).

Proof: By defining the client-side parameters as $\psi_k^t = [\phi_k^t, \kappa_k^t]$, we can write

$$\psi_k^{t+1} = \lambda \cdot \left(\psi_k^t - \eta_t \tilde{\nabla}_\psi F_k(\psi_k^t)\right) + (1-\lambda) \sum_{i=1}^K \alpha_i \cdot (\psi_i^t - \eta_t \tilde{\nabla}_\psi F_i(\psi_i^t)) \tag{42}$$

$$= \sum_{i=1}^K m_{k,i} \psi_i^t - \eta_t \sum_{i=1}^K m_{k,i} \tilde{\nabla}_\psi F_i(\psi_i^t) \tag{43}$$

where $\{m_{k,i}\}_{1\le k,i\le K}$ are the values that satisfy $\sum_{k=1}^K m_{k,i} = 1$, $\sum_{i=1}^K m_{k,i} = 1$. Now define a doubly-stochastic and symmetric matrix $M = [m_{k,i}]_{1\le k,i\le K}$. We also define $\Psi^t = [(\psi_1^t)^\top \quad \cdots \quad (\psi_K^t)^\top]^\top$ by stacking the row vectors ψ_k^t. Then, we have $\Psi^{t+1} = M\Psi^t - \eta_t M \tilde{\partial} F(\Psi^t)$, which can be rewritten as

$$\Psi^{t+1} = M^{t+1}\Psi^0 - \sum_{j=0}^t \eta_j M^{t+1-j} \tilde{\partial} F(\Psi^j), \tag{44}$$

where $\tilde{\partial} F(\Psi^t) = [(\tilde{\nabla} F_1(\psi_1^t))^\top, \quad \cdots \quad, (\tilde{\nabla} F_K(\psi_K^t))^\top]$. Moreover, by defining $\psi^t = \frac{1}{K}\sum_{k=1}^K \psi_k^t$, we obtain

$$\psi^{t+1} = \frac{1}{K}\mathbf{1}_K^\top \Psi^{t+1} \underset{(a)}{=} \psi^0 - \frac{1}{K}\sum_{j=0}^t \eta_j \sum_{k=1}^K \tilde{\nabla} F_k(\psi_k^j). \tag{45}$$

where (a) comes from applying $\sum_{k=1}^K m_{k,i} = 1$, $\sum_{i=1}^K m_{k,i} = 1$ to (44). Recall that the server-side parameters θ^t are shared across all clients. By adopting the similar derivation, we obtain

$$\theta^{t+1} = \frac{1}{K}\mathbf{1}_K^\top \Theta^{t+1} = \theta^0 - \frac{1}{K}\sum_{j=0}^t \eta_j \sum_{k=1}^K \tilde{\nabla} F_k(\theta_k^j), \tag{46}$$

where $\Theta^t = [(\theta_1^t)^\top \quad \cdots \quad (\theta_K^t)^\top]^\top$ and $\theta_k^t = \theta^t$ for all $k \in \{1, 2, \ldots, K\}$. Finally, by combining (45) and (46), we have

$$v^{t+1} = \frac{1}{K}\mathbf{1}_K^\top V^{t+1} = v^0 - \frac{1}{K}\sum_{j=0}^t \eta_j \sum_{k=1}^K \tilde{\nabla} F_k(v_k^j), \tag{47}$$

Now by adopting the result of Theorem 2 of [53], we complete the proof.

From the upper bound of $\mathbb{E}[\|V^T - \mathbf{1}_K v^T\|^2]$, it can be seen that each model theoretically converges to the average of models. Moreover, the upper bound of $\mathbb{E}[\|v^T - v^*\|^2]$ shows that the average of models v^t converges to the optimal model v^*. From these two, Theorem 2 indicates that all models converge to the optimal model of v^* that minimizes (14), which theoretically guarantees generalization capability of the models for the strongly convex case. Compared to the non-convex loss function considered in Theorem 1, in the strongly convex case, the convergence to the optimal point can be directly guaranteed with a milder learning rate constraint. More specifically, the learning rate should be decayed faster to guarantee the convergence to the stationary point in the non-convex case. We also note that the bounded gradient assumption (Assumption 2) is also not required when guaranteeing the convergence for the strongly convex case.

V. SplitGP Inference Time Analysis in Wireless Networks

Given the trained models $v_k^T = [\phi_k^T, \kappa_k^T, \theta^T]$ for each $k = 1, 2, \ldots, K$, recall that SplitGP selectively offloads the inference task to the server over the network based on the E_{th} value. In this section, we analyze the storage, computation, communication and time required during the inference stage, and provide guidelines on selecting the E_{th} parameter of SplitGP to satisfy the inference time constraint depending on various system parameters in wireless networks.

A. System Model

Let P_C and P_S be the available computing powers of each client and the server, respectively. Let $|\phi|$, $|\theta|$, $|\kappa|$ be the numbers of parameters of ϕ, θ, κ, respectively. Considering $P_C \ll P_S$ in practice, we split the model $w = [\phi, \theta]$ such that the size of client-side component ϕ is significantly smaller than the server-side component θ, i.e., $|\phi| \ll |\theta|$. Moreover, κ is assumed to be a small classifier satisfying $|\kappa| \ll |\phi|$ and $|\kappa| \ll |\theta|$. The inference time is the value that is affected by the model size. To make our analysis tractable, we assume that the inference time is proportional to the number of parameters of the model [40], [54]. For example, given a model with $|\phi| + |\kappa|$ parameters and a test dataset of size $|D|$, the inference time at the client will be proportional to $\frac{(|\phi|+|\kappa|)|D|}{P_C}$. One may also consider different latency models which is beyond the scope of this paper. During inference, we consider orthogonal frequency division multiple access (OFDMA) to serve each client using a fixed frequency band without interference [11], [34]. By letting p_k be the transmit power of client k, the uplink communication rate R_k between the server and client k can be written as

$$R_k = b_k \log_2\left(1 + \frac{p_k |h_k|^2}{b_k N_0}\right), \tag{48}$$

where h_k denotes the channel coefficient between client k and the server, b_k is the bandwidth allocated to client k, and N_0 is the noise power density. Finally, q_c denotes the dimension of the cut-layer (i.e., output dimension of ϕ) and q denotes the size of the test sample (input dimension of ϕ).

B. Resource and Latency Analysis

Table II compares our methodology with existing frameworks at the inference stage. By adopting OFDMA, we focus on a specific client and omit the index k in this subsection for notational simplicity. We first present the derivations of two baseline inference frameworks.

TABLE II
RESOURCES REQUIRED AND LATENCY INCURRED AT EACH CLIENT DURING INFERENCE: COMPARING SPLITGP WITH BASELINES

Methods	Storage	Computation	Communication	Inference time
Full model at the server-side	0	0	$q\|D\|$	$\frac{q\|D\|}{R} + \frac{(\|\phi\|+\|\theta\|)\|D\|}{P_S}$
Full model at the client-side	$\|\phi\|+\|\theta\|$	$(\|\phi\|+\|\theta\|)\|D\|$	0	$\frac{(\|\phi\|+\|\theta\|)\|D\|}{P_C}$
Proposed framework (SplitGP)	$\|\phi\|+\|\kappa\|$	$(\|\phi\|+\|\kappa\|)\|D\|$	$\beta q_c\|D\|$	$\frac{(\|\phi\|+\|\kappa\|)\|D\|}{P_C} + \frac{\beta q_c\|D\|}{R} + \frac{\beta\|\theta\|\|D\|}{P_S}$

Baseline 1: Full Model at the Client: The first baseline strategy for inference is to deploy the full model $w = [\phi, \theta]$ at individual clients, as depicted in Fig. 1(a); each client makes all predictions using its full model, without any communications with the server. In this inference strategy, the required storage for each client is $|\phi| + |\theta|$ for storing the full model. Hence, the client-side computational load becomes $(|\phi| + |\theta|)|D|$. Since all predictions are made at the client, no communication is required during inference. Hence, the inference time can be written as follows:

$$\tau_1 = \frac{(|\phi| + |\theta|)|D|}{P_C}. \tag{49}$$

Baseline 2: Full Model at the Server: When the full model is deployed at the edge server as in Fig. 1(b), all inference tasks are offloaded from each client to the server, and each client receives the predicted result back from the server. In this baseline, client-side storage is unused and the client-side computational load is also zero. Since the entire test set must be sent to the server, the required communication load during inference becomes $q|D|$, which results in uplink communication time of $\tau_2^{\text{comm}} = \frac{q}{R} = \frac{q}{b\log_2(1+\frac{p|h|^2}{bN_0})}$ per test sample. Then, the server makes inference using the full model (ϕ combined with θ), which requires server-side computation time of $\tau_2^{\text{comp}} = \frac{(|\phi|+|\theta|)}{P_S}$. The inference time can be written as the sum of communication time and server-side computation time as follows:

$$\tau_2 = (\tau_2^{\text{comm}} + \tau_2^{\text{comp}})|D| = \frac{q|D|}{R} + \frac{(|\phi| + |\theta|)|D|}{P_S}. \tag{50}$$

SplitGP: In our approach, the required storage space at each client is $|\phi| + |\kappa|$ while the server-side storage is $|\theta|$. The client first performs forward propagation for all test samples, which requires client-side computation of $(|\phi| + |\kappa|)|D|$, and computation time of

$$\tau_c^{\text{comp}} = \frac{(|\phi| + |\kappa|)|D|}{P_C}. \tag{51}$$

Now based on the Shannon entropy $E_k(z)$ computed for sample $z \in D$ as in (12), the client determines whether to offload the inference task for sample z to the edge server or not, depending on the entropy threshold E_{th}. If $E_k(z) \leq E_{th}$, the prediction for sample z is made at the client which requires no additional time. Otherwise, the client sends the output feature of z at ϕ_k to the server, which requires an additional communication load of q_c and latency of

$$\tau^{\text{comm}} = \frac{q_c}{R} = \frac{q_c}{b\log_2(1 + \frac{p|h|^2}{bN_0})}, \tag{52}$$

where q_c is the cut-layer dimension, i.e., the output dimension of ϕ. Then, the server performs forward propagation with θ for completing the inference, that incurs server-side computation time of

$$\tau_s^{\text{comp}} = \frac{|\theta|}{P_S}. \tag{53}$$

Overall, for the sample $z \in D$ with $E_k(z) \leq E_{th}$, additional latency of $\tau^{\text{comm}} + \tau_s^{\text{comp}}$ is required for making the prediction at the server-side. As a result, choosing a smaller E_{th} reduces the inference time, but can limit the overall performance since all predictions (including out-of-distribution test samples) are made at the client-side.

Suppose β portion of samples are predicted at the server-side, which is determined by the E_{th}. Then, the overall inference time can be written as

$$\begin{aligned}\tau(E_{th}) &= \tau_c^{\text{comp}} + \beta(E_{th}) \cdot (\tau^{\text{comm}} + \tau_s^{\text{comp}}) \\ &= \frac{(|\phi| + |\kappa|)|D|}{P_C} + \frac{\beta(E_{th})q_c|D|}{R} + \frac{\beta(E_{th})|\theta||D|}{P_S}.\end{aligned} \tag{54}$$

C. E_{th} Design in SplitGP

Let τ' be the average inference time per test sample that the system should support. We must have a small τ' in low-latency applications such as self-driving cars, while τ' can be relatively large for applications like smart agriculture where latency is not significant. To satisfy the latency constraint

$$\tau(E_{th})/|D| \leq \tau', \tag{55}$$

we must have

$$\beta(E_{th}) \leq \beta^*, \tag{56}$$

where

$$\beta^* = \frac{\tau' - \frac{(|\phi|+|\kappa|)}{P_C}}{\frac{q_c}{R} + \frac{|\theta|}{P_S}}. \tag{57}$$

Now define $E_{th}^{(\beta)}$ as the minimum entropy value that makes at least $1 - \beta$ portion of test samples in D to have entropy values less than or equal to E_{th} as follows:

$$E_{th}^{(\beta)} = \min\left\{E_{th} : \frac{\sum_{z\in D} \mathbb{1}_{E(z)\leq E_{th}}}{|D|} \geq 1 - \beta\right\}, \tag{58}$$

where $\mathbb{1}_A$ is an indicator function with $\mathbb{1}_A = 1$ if A is true and $\mathbb{1}_A = 0$, otherwise. Based on (57) and (58), we must have

$$E_{th} \geq E_{th}^{(\beta^*)} \tag{59}$$

to satisfy the inference time constraint. As can be seen from (57), $E_{th}^{(\beta^*)}$ is affected by various factors as follows:

- Latency threshold τ': As the latency requirement τ' decreases, β^* becomes smaller, which results in larger $E_{th}^{(\beta^*)}$. This means that we need to increase the portion of test samples predicted at the client-side to satisfy the tighter inference time requirement.
- Model sizes $|\theta|$, $|\phi|$, $|\kappa|$: Larger model sizes results in larger $E_{th}^{(\beta^*)}$. In other words, more test samples should be predicted at the client-side to meet the inference time requirement using larger models.
- Computing powers P_C, P_S: Larger computing powers of clients and server lead to smaller $E_{th}^{(\beta^*)}$; due to the increased computing powers, one can achieve the same latency constraint with more inference task offloaded to the server.
- Channel gain $|h|$: As the channel gain increases, the communication rate R between the server and the client increases. This reduces the time for task offloading from the client to the server, resulting in smaller $E_{th}^{(\beta^*)}$; more tasks can be offloaded to the server while satisfying the same inference time requirement.

D. Feasible Regimes

Based on the inference time analysis, we also pose the following question: when is our framework with split models beneficial compared to other baselines in terms of inference time? We first compare with the case where the full model is deployed at the client. From (49) and (54), we have the following proposition:

Proposition 1: Suppose E_{th} is given. We have $\tau(E_{th}) \leq \tau_1$ if and only if

$$P_C \leq \frac{|\theta| - |\kappa|}{\beta(E_{th}) \cdot (\frac{q_c}{R} + \frac{|\theta|}{P_S})}. \tag{60}$$

This is equivalent to

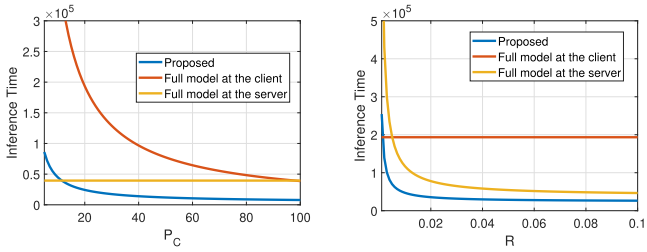
$$E_{th} \geq E_{th}^{(\beta_1)}, \text{ where } \beta_1 = \frac{|\theta| - |\kappa|}{P_C \cdot (\frac{q_c}{R} + \frac{|\theta|}{P_S})}. \tag{61}$$

The above result in (60) indicates that our solution is faster than the baseline when the client-side computing power P_C is smaller than a specific threshold. This makes intuitive sense because deploying the full model at the client-side incur significant inference latency when P_C is small (e.g., low-cost IoT devices). Equation (61) shows the equivalent condition with respect to E_{th}. Our scheme is faster compared to the baseline when we set E_{th} to be larger than $E_{th}^{(\beta_1)}$, to limit the number of data points to be offloaded to the server.

Second, we compare with the baseline where the full model is implemented at the edge server during inference. Based on (50) and (54), we state the following proposition:

Proposition 2: Suppose E_{th} is given. We have $\tau(E_{th}) \leq \tau_2$ if and only if

$$R \leq \frac{q - \beta q_c}{\frac{|\phi|+|\kappa|}{P_C} - \frac{(1-\beta)(|\phi|+|\theta|)}{P_S}}. \tag{62}$$



(a) Inference time versus P_C (b) Inference time versus R

Fig. 4. Comparison of inference time depending on client-side computing power P_C and communication rate R. Our framework demonstrates significant advantage compared to the baselines in various power-rate regimes.

This is equivalent to

$$E_{th} \geq E_{th}^{(\beta_2)}, \text{ where } \beta_2 = \frac{q - R \cdot \left(\frac{|\phi|+|\kappa|}{P_C} - \frac{(1-\beta)(|\phi|+|\theta|)}{P_S}\right)}{q_c}. \tag{63}$$

According to (62), our solution is beneficial when the communication rate R is smaller than a specific value, since this baseline requires transmission of all test samples from client to server.

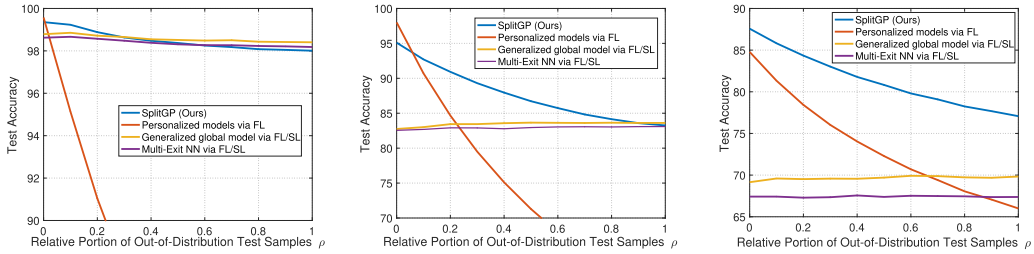
Fig. 4 compares the inference times of different schemes in Table II with $|\phi| = 387,840$, $|\theta| = 3,480,330$, $|\kappa| = 23,050$, which corresponds to the convolutional neural network (CNN) that is utilized for experiments in the next section. Other parameters are $P_S = 100$, $P_C = 20$, $R = 1$, $\beta = 0.1$, $|D| = 1$. It can be seen that our framework achieves smaller inference time compared to existing baselines in various P_C and R regimes, confirming its advantage compared to existing frameworks where the full model is deployed at the client or at the server. We also observe the effect of E_{th} on latency with more detailed analysis, in the next section.

VI. Experimental Results

A. Experimental Setup

We evaluate our method on MNIST [55], FMNIST [56] and CIFAR-10 [57] using different ML models. MNIST and FMNIST consist of 60,000 training samples and 10,000 test samples while CIFAR-10 has 50,000 training samples and 10,000 test samples. We utilize a CNN with 5 convolutional layers and 3 fully connected layers for MNIST and FMNIST datasets. For CIFAR-10, we adopt VGG-11.

Implementation: We consider $K = 50$ clients. To model non-IID data distributions, following the setup of [4], we first sort the overall train set based on classes and divide it into 100 shards; in each shard, there are 600 train samples for MNIST and FMNIST and 500 train samples for CIFAR-10. We then randomly allocate 2 shards to each client. We used a learning rate of $\eta = 0.01$ for all schemes. In each global round, each client updates its model for one epoch with a mini-batch size of 50 (i.e., 12 local updates for MNIST and FMNIST, 10 local updates for CIFAR-10), and cross-entropy loss is utilized throughout the training process. Moreover, we set $\lambda = 0.2$ and choose the optimal $E_{th} \in \{0.05, 0.1, 0.2, 0.4, 0.8, 1.2, 1.6, 2.3\}$



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

Fig. 5. Test accuracy versus ρ. For the simplest dataset MNIST, the baselines can also achieve accuracies over 98%. For more complicated datasets, FMNIST and CIFAR-10, SplitGP has advantages for most settings of ρ by capturing personalization and generalization.

unless otherwise stated. We also study the effects of λ and E_{th} in Section VI-C. We train the CNN model with MNIST and FMNIST for 120 global rounds and VGG-11 model with CIFAR-10 for 800 global rounds. For our scheme, we split the full CNN model (for MNIST and FMNIST) such that the client-side ϕ contains 4 convolutional layers ($|\phi| = 387,840$) and the server-side θ contains 1 convolutional layer and 3 fully connected layers ($|\theta| = 3,480,330$). The fully connected layer with size $|\kappa| = 23,050$ is utilized as the auxiliary classifier. We also split the VGG-11 as $|\phi| = 972,554$ and $|\theta| = 8,258,560$, and adopt the fully connected layer with size $|\kappa| = 10,250$ as a classifier. All experiments are using a NVIDIA GeForce RTX 2080Ti GPU.

Baselines: We compare SplitGP with the following baselines. First, we consider the personalized FL scheme proposed in [20], where the trained personalized models are deployed at individual clients during inference. We also consider a generalized global model constructed via conventional FL [4] as well as SplitFed [38]. Note that FL and SplitFed produce the same model while SplitFed can save storage and computation resources during training. This generalized global model can be deployed either at the client or at the server during inference. Finally, we consider a multi-exit neural network [44] that has two exits, one at the client-side and the other at the server-side, constructed via FL or SL. For this baseline, the prediction can be made either at the client or at the server, which can be viewed as the case with $\lambda = 0$ in our scheme. For a fair comparison, FedAvg [4] is adopted for the model aggregation process of all schemes.

Evaluation: When training is finished, the overall performance is measured by averaging the local test accuracies of all clients. To reflect the practical setup with data distribution shift between training and inference, we construct the local test set of each client as the union of the main test samples and the out-of-distribution test samples. The main test samples are constructed by selecting all test samples of the main classes, e.g., if client k has only classes 1 and 2 in its local data, all the test samples with classes 1 and 2 in the original test set are selected to construct the main test samples. When constructing the out-of-distribution test samples, we introduce a parameter for the relative portion of out-of-distribution test samples, which is defined as

$$\rho = \frac{\text{\# of out-of-distribution test samples}}{\text{\# of main test samples}}. \tag{64}$$

TABLE III
EFFECT OF OUT-OF-DISTRIBUTION TEST SAMPLES ON FMNIST CORRESPONDING TO FIG. 5

Methods	$\rho = 0$	$\rho = 0.2$	$\rho = 0.4$	$\rho = 0.6$	$\rho = 0.8$
Personalized FL	**98.00%**	84.67%	75.11%	67.96%	62.43%
Generalized FL	82.75%	83.44%	83.57%	83.62%	83.64%
SplitGP (Ours)	95.10%	**90.93%**	**87.95%**	**85.74%**	**84.15%**

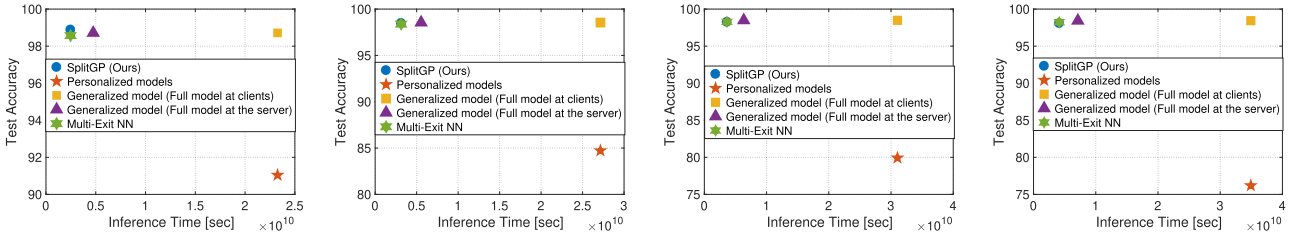
TABLE IV
EFFECT OF OUT-OF-DISTRIBUTION TEST SAMPLES ON CIFAR-10 CORRESPONDING TO FIG. 5

Methods	$\rho = 0$	$\rho = 0.2$	$\rho = 0.4$	$\rho = 0.6$	$\rho = 0.8$
Personalized FL	84.78%	78.42%	74.04%	70.7%	68.05%
Generalized FL	69.15%	69.52%	69.56%	69.92%	69.73%
SplitGP (Ours)	**87.57%**	**84.34%**	**81.78%**	**79.78%**	**78.24%**

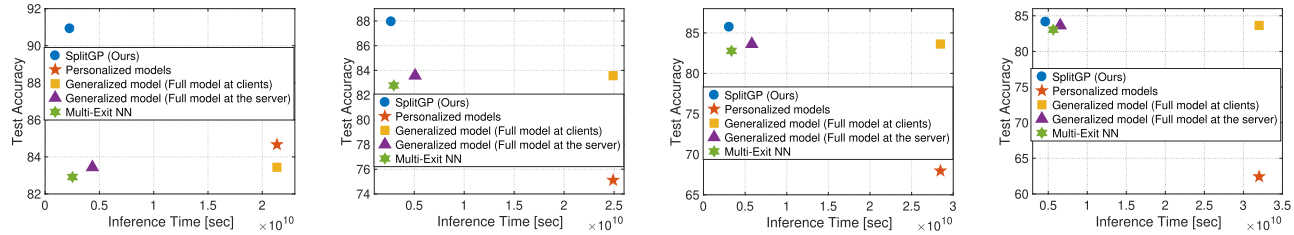
Given the main test samples, a fraction ρ of out-of-distribution samples are selected from the original test set. We reiterate that the previous works on personalized FL adopted $\rho = 0$ for evaluation.

B. Comparison With Baselines

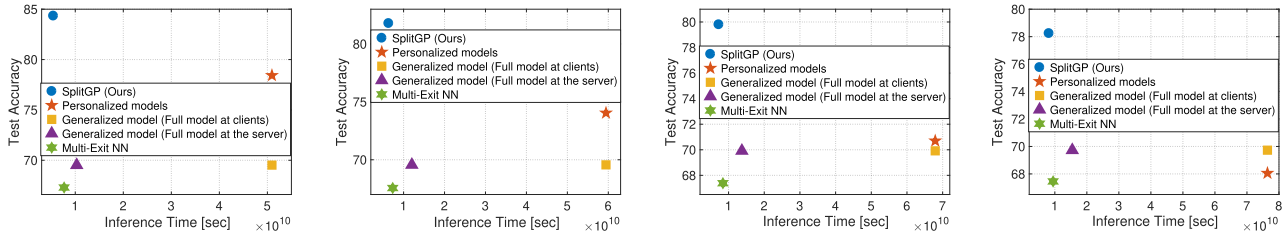
Effect of Out-of-Distribution Data on Test Accuracy: We first observe Fig. 5, which shows the performance of each scheme depending on the relative portion of out-of-distribution data ρ during inference. Tables III and IV show the result in a tabular form for FMNIST and CIFAR-10. We have the following key observations from Fig. 5 and Tables III & IV. First, the performance of the generalized global model and the multi-exit neural network constructed via FL/SL do not dramatically change with varying ρ. This implies that all classes pose a similar level of difficulty for classification, which is consistent with the class-balanced nature of the datasets we adopted. It can be also seen that the performance of personalized FL is significantly degraded as ρ grows, since the personalized models are designed to improve the performance on the main classes, not the out-of-distribution classes. Finally, it is observed that SplitGP captures both personalization and generalization capabilities: due to the personalization capability, our scheme achieves a strong performance when ρ is small, and due to the generalization capability, our scheme is more robust against ρ compared to personalized FL. Especially with more complicated datasets such as FMNIST and CIFAR-10, SplitGP is advantage against existing baselines for most settings of ρ.



(a) MNIST, $\rho = 0.2$, 2405 samples (b) MNIST, $\rho = 0.4$, 2807 samples (c) MNIST, $\rho = 0.6$, 3208 samples (d) MNIST, $\rho = 0.8$, 3611 samples

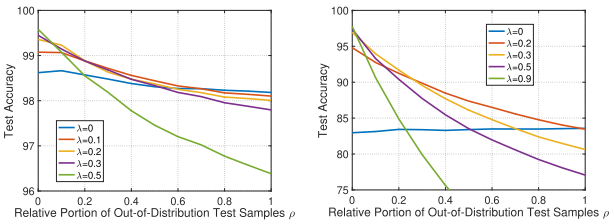


(e) FMNIST, $\rho = 0.2$, 2207 samples (f) FMNIST, $\rho = 0.4$, 2575 samples (g) FMNIST, $\rho = 0.6$, 2943 samples (h) FMNIST, $\rho = 0.8$, 3310 samples



(i) CIFAR10, $\rho = 0.2$, 2207 samples (j) CIFAR10, $\rho = 0.4$, 2575 samples (k) CIFAR10, $\rho = 0.6$, 2943 samples (l) CIFAR10, $\rho = 0.8$, 3310 samples

Fig. 6. Achievable accuracy-latency plot. The average number of test samples at each client is also reported. Our scheme achieves the best accuracy with smallest inference time for most settings of ρ, underscoring the ability of SplitGP to provide personalization and generalization while reducing inference resource requirements.



(a) MNIST (b) FMNIST

Fig. 7. Effect of λ in SplitGP. Larger λ leads to stronger personalization for handling the main classes while smaller λ leads to stronger generalization for handling the out-of-distribution classes.

TABLE V
EFFECT OF λ CORRESPONDING TO FIG. 7

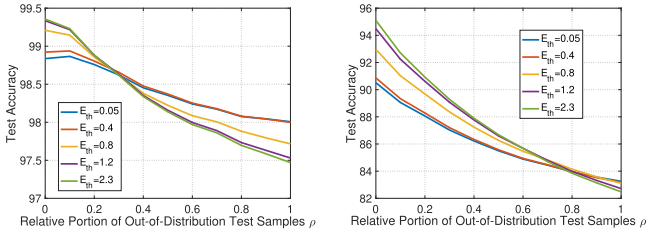
Methods	$\rho = 0$	$\rho = 0.2$	$\rho = 0.4$	$\rho = 0.6$	$\rho = 0.8$
$\lambda = 0.2$	95.10%	90.93%	**87.95%**	**85.74%**	**84.15%**
$\lambda = 0.3$	96.93%	**91.69%**	87.70%	84.79%	82.39%
$\lambda = 0.5$	97.39%	90.40%	85.46%	81.96%	79.24%
$\lambda = 0.9$	**97.75%**	84.87%	75.62%	68.79%	63.46%

The value of λ should be chosen to achieve both generalization and personalization, depending on the expected range of ρ.

TABLE VI
PERFORMANCE OF THE CLIENT-SIDE MODEL AND THE FULL MODEL

Methods	$\rho = 0.2$	$\rho = 0.4$	$\rho = 0.6$	$\rho = 0.8$
Client model (SplitGP)	90.93%	87.90%	85.68%	83.85%
Full model (SplitGP)	88.06%	86.22%	84.89%	83.96%
Overall performance (SplitGP)	**90.93%**	**87.95%**	**85.74%**	**84.15%**

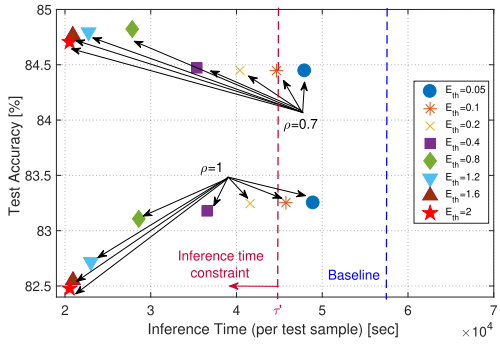
Our scheme takes the benefits of both models.



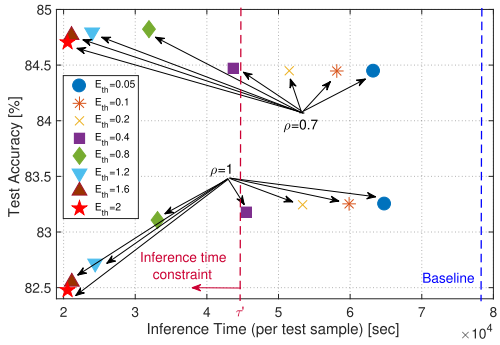
(a) MNIST (b) FMNIST

Fig. 8. Effect of E_{th} in test accuracy of SplitGP. A larger E_{th} (i.e., less task offload to the server) is a good option when ρ is small, while a smaller E_{th} (i.e., more task offload to the server) achieves a better performance when ρ is large enough.

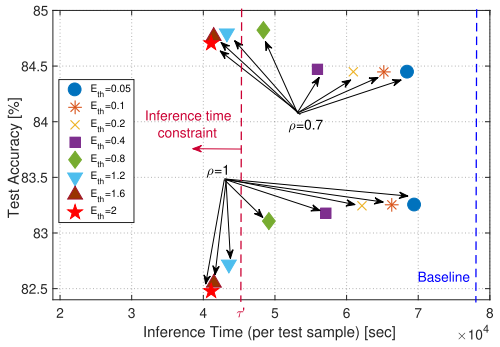
Latency, Accuracy, and Resource Improvements: Fig. 6 shows the achievable accuracy-latency performance of the different schemes. For personalized FL, the models are deployed at individual clients while the generalized global model can be deployed either at the client-side or at the server-side. To evaluate the inference time, we compute the latency from Table II by setting $P_C = 20$, $P_S = 100$, $R = 1$, as in Fig. 3. From $R_k = b_k \log_2(1 + \frac{p_k |h_k|^2}{b_k N_0})$, this corresponds to the case with $b_k = 0.1$ MHz, $N_0 = 4 \times 10^{-21}$ W/Hz, $p_k = 0.01$ W, $h_k =$



(a) $R = 1$, $P_C = 20$



(b) $R = 0.1$, $P_C = 20$



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

Fig. 9. Effect of E_{th} in both test accuracy and inference time with varying system parameters. The baseline inference time with full model at the client is marked in blue. SplitGP can flexibly choose E_{th} to satisfy the inference time constraint depending on the network resource availability.

$\sqrt{d_k^{-\varsigma}}g$, where $d_k = 300$ m is the distance between client k and the server and $\varsigma = 4$ is the pathloss factor and g is the Rayleigh fading parameter, which results in $R = 0.99 \approx 1$ Mbps. It can be seen that SplitGP achieves the best accuracy with smallest inference time for most values of ρ, confirming the advantage of our solution. Note that this performance advantage is achieved with considerable storage savings at the clients compared to deploying the full model at the client; compared to the case where $w = [\phi, \theta]$ is deployed at each client, our scheme only requires 10.62%, 10.62% and 10.64% of the storage space for MNIST, FMNIST and CIFAR-10, respectively, by storing only ϕ at the client-side. The communication load is also significantly reduced compared to the case where the full model is at the server; for example, when $\rho = 0.8$ in FMNIST, our scheme achieves the best performance while inferring only 20.30% of the test samples at the server.

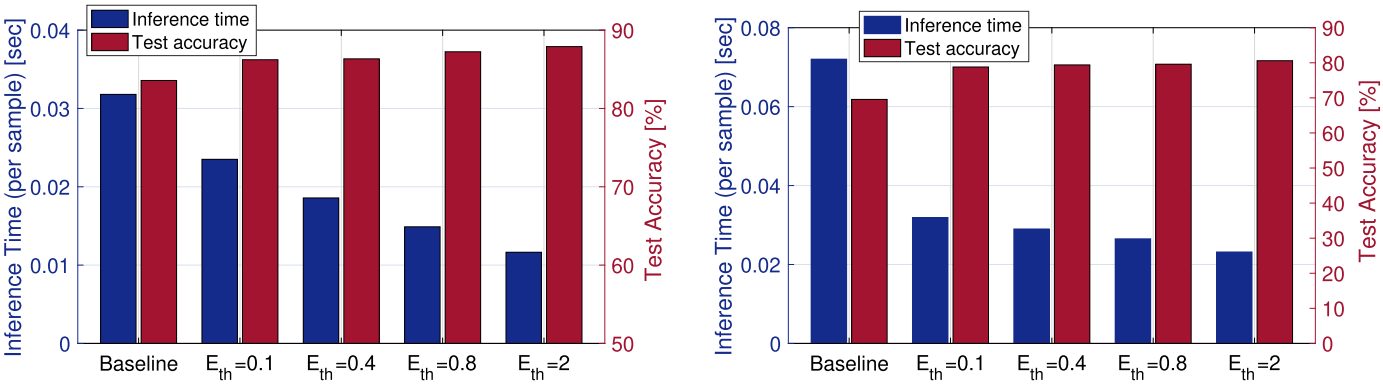
Fig. 10. Raspberry Pi testbed with the laptop working as a server. For SplitGP implementation, the client-side model components are deployed at the Raspberry Pi device, while the server-side model is deployed at the laptop.

C. Varying Parameters in SplitGP

Effect of λ: In Fig. 7, we study the effect of λ which controls the weights for personalization and generalization in SplitGP. Table V shows the details of Fig. 7(b). When λ is relatively large, the weight for the personalized client-side model increases, which leads to stronger personalization. In other words, the case with large λ performs well when ρ is small. However, the performance degrades as ρ increases, since the scheme with large λ lacks generalization capability. In general, the best λ depends on the ρ value. Without prior information, i.e., assuming ρ is uniform in the range of [0,1], $\lambda = 0.2$ gives the best expected accuracy for FMNIST. On the other hand, if we have prior knowledge that ρ is uniform in [0,0.2], $\lambda = 0.3$ is a better option.

Effect of E_{th}: In Fig. 8, we observe the effect of E_{th} which controls the amount of inference task to be offloaded to the edge server. Similar to λ, one can choose an appropriate E_{th} given the expected ρ (or the range of ρ). When ρ is small, a large E_{th} performs well, which means that a relatively large number of samples should be predicted at the client-side to achieve the highest accuracy. On the other hand, when ρ is large, smaller E_{th} performs well which indicates that a large number of samples should be predicted at the server. These observations are consistent with our intuition that the main test samples should be predicted at the client-side (with strong personalization) while the out-of-distribution samples should be predicted at the server (with strong generalization), to achieve the most robust performance.

Performance of Each Component: We also consider the performance of different components of our model. Table VI compares the performance of the client-side model (ϕ_k combined with h_k) and the full model (ϕ_k combined with θ) with the complete SplitGP on FMNIST. Due to the personalization capability, it can be seen that SplitGP relies on the client model when ρ is small. As ρ increases, SplitGP relies on both the client model and the server model to achieve generalization and personalization jointly.



(a) FMNIST (b) CIFAR-10

Fig. 11. Inference time per test sample in seconds measured with Raspberry Pi devices.

Effect of System Parameters in SplitGP Inference Design: As discussed in Section V-C, we must select E_{th} to satisfy the inference time constraint depending on the system parameters. Fig. 9 shows how E_{th} affects the test accuracy and inference time on FMNIST with different varying communication rate R and client-side computing power P_C. As the communication rate decreases (e.g., by smaller channel gain), the feasible E_{th} range becomes smaller to meet the inference time constraint τ'. As the client-side computation power decreases, SplitGP also offloads less task to the server to satisfy the latency constraint. SplitGP can be designed to satisfy the inference time constraint by controlling the feasible E_{th} range depending on the system parameters as in (59). On the other hand, the baseline with full model at the client-side (marked in blue) is not able to satisfy the inference time requirement.

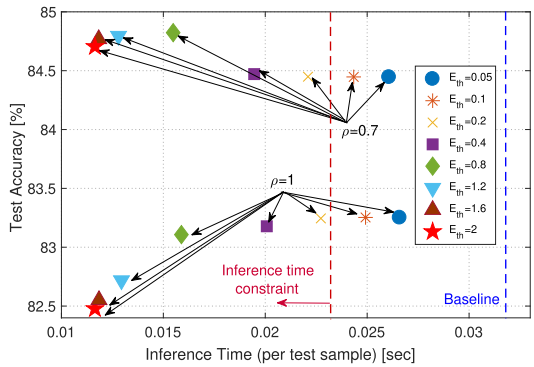


Fig. 12. Results obtained using Raspberry Pi devices with varying E_{th} and ρ. Other settings are the same as in Fig. 9.

D. Implementation on Raspberry Pi Testbed

Finally, we also implemented SplitGP using Raspberry Pi testbed to corroborate our findings in previous sections using real-world resource constrained devices. We used the Raspberry Pi 4 Model B as a client and the laptop with NVIDIA GeForce RTX 3070 Ti Laptop GPU as a server, as shown in Fig. 10. For the baseline, the full model is deployed at each Raspberry Pi device, while for SplitGP, we split the model according to the setup described in Section VI-A. During SplitGP inference, PyTorch tensors, representing intermediate features, were encoded (decoded) into (from) byte representations. TCP was then used to communicate between client and server over a typical consumer-grade router running approximately 100 Mbps.

Fig. 11 shows the inference time per test sample measured with our testbed when $\rho = 0.4$. The corresponding test accuracies are also reported. We make the following observations. First, the baseline scheme requires the largest inference time in resource-constrained devices. Second, as E_{th} increases, the inference time of SplitGP decreases, as more test samples are predicted at the client-side without communicating with the server. Here, the inference time of SplitGP is less affected by E_{th} for the CIFAR-10 dataset. This is because a larger model is adopted for CIFAR-10, which makes computation time at the Raspberry Pi device dominant compared to the communication time. According to the results on CIFAR-10, the required inference time for the baseline is larger than 7 milliseconds (ms) per test sample, while the inference time of our scheme is below 3 ms per test sample most of the cases. Considering end-to-end delay requirements for 5 G wireless reported to be on the order of 5 ms [58], an inference time reduction from 7 ms to 3 ms is expected to have a large impact on user experience. The overall results validate the effectiveness of SplitGP in real-world resource-constrained scenarios.

In Fig. 12, we consider other scenarios with $\rho = 0.7$ and $\rho = 1$, where the setups are exactly the same as in Fig. 9. It can be seen that it is more beneficial to offload more test samples to the server when ρ is large. It is also observed that the proposed scheme can achieve smaller inference time compared to the baseline where the full model is deployed at each client. The overall results in Fig. 12 obtained with Raspberry Pi devices are consistent with the ones in Fig. 9, confirming the advantage of the proposed algorithm.

VII. Conclusion

In this paper, we proposed SplitGP, a joint training and inference strategy that can capture both personalization and generalization capabilities in FL for efficient edge-AI over the network.

Based on a hybrid federated and split learning methodology, SplitGP enables the client-side model to capture personalization, while the server-side model to capture generalization. During inference, SplitGP enables each client to selectively offload the task over the network for effectively handling the out-of-distribution tasks. The convergence analysis is conducted to provide insights into SplitGP training, while the latency analysis provides guidelines on the inference task offloading depending on the inference time constraint. Extensive experimental results on real-world datasets confirmed the advantage of SplitGP in practical settings where each client needs to make predictions frequently for its main classes but also occasionally for its out-of-distribution classes.

REFERENCES

[1] D.-J. Han, D.-Y. Kim, M. Choi, C. G. Brinton, and J. Moon, "SplitGP: Achieving both generalization and personalization in federated learning," in *Proc. IEEE Conf. Comput. Commun.*, 2023, pp. 1–10.

[2] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, no. 1/2, pp. 1–210, 2021.

[3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[5] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, 2020, pp. 429–450.

[6] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," in *Proc. Int. Conf. Learn. Representations*, 2020.

[7] D. A. E. Acar, Y. Zhao, R. Matas, M. Mattina, P. Whatmough, and V. Saligrama, "Federated learning based on dynamic regularization," in *Proc. Int. Conf. Learn. Representations*, 2020.

[8] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5132–5143.

[9] S. Wang et al., "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[10] M. M. Amiri and D. Gündüz, "Federated learning over wireless fading channels," *IEEE Trans. Wireless Commun.*, vol. 19, no. 5, pp. 3546–3557, May 2020.

[11] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269–283, Jan. 2021.

[12] Y. Park, D.-J. Han, D.-Y. Kim, J. Seo, and J. Moon, "Few-round learning for federated learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 28612–28622.

[13] J. Park, D.-J. Han, M. Choi, and J. Moon, "Sageflow: Robust federated learning against both stragglers and adversaries," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 840–851.

[14] D.-J. Han, M. Choi, J. Park, and J. Moon, "FedMes: Speeding up federated learning with multiple edge servers," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3870–3885, Dec. 2021.

[15] H. H. Yang, Z. Liu, T. Q. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Trans. Commun.*, vol. 68, no. 1, pp. 317–333, Jan. 2020.

[16] Y. Tu, Y. Ruan, S. Wagle, C. G. Brinton, and C. Joe-Wong, "Network-aware optimization of distributed learning for fog computing," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2509–2518.

[17] S. Wang, M. Lee, S. Hosseinalipour, R. Morabito, M. Chiang, and C. G. Brinton, "Device sampling for heterogeneous federated learning: Theory, algorithms, and implementation," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[18] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-IID data with reinforcement learning," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1698–1707.

[19] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4427–4437.

[20] Y. Deng, M. M. Kamani, and M. Mahdavi, "Adaptive personalized federated learning," 2020, *arXiv: 2003.13461*.

[21] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 3557–3568.

[22] M. Zhang, K. Sapra, S. Fidler, S. Yeung, and J. M. Alvarez, "Personalized federated learning with first order model optimization," in *Proc. Int. Conf. Learn. Representations*, 2021.

[23] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and robust federated learning through personalization," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 6357–6368.

[24] C. T. Dinh et al., "Federated learning over wireless networks: Convergence analysis and resource allocation," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 398–409, Feb. 2021.

[25] K. Wei et al., "User-level privacy-preserving federated learning: Analysis and performance optimization," *IEEE Trans. Mobile Comput.*, vol. 21, no. 9, pp. 3388–3401, Sep. 2022.

[26] S. Itahara, T. Nishio, Y. Koda, M. Morikura, and K. Yamamoto, "Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-IID private data," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 191–205, Jan. 2023.

[27] J. Liu et al., "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 674–690, Feb. 2023.

[28] M. N. Nguyen, N. H. Tran, Y. K. Tun, Z. Han, and C. S. Hong, "Toward multiple federated learning services resource sharing in mobile edge networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 541–555, Jan. 2023.

[29] X. Lin, J. Wu, J. Li, X. Zheng, and G. Li, "Friend-as-learner: Socially-driven trustworthy and efficient wireless federated edge learning," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 269–283, Jan. 2023.

[30] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," 2019, *arXiv: 1912.00818*.

[31] K. Pillutla, K. Malik, A.-R. Mohamed, M. Rabbat, M. Sanjabi, and L. Xiao, "Federated learning with partial model personalization," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 17716–17758.

[32] H.-Y. Chen and W.-L. Chao, "On bridging generic and personalized federated learning for image classification," in *Proc. Int. Conf. Learn. Representations*, 2021.

[33] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1935–1949, Mar. 2021.

[34] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Convergence time optimization for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 4, pp. 2457–2471, Apr. 2021.

[35] Z. Zhao et al., "Federated learning with Non-IID data in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 1927–1942, Mar. 2022.

[36] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," 2018, *arXiv: 1812.00564*.

[37] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, 2018.

[38] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "SplitFed: When federated learning meets split learning," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 8485–8493.

[39] C. He, M. Annavaram, and S. Avestimehr, "Group knowledge transfer: Federated learning of large CNNs at the edge," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, Art. no. 1180.

[40] D.-J. Han, H. I. Bhatti, J. Lee, and J. Moon, "Accelerating federated learning with split learning on locally generated losses," in *Proc. ICML Workshop Federated Learn. User Privacy Data Confidentiality*, 2021.

[41] S. Oh et al., "LocFedMix-SL: Localize, federate, and mix for improved scalability, convergence, and latency in split learning," in *Proc. ACM Web Conf.*, 2022, pp. 3347–3357.

[42] W. J. Yun et al., "SlimFL: Federated learning with superposition coding over slimmable neural networks," *IEEE/ACM Trans. Netw.*, early access, Jan. 02, 2023, doi: 10.1109/TNET.2022.3231864.

[43] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.
[44] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.
[45] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell, "Learning anytime predictions in neural networks via adaptive loss balancing," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 3812–3821.
[46] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *Proc. Int. Conf. Learn. Representations*, 2018.
[47] H. Li, H. Zhang, X. Qi, R. Yang, and G. Huang, "Improved techniques for training adaptive deep networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1891–1900.
[48] M. Phuong and C. H. Lampert, "Distillation-based training for multi-exit architectures," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1355–1364.
[49] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 656–672.
[50] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on non-IID data," in *Proc. Int. Conf. Learn. Representations*, 2020.
[51] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 2021–2031.
[52] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, Art. no. 1316.
[53] S. Zehtabi, S. Hosseinalipour, and C. G. Brinton, "Event-triggered decentralized federated learning over resource-constrained edge devices," 2022, *arXiv:2211.12640*.
[54] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," 2016, *arXiv:1605.07678*.
[55] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[56] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[57] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
[58] W. Saad, M. Bennis, and M. Chen, "A vision of 6G wireless systems: Applications, trends, technologies, and open research problems," *IEEE Netw.*, vol. 34, no. 3, pp. 134–142, May/Jun. 2020.

Dong-Jun Han (Member, IEEE) received the BS, MS, and PhD degrees from KAIST, in 2016, 2018, and 2022, respectively. He is currently a postdoctoral researcher with Purdue University.

Do-Yeon Kim received the BS degree from Yonsei University, in 2019, and the MS degree from KAIST, in 2021. He is currently working toward the PhD degree with KAIST.

Minseok Choi (Member, IEEE) received the BS, MS, and PhD degrees from KAIST, in 2011, 2013, and 2018, respectively. He is currently an assistant professor with Kyung Hee University.

David Nickel (Graduate Student Member, IEEE) received the BS degree from Purdue University, in 2023. He is currently working toward the PhD degree with Purdue University.

Jaekyun Moon (Fellow, IEEE) received the PhD degree from Carnegie Mellon University, in 1990. He is currently the dean with the College of Engineering and a professor of EE with KAIST.

Mung Chiang (Fellow, IEEE) received the PhD degree from Stanford University, in 2003. He is currently the president with Purdue University and a Roscoe H. George distinguished professor of ECE.

Christopher G. Brinton (Senior Member, IEEE) received the PhD degree in EE from Princeton University, in 2016. He is currently the Elmore Chaired assistant professor of ECE with Purdue University.