SPIDER: Searching Personalized Neural Architecture for Federated Learning

Erum Mushtaq [1], **Chaoyang He** [1], **Jie Ding** [2], **Salman Avestimehr** [1]

[1]University of Southern California, [2]University of Minnesota
emushtaq@usc.edu, chaoyang.he@usc.edu, dingj@umn.edu, avestime@usc.edu

Abstract

Federated learning (FL) is an efficient learning framework that assists distributed machine learning when data cannot be shared with a centralized server due to privacy and regulatory restrictions. Recent advancements in FL use predefined architecture-based learning for all the clients. However, given that clients' data are invisible to the server and data distributions are non-identical across clients, a predefined architecture discovered in a centralized setting may not be an optimal solution for all the clients in FL. Motivated by this challenge, in this work, we introduce SPIDER, an algorithmic framework that aims to **S**earch **P**ersonal**I**zed neural architecture for fe**DER**ated learning. SPIDER is designed based on two unique features: (1) alternately optimizing one architecture-homogeneous global model (Supernet) in a generic FL manner and one architecture-heterogeneous local model that is connected to the global model by weight sharing-based regularization (2) achieving architecture-heterogeneous local model by a novel neural architecture search (NAS) method that can select optimal subnet progressively using operation-level perturbation on the accuracy value as the criterion. Experimental results on CIFAR10 and CIFAR100 datasets demonstrate that SPIDER outperforms other state-of-the-art personalization methods, and the searched personalized architectures are more inference efficient.

1 Introduction

Federated Learning (FL) is a promising decentralized machine learning framework that facilitates data privacy and low communication costs. It has been extensively explored in various machine learning domains such as computer vision, natural language processing, and data mining. Despite many benefits of FL, one major challenge involved in FL is data heterogeneity, meaning that the data distributions across clients are not identically or independently (non-I.I.D) distributed. The non-I.I.D distributions result in the varying performance of a globally learned model across different clients. In addition to data heterogeneity, data invisibility is another challenge in FL. Since clients' private data remain invisible to the server, from the server's perspective, it is unclear how to select a pre-defined architecture from a pool of all available candidates. In practice, it may require extensive experiments and hyper-parameter tuning over different architectures, a procedure that can be prohibitively expensive.

To address the data-heterogeneity challenge, variants of the standard FedAvg have been proposed to train a global model, including the `FedProx` (Li et al. 2018), `FedOPT` (Reddi et al. 2020), and `FedNova` (Wang et al. 2020). In addition to training of a global model, frameworks that focus on training personalized models have also gained a lot of popularity. The `Ditto` (Li et al. 2021b), `PerFedAvg` (Fallah, Mokhtari, and Ozdaglar 2020a), and `pFedMe` (Dinh, Tran, and Nguyen 2020) are some of the recent works that have shown promising results to obtain improved performance across clients. However, all these works exploit predefined architectures and operate at the optimization layer. Consequently, in addition to their inherent hyper-parameters tuning, these personalization frameworks often encounter the data-invisibility challenge that one has to select a suitable model architecture involving a lot of hyper-parameter tuning.

In this work, we adopt a different and complementary technique to address data heterogeneity challenge for FL. We introduce SPIDER, an algorithmic framework that aims to **S**earch **P**ersonal**I**zed neural architecture for fe**DER**ated learning. Recall that in a centralized setting, the neural architecture search (NAS) aims to search for optimal architecture to address system design challenges such as lower latency (Wu et al. 2019), lesser memory cost (Li et al. 2021a), and smaller energy consumption (Yang et al. 2020). For architecture search, there are three well known methods explored in literature, gradient-based (Liu, Simonyan, and Yang 2018), evolutionary search (Liu et al. 2021), and reinforcement learning (Jaafra et al. 2019). Out of these, gradient-based methods are generally considered more efficient because of their ability to yield higher performance in comparatively lesser time (Santra, Hsieh, and Lin 2021).

To achieve personalization at the architecture level in FL, we propose a unified framework, SPIDER. This framework essentially deploys two models, local and global models, on each client. Initially, both models use the DARTS search space based Supernet (Liu, Simonyan, and Yang 2018), an over-parameterized architecture. In the proposed framework, the global model is shared with the server for the FL updates and therefore, stays the same in the architecture design. On the other hand, the local model stays

completely private, and performs personalized architecture search, and therefore, gets updated. To search for the personalized child model, SPIDER deploys SPIDER-Searcher on each client's local model. The SPIDER-Searcher is built upon a well-known gradient based NAS method, named perturbation based NAS (Wang et al. 2021). The main objective of the SPIDER framework is to allow each client to search and optimize their local models using SPIDER-Searcher while benefiting from the global model. To achieve this goal, we propose an alternating bi-level optimization based SPIDER Trainer, that trains local and global models in an alternate fashion. However, the challenge here is the optimization of an evolving local model architecture while exploiting a fixed global architecture. To address this challenge, SPIDER-Trainer performs weight sharing based regularization, that regularizes the common connections between global model's Supernet and local model's child model. This aids clients to search and train heterogeneous architectures tailored for their local data distributions. In nutshell, this approach not only yields architecture personalization in FL but also facilitates model privacy (in the sense that the derived child local model is not shared with the server at all).

To evaluate the performance of the proposed algorithm, we consider a cross-silo FL setting and use Dirichlet distribution to create non-I.I.D data distribution across clients. For evaluation, we report test accuracy at each client on the 20% of training data kept as test data for each client. We show that the architecture personalization yields better results than state-of-the-art personalization algorithms based solely on the optimization layer, such as Ditto (Li et al. 2021b), perFedAvg (Fallah, Mokhtari, and Ozdaglar 2020a), and local adaptation (Cheng, Chadha, and Duchi 2021).

To summarize, the following are the key contributions of our work.

- We propose and formulate a personalized neural architecture search framework for FL named SPIDER, from a perspective complementary to the state-of-the-arts to address data heterogeneity challenges in FL.
- SPIDER is designed based on two unique features: (1) maintaining two models at each client, one to communicate with the server and the other to perform a local progressive search, and (2) operating local search and training at each client by an alternating bilevel optimization and weight sharing-based regularization along the FL updates.
- We run extensive experiments to demonstrate the benefit of SPIDER compared with state-of-the-art personalized FL approaches such as Ditto (Li et al. 2021b), perFedAvg (Fallah, Mokhtari, and Ozdaglar 2020a) and Local Adaptation (Cheng, Chadha, and Duchi 2021). In particular on CIFAR10 dataset with heterogeneous distribution we demonstrate an increase of the average local accuracy by 2.8%, 1.7%, and 5.5%, over Ditto, PerFedAvg, and Local Adaption, respectively.
- We also demonstrate that SPIDER learns smaller personalized architectures of average size around 14MB, which is three times smaller than a pre-defined architecture Resnet18 (of size 44MB).

2 Related Works

Heterogeneous Neural Architecture for FL Heterogeneous neural architecture is one way to personalize the model in FL. For personalization, the primal-dual framework (Smith et al. 2017a), clustering (Sattler, Müller, and Samek 2020), fine-tuning with transfer learning (Yu, Bagdasaryan, and Shmatikov 2020a), meta-learning (Fallah, Mokhtari, and Ozdaglar 2020a), regularization-based method (Hanzely and Richtárik 2020; Li et al. 2021b) are among the popular methods explored in the FL literature. Although these techniques achieve improved personalized performance, all of them use a pre-defined architecture for each client. HeteroFL (Diao, Ding, and Tarokh 2020) is a recent work that accomplishes the aggregation of heterogeneous models by assigning sub-parts of the global model based on their computation budget and aggregating the parameters common between different clients. Another work (Lin et al. 2020a) accomplishes this task by forming clusters of clients of the same model and allowing for heterogeneous models across clusters. On the server-side, the aggregation is based on cluster-wise aggregation followed by a knowledge distillation from the aggregated models into the global model. Given data invisibility in FL, deciding which predefined architecture would work for which client is a challenging task and requires exploration. As such, our proposed method aims to achieve personalized architecture automatically.

Neural Architecture Search for FL Neural Architecture Search (NAS) has gained momentum in recent literature to search for a global model in a federated setting. FedNAS (He, Annavaram, and Avestimehr 2020a) explores the compatibility of MileNAS solver with Fed averaging algorithm to search for a global model. Direct Federated NAS (Hu et al. 2020) is another work in this direction that explores the compatibility of a one-shot NAS method, DSNAS (Hu et al. 2020), with Fed averaging algorithm with the same application, in search of a global model. (Zhu and Jin 2021) uses evolutionary NAS to design a master (global) model. (Singh et al. 2020) explores the concept of differential privacy using DARTs solver (Liu, Simonyan, and Yang 2018) to explore the trade-off between accuracy and privacy of a global model. (Xu et al. 2020) starts with a pre-trained handcraft model and continues pruning the model until it satisfies the efficiency budget. Where all these models search for a unified global model, a key distinction of our work with these works is that we aim to search for a personalized model for each client.

3 Preliminaries, Motivation, and Design Goals

In this section, we introduce the state-of-the-art methods for personalized federated learning, discuss the motivation for personalizing model architectures, and summarize our design goals.

Personalized Federated Learning A natural formulation of FL is to assume that among K distinct clients, each client

k has its own distribution P_i, draws data observations (samples) from P_i, and aims to solve a supervised learning task (e.g., image classification) by optimizing a global model w with other clients collaboratively. At a high-level abstraction, the optimization objective is then defined as:

$$\min_{w^*} G\left(F_1(w), ..., F_K(w)\right), \tag{1}$$

where $F_k(w)$ measures the performance of the model global w on the private dataset at client k (local objective), and G is the global model aggregation function that aggregates each client's local objectives. For example, for FedAvg, $G(.)$ would be weighted aggregation of the local objectives, $\sum_{k=1}^{K} p_k F_k(w)$, where $\sum_{k=1}^{K} p_k = 1$.

However, as distributions across individual clients are typically heterogeneous (i.e., non-I.I.D.), there is a growing line of research that advocates to reformulate FL as a personalization framework, dubbed as personalized FL (PFL). In PFL, the objective is redirected to find a personalized model v_k for device k that performs well on the local data distribution:

$$\min_{v_1^*,...,v_K^*} \left(F_1(v_1), ..., F_K(v_K)\right), \tag{2}$$

To solve this challenging problem, various PFL methods are proposed, including FedAvg with local adaptation (Local-FL) (Cheng, Chadha, and Duchi 2021; Yu, Bagdasaryan, and Shmatikov 2020b; Wang et al. 2019), MAML-based PFL (MAML-FL) (Fallah, Mokhtari, and Ozdaglar 2020b; Jiang et al. 2019), clustered FL (CFL) (Ghosh et al. 2020; Sattler, Müller, and Samek 2021), personalized layer-based FL (PL-FL) (Liang et al. 2020), federated multitask learning (FMTL) (Smith et al. 2017b), and knowledge distillation (KD) (Lin et al. 2020b; He, Annavaram, and Avestimehr 2020b).

Motivation for Neural Architecture Personalization Distinct from these existing works on PFL, we propose a new approach to instead personalize model architecture at each client. We are motivated by three key potential benefits. First, the searched architecture at each client is expected to fit its own distinct distribution, which has the potential to provide substantial improvement over the existing PFL baselines that only personalize model weights. Second, a personalized architecture search can result in a more compressed model at each client that will reduce inference latency and efficiency. Third, a personalized architecture search allows the clients to even keep their local model architectures private in a sense the server and other clients neither know the architecture nor the weights of that architecture. This further enhances the privacy guarantees of FL and is helpful in business cases that each client hopes to also protect its model architecture.

Design Goals Our goal is to enable personalized neural architecture search for all clients in FL. In this context, the limitation of existing personalized FL methods is obvious: Local-FL and MAML-FL need every client to have the same architecture to perform local adaptations; In CFL, the clustering step requires all clients to share a homogeneous model architecture; PL-FL can only obtain heterogeneous architectures for a small portion of personalized layers, but it does not provide an architecture-agnostic method to determine the boundary of personalized layers in an automated mechanism; FMTL is a regularization-based method which cannot perform regularization when architectures are heterogeneous across clients; KD has an unrealistic assumption that the server has enough public dataset as the auxiliary data for knowledge distillation. In addition, an ideal FL framework for deployment is the one that can jointly optimize the inference latency/efficiency during training. However, after federated training, none of these PFL clearly specify the method for efficient inference. An additional model compression procedure (e.g., pruning or KD-based) may be required, but it is impractical to perform a remote model compression client by client.

To avoid these limitations, our goal is to design an architecture-personalized FL framework with the following requirements:

- **R1**: *allowing heterogeneous architectures for all clients*, which can capture fine-grained data heterogeneity;
- **R2**: *searching and personalizing the entire architecture space*, to avoid the heuristic search for the boundary of personalized layers;
- **R3**: *requiring no auxiliary data at the client- or server-side (unlike knowledge distillation-based PFL)*;
- **R4**: *resulting in models with practical inference latency and efficiency*, to avoid the need for an additional model compression procedure at each client.

We now introduce SPIDER that meets the above requirements in a unified framework.

4 Methodology: SPIDER

4.1 Overview

The overall framework of SPIDER is illustrated in Figure 1. Essentially, each client maintains two models in this framework: one architecture-homogeneous global model for collaborative training with other clients, and one architecture-heterogeneous local model that initially shares the same super architecture space as the global model. At a high-level, SPIDER is formulated as an **architecture-personalized bi-level optimization** problem (Section 4.2) and proposes the solver as the orchestration of **SPIDER Trainer** (Section 4.3) and **SPIDER-Searcher** (Section 4.4). **SPIDER Trainer** is an architecture-personalized training framework that can collaboratively train heterogeneous neural architectures across clients.

To allow federated training on the expected heterogeneous local architectures, it enables regularization between an arbitrary personalized architecture and the global model via weight sharing. With this support, **SPIDER-Searcher** is designed to *dynamically adjust the architecture of each client's local model on the way*. To search a personalized architecture for the local data distribution of each client, SPIDER-Searcher is built on a novel neural architecture search (NAS) method that searches optimal local Subnet

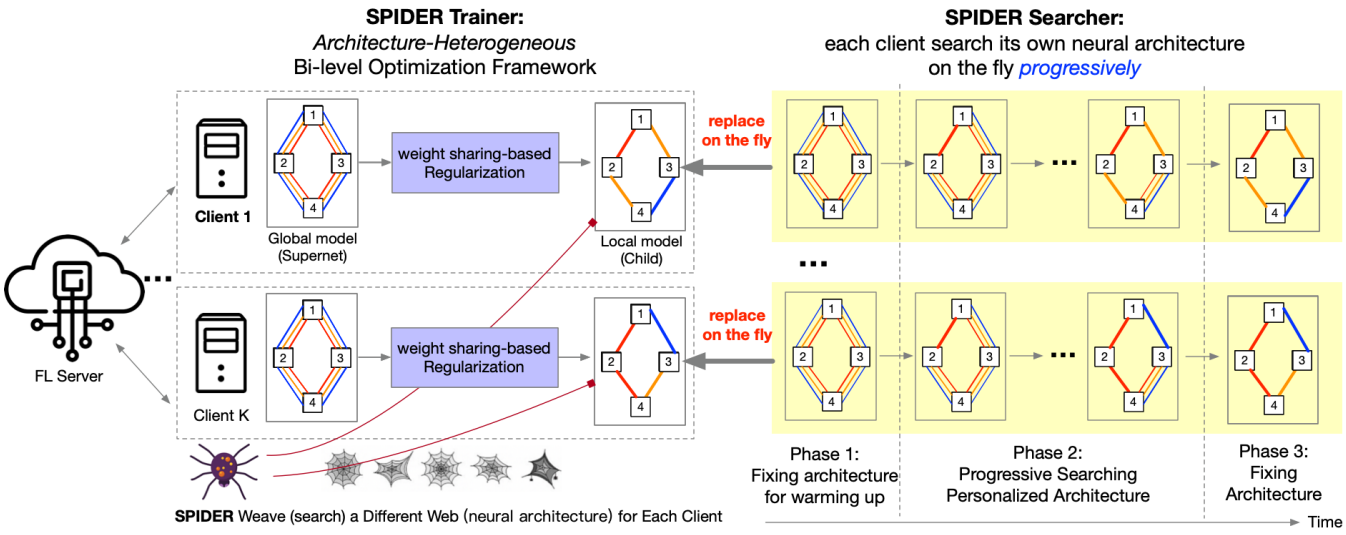


Figure 1: Illustration of SPIDER framework. SPIDER weaves (searches) a different web (neural architecture) for each client.

progressively using operation-level perturbation on the accuracy value as the criterion. Overall, each client's local model goes through three phases (also shown in Figure 1): pre-training to warm up the initial local model, progressive neural architecture search, and final training of the searched architecture-personalized model.

SPIDER can meet the design goals **R1**-**R4** introduced in Section 3 because 1) each client performs independent architecture personalization with its own private data (**R1**), 2) the search space is not restricted to a portion of the model (**R2**), 3) no auxiliary data is used to assist the search and train process (**R3**), and 4) progressive operation selection gradually reduces the number of candidate architectures, leading to a sparse and efficient model for inference (**R4**).

4.2 SPIDER Formulation: Architecture-personalized Bi-level Optimization

SPIDER aims to personalize (weave) a different neural architecture (web) for each client. To generate heterogeneous architectures across clients, we use two models, a local model (a_k) and a global model (Supernet $\mathcal{A}$), at each client and formulate SPIDER as an architecture-personalized bi-level optimization problem for each client $k \in [K]$:

$$\min_{v_k, a_k \subseteq \mathcal{A}} \quad F_k(v_k, a_k; w^*, \mathcal{A}) \tag{3}$$

$$\text{s.t.} \quad w^* \in \arg\min_{w} G\left(F_1(w, \mathcal{A}), ..., F_K(w, \mathcal{A})\right), \tag{4}$$

where F_k is the local objective of the client k; w, v_k, and a_k are all learnable parameters; w denotes the parameter of the global model architecture $\mathcal{A}$, v_k is the weight parameter of the local personalized architecture a_k of the client k. Here, a_k is a child neural architecture of a Supernet $\mathcal{A}$, denoted by $a_k \subseteq \mathcal{A}$. Note that in Eq.(4), we aim to learn a global model $\mathcal{A}$ in a federated learning setting, which formulates our outer optimization. However, in the inner optimization given in Eq.(3), the objective of each client is to optimize its local model's architecture a_k and its associated parameters v_k while benefiting from the global model w^*.

Definition of Supernet $\mathcal{A}$ and Child Neural Architecture a_k As a tractable, yet general case study, SPIDER reuses the DARTS architecture space as Supernet $\mathcal{A}$: there are 8 cells, and each cell consists of multiple edges; each edge connects two intermediate representations (node) by a mixture of multiple operations frequently used in various modern CNNs (e.g., sep convolution 3x3, sep convolution 5x5, dil convolution 3x3, skip connection, max pool 3x3, avg pool 3x3); the mixture uses softmax over all possible operations to relax the categorical discrete candidate to a continuous search space. More precisely, $\mathcal{A}$ contains a set of edges $\{e_1, ..., e_E\}$, and each edge e has multiple operations $\{o_1, ..., o_O\}$. Based on this definition, a_k maintains the operation-level granularity: a_k's edge set space is a subset of $\mathcal{A}$'s edge set space, and the operation set in a_k's each edge may also be a subset space.

The difficulty of jointly optimizing the architecture a_k and related weight parameters v_k The key difference of our formulation from existing bi-level optimization for FL (e.g., (Li et al. 2021b)) is that in our case, a_k is also a learnable parameter (Eq.(3)). We assume each client can have an evolving architecture a_k, i.e., Eq.(3) has to optimize the architecture a_k and its related weight parameters v_k jointly, while using complete Supernet-based global model weights, w. SPIDER addresses this challenge by the orchestration of SPIDER-Trainer and SPIDER-Searcher.

4.3 SPIDER Trainer: Federated Training on Heterogeneous Architectures

In this section, we describe SPIDER trainer, an architecture-personalized training framework that can collaboratively train heterogeneous neural architectures across clients.

To clearly show how SPIDER handles the optimization difficulty of Eq.(3), we first downgrade the objective to the

case that all clients use *predefined* (fixed) heterogeneous architectures (derived from the Supernet $\mathcal{A}$). More specially, we reduce the aforementioned optimization framework in Eq.(3) and Eq.(4) to the following:

$$\min_{v_k} \quad h_k(v_k, a_k; w^*, \mathcal{A}) = F_k(v_k) + \frac{\lambda}{2}||v_k - w^*_{share}||^2 \tag{5}$$

$$\text{s.t.} \quad w^* \in \arg\min_{w} G\left(F_1(w, \mathcal{A}), ..., F_K(w, \mathcal{A})\right), \tag{6}$$

where local model's weights v_k are regularized towards the global model w^*_{share}, where $w^*_{share} = w^* \odot a_k$ (i.e., only using the weight parameters of the operation set space overlapping (sharing) with a_k). Also, λ is the regularization hyper-parameter. Note that, now, only v_k needs to be optimized in Eq.5, while a_k is fixed during the optimization.

We then solve Eq.5 and Eq.6 alternately. We summarize this optimization procedure as SPIDER-Trainer with a detailed pseudo code illustrated in Algorithm 1. In this algorithm, we can note that the global model (line #12) and the local model (line #14) are updated alternately. The strength of this algorithm lies in its elaborate design, which provides the following key benefits:

Algorithm 1: **SPIDER Trainer**

1: **Initialization:** initialize K clients with the k-th client has a global model w_k using Supernet $\mathcal{A}$, and a local model v_k using subnet a_k (set $a_k = \mathcal{A}$ at the begining); E is the number of local epochs; T is the number of rounds; T_s number of rounds to start search; τ is the recovery periods in the units of rounds.
2: **Server executes:**
3: **for** each round $t = 0, 1, 2, ..., T-1$ **do**
4: **for** each client k **in parallel do**
5: $w_k^{t+1} \leftarrow$ ClientLocalSearch(k, w^t, t)
6: **end for**
7: $w^{t+1} \leftarrow \sum_{k=1}^{K} \frac{N_k}{N} w_k^{t+1}$
8: **end for**
9:
10: *function* **ClientLocalSearch**(k, w^t, t): // *Run on client* k
11: **for** e in epoch **do**
12: **for** minibatch in training and validation data **do**
13: $a_k^t = \textbf{ProgressiveNAS}(a_k^t, T_s, \tau, t)$
14: Update Global model: $w^{t+1} = w^t - \eta_w \nabla^{\text{tr}}_{wk}(w^t)$
15: $w^{t+1}_{share} = w^{t+1} \odot a_k^t$ //weight sharing
16: Update Local Model: $v_k^{t+1} = v_k^t - \eta_v \left(\nabla^{\text{tr}}_{vk}(v_k^t) + \gamma(v_k^t - w^{t+1}_{share})\right)$
17: **end for**
18: **end for**
19: **return** w to server

(1) Enabling regularization between an arbitrary personalized architecture and the global model Most importantly, SPIDER-Trainer connects each personalized model with the global model by enabling the regularization between two different architectures: an arbitrary personalized architecture for the local model a_k of client k and the global model with Supernet $\mathcal{A}$. This is done by weight sharing. More specially, in Eq.6, $w^*_{share} = w^* \odot a_k$, which provides us the global model's weight parameters for the connections/edges common between the child neural architecture a_k and the global model's Supernet $\mathcal{A}$. w^*_{share} is essentially used to regularize a subnet (a_k) model parameters v_k towards the global model shared/common parameters w^*_{share}, as shown in Eq. 6.

(2) Avoiding heterogeneous aggregation SPIDER-Trainer avoids the aggregation of heterogeneous model architectures at the server side. As such, no sophisticated and unstable aggregation methods are required (e.g., masking, knowledge distillation (Lin et al. 2020b), etc.), and it is flexible to use other aggregation methods beyond FedAvg (e.g., (Karimireddy et al. 2020; Reddi et al. 2021)) to update the global model.

(3) Enabling architecture privacy In this algorithm, only the global model is transmitted between the client and the server. This enables architecture privacy because each client's architecture is hidden from server and other clients.

(4) Potential robustness to adversarial attacks The weight sharing-based regularization not only yields the benefit of personalization in FL, but also makes the FL framework more robust to adversarial attacks. Its robustness advantage comes from its ability to keep the local model private and regularizing towards the global model based on its regularization parameter, as show before by architecture-homogeneous bi-level optimization (Li et al. 2021b).

4.4 SPIDER-Searcher: Personalizing Architecture

Although SPIDER trainer is able to collaboratively train heterogeneous architectures, manual design of the architecture for each client is impractical or suboptimal. As such, we further add a neural architecture search (NAS) component, SPIDER-Searcher, in Algorithm 1 (line #11) to adapt a_k to its local data distribution in a progressive manner. We now present the details of SPIDER-Searcher.

Progressive Neural Architecture Search Essentially, SPIDER-Searcher dynamically changes the architecture of a_k during the entire federated training process. This is feasible because the weight sharing-based regularization can handle an arbitrary personalized architecture (introduced in Section 4.3). Due to this characteristic, SPIDER-Searcher can search a_k in a progressive manner (shown in Figure 1): **Phase 1**: At the beginning, a_k is set equal to Supernet $\mathcal{A}$. The intention of SPIDER-Searcher in this phase is to warm up the training of the initial a_k so it does not change a_k for a few rounds; **Phase 2**: After warming up, SPIDER-Searcher performs edge-by-edge search gradually. In each edge search, only the operation with the highest impact to the accuracy is kept. It also uses a few rounds of training as a recovery time before proceeding the next round of search. This process continues until all edges finish searching; **Phase 3**: After all edges finish searching, SPIDER-Searcher does not change a_k. This serves as a final training of the searched architecture-personalized model. This three-phase procedure is summarized as Algorithm 2. Now, we proceed to elaborate how we calculate the impact of an operation on the Supernet.

Algorithm 2: **SPIDER-Searcher**

Search Space: in the architecture $a_k^t \subseteq \mathcal{A}$, $\mathcal{E}$ is the super set of all edges $\{e_1, ..., e_E\}$, $\mathcal{E}_s$ is the remaining subset of edges that have not been searched, and each edge e has multiple operations $\{o_1, ..., o_O\}$.
function **ProgressiveNAS**(a_k, T_s, τ, t)
  if $t \geq T_s$ and t % τ == 0 and LEN($\mathcal{E}_s$) > 0 **then**
    e_i = RANDOM ($\mathcal{E}$) // random selection
    // searching without training
    for all operation o_j on edge e_i **do**
      evaluate validation accuracy of a_k^t when o_j is removed($ACC_{\setminus o}$)
    end for
    in e_i, keep only one operation corresponding to the lowest value of ($ACC_{\setminus o}$), i.e., highest impact.
    remove e_i from $\mathcal{E}$
  else
    return a_k^t directly
  end if
  return updated a_k^t after selection

Operation-level perturbation-based selection In phase 2, we specify selecting the operation with the highest impact using operation-level perturbation. More specially, instead of optimizing the mixed operation architecture parameters α using another bi-level optimization as DARTS (a.k.a. gradient-based search) to pick optimal operation according to magnitude of α parameters (*magnitude-based selection*), we fix a uniform distribution for α and use *the impact of an operation on the local validation accuracy* (perturbation) as a criterion to search on the edge. This simplified method is much more efficient given that it only requires evaluation-based search rather than training-based search (optimizing α). In addition, it avoids inserting another bi-level optimization for NAS inside a bi-level optimization for FL, making the framework stable and easy to tune. Finally, this method avoids suboptimal architecture (Wang et al. 2021) lead by magnitude-based selection in differentiable NAS.

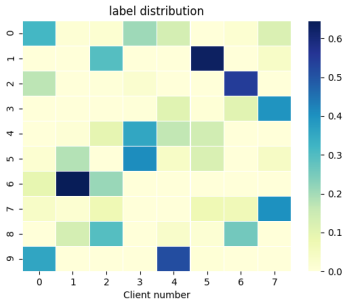
5 Experiments

This section presents the experimental results of the proposed method, SPIDER. All our experiments are based on a non-IID data distribution among FL clients. We have used latent Dirichlet Distribution (LDA), which is a common data distribution used in FL to generate non-IID data across clients (He et al. 2020), (Yurochkin et al. 2019).

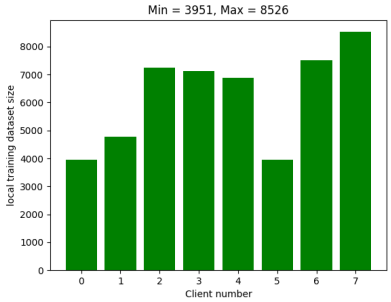
5.1 Experimental Setup

Implementation and Deployment. We implement the proposed method for distributed computing with nine nodes, each equipped with a GPU. We set this as a cross-silo FL setting with one node representing the server and eight nodes representing the clients. These clients nodes can represent real-world organizations such as hospitals and clinics that aim to collaboratively search for personalized architectures for local benefits in a privacy-preserving FL manner.

Task and Dataset. We perform an image classification task on the CIFAR10 dataset that consists of 60000 32x32 color images in 10 classes, with 6000 images per class. We generate non-IID data across clients by exploiting LDA distribution with parameter ($\alpha = 0.2$) for the training data of CIFAR10. The actual data distribution has been shown in figure 2. Sub-figure 2a represents the label distribution across clients, where a darker color indicates more images of that class/label. The other sub-figure 2b represents the total number of data samples preset at each client. In addition to CIFAR10, we also present results with CIFAR100 dataset that consists of 60000 32x32 color images in 100 classes, with 600 images per class. For CIFAR100 dataset, we generate non-IID data across clients by exploiting LDA distribution with parameter ($\alpha = 0.2$) for the training data of CIFAR100.



(a) Label distribution per client



(b) Image distribution per client

Figure 2: CIFAR10: LDA distribution

For personalized architecture experiments with SPIDER, we split the total training data samples present at each client into training (50%), validation (30%), and testing sets (20%). For other personalization schemes used for comparison, we do not need validation data. Therefore, we split the data samples of each client with training (80%) and test (20%) for a fair comparison. In addition, we fix the non-IID dataset distribution in all experiments. We also keep the data split ratio same for both CIFAR10 and CIFAR100 datasets.

5.2 Results on Average Validation Accuracy

Here, we report the comparison of our proposed method, SPIDER, with the other state-of-the-art personalized methods; Ditto, perFedAvg, and Local adaptation for CIFAR10 and CIFAR100 datasets. Since these schemes use a predefined architecture, we use the Reset18 model because of its comparable model size. For CIFAR10 dataset, we also

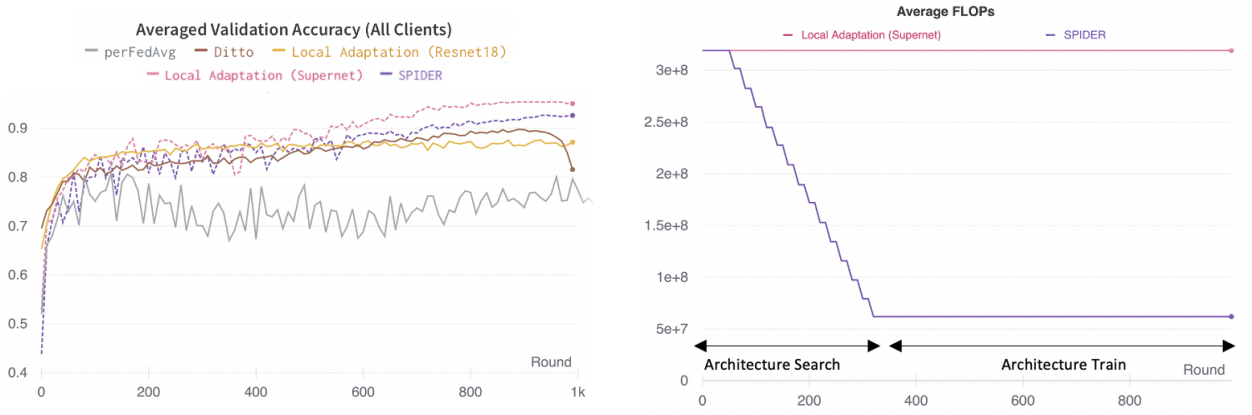


Figure 3: Comparison of our proposed SPIDER with other state-of-the-art personalization methods (Ditto, Local Adaptation, and perFedAvg) for CIFAR10 dataset. The right figure illustrates the architecture search (progressive perturbation of the Supernet) and derived child model's architecture train phase of the proposed method.

Table 1: CIFAR10 dataset - Average local validation Accuracy Comparison of SPIDER with other personalization techniques

Method	**Average Accuracy**	**Parameter Size**	**FLOPs**	**Estimated Model Size**
Local Adapation - Supernet	0.950±0.010	1.9M	319M	104MB
SPIDER	0.926±0.020	345K	62M	14MB
Local Adaptation - Resnet18	0.871±0.025	11M	76M	44MB
Ditto - Resnet18	0.898±0.026	11M	76M	44MB
perFedAvg - Resnet18	0.909±0.022	11M	76M	44MB

explore Local adaptation with the complete Supernet. This exploration will help us investigate how much performance drop we get if we use a smaller but personalized model as compared to a locally adapted complete Supernet.

Average Accuracy for CIFAR10 dataset: In Figure 3, we report average of the validation accuracy calculated on client's test dataset using the personalized architectures for CIFAR10 dataset. The right sub-figure in Fig. 3 illustrates the comparison of the proposed method with the state-of-art methods; Ditto, Local adaptation, and perFedAvg. We note that local adaptation with Supernet provides the highest average accuracy. We expect it from the Supernet because of its 8 operations-based mixed operation formulations. This result helps us investigate the performance reduction with our proposed method. We observe that our proposed method shows a performance reduction of 2.5% in average validation accuracy with the benefit of a reduced average number of FLOPs and model size. The models we use during training are much smaller and require less memory as well as computation.

From empirical results, we observe that the proposed approach of architecture personalization outperforms the other state-of-the-art personalization methods; Ditto, perFedAvg, and Local adaptation with ResNet18 for both CIFAR10 and CIFAR100 datasets. For CIFAR10, among these personalization schemes, perFedAvg yields the highest performance. Although in Figure 3 the performance curve is lower, it gains 90.9% accuracy around 1500 rounds. Compared with perFedAvg, we obtain 1.7% higher average accuracy. Moreover, Ditto and Local adaptation yield 89.8% and 87% average accuracy, respectively. For personalization, the standard deviation (std) is considered an important metric, as average accuracy alone may not represent fairly how well a model is performing across clients. Therefore, we also report the standard deviation for each method in Table 1. Besides the Local Adaptation with Supernet that yields 0.01 standard deviation, lowest among all, our methods yields 0.02 std and outperforms PerFedAvg (0.022), Ditto (0.026) and Local Adaptation (0.025).

Average Accuracy for CIFAR100 dataset: For CIFAR100 dataset, we note that our method outperforms all three methods by yielding higher accuracy around 67% as shown in Figure 4 and Table 2. Similar to our observation with the CIFAR10 dataset, perFedAvg yields the second highest accuracy, 64%. We also observe that contrary to CIFAR10 results, Ditto provided the lowest performance among the three personalization methods. From personalization perspective, we observe that our method achieves the same standard deviation, 0.032, as perFedAvg. However, we achieve this standard deviation with a 3.5% higher mean validation accuracy value. The local adaptation and Ditto yield higher standard deviation, which translates to a lower per-

Table 2: CIFAR100 dataset - Average local validation Accuracy Comparison of SPIDER with other personalization techniques

Method	Average Accuracy	Parameter Size	FLOPs	Estimated Model Size
SPIDER	0.6740±0.032	392K	64M	16MB
Local Adaptation - ResNet18	0.6115±0.033	11M	76M	44MB
Ditto - ResNet18	0.5700±0.033	11M	76M	44MB
perFedAvg - ResNet18	0.6390±0.032	11M	76M	44MB

sonalization (fairness) across clients.

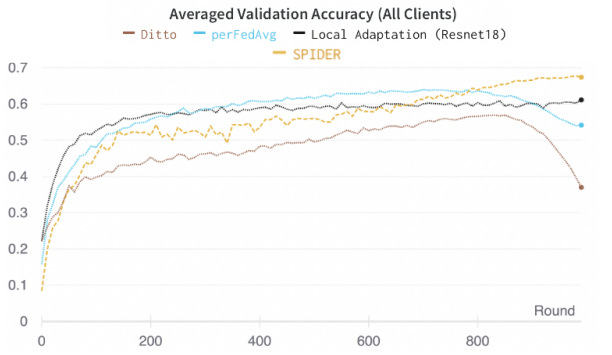


Figure 4: This figure shows the average validation accuracy comparison between our proposed method SPIDER and the other state-of-the-art methods; perFedAvg, Ditto, and local Adaptation with CIFAR100 dataset.

5.3 Results on Efficiency

Average FLOPs In figure 3, we report average flops to gauge the architecture search and train phase. Following Algorithm 1, Phase 1 continues for 60 rounds, Phase 2 for another 260 rounds, Phase 3 for next 680 rounds. We used recovery time of around 20 rounds. In centralized perturbation-based NAS (Wang et al. 2021), it was empirically found that 5 epochs are sufficient for fine-tuning the Supernet. However, from empirical results in FL, we found 20-30 communication rounds to be a reasonable number of rounds for recovery. The Average FLOPs vs. number of rounds figure illustrates the reduction in the size of computations as search proceeds. The reduction in the average size of computations has been from 319M to 62M and 64M for CIFAR10 and CIFAR100, respectively. However, for the local adaptation with Supernet, the average FLOPs remain the same. Likewise, the average parameter size drops from 1.9M to 345K and 392 with CIFAR10 and CIFAR100, respectively.

Model Size Estimated model size includes the memory size of a forward pass and parameter size of that model. It essentially corresponds to how much memory a trained model with trained weights would occupy to obtain inference. Supernet takes the highest amount of space, 104MB, ResNet18 takes 44MB, and the average model size from our proposed method occupies 14MB and 16MB space for CIFAR10 and CIFAR100, respectively. The reason we report this number is that even if Supernet has less parameter size, 1.9M (due to many non-parameterized operations, i.e., skip connection, max pool, avg pool), it requires a vast number of FLOPs to compute mixed operations on the feature maps of all the operations used in DARTs search space. Therefore, the estimated model size can better represent the actual memory/compute of these models. In nutshell, the proposed search yields architectures that are smaller in size and inference efficient, which can be beneficial for many business models.

6 Conclusion

We proposed SPIDER, an algorithmic framework that can search personalized neural architecture for FL. SPIDER specializes a weight-sharing-based global regularization to perform progressive neural architecture search. Experimental results demonstrate that SPIDER outperforms other state-of-the-art personalization methods, and the searched personalized architectures are more inference efficient.

References

Cheng, G.; Chadha, K. N.; and Duchi, J. C. 2021. Fine-tuning in Federated Learning: a simple but tough-to-beat baseline.

Diao, E.; Ding, J.; and Tarokh, V. 2020. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264*.

Dinh, C. T.; Tran, N. H.; and Nguyen, T. D. 2020. Personalized federated learning with moreau envelopes. *arXiv preprint arXiv:2006.08848*.

Fallah, A.; Mokhtari, A.; and Ozdaglar, A. 2020a. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948*.

Fallah, A.; Mokhtari, A.; and Ozdaglar, A. E. 2020b. Personalized Federated Learning: A Meta-Learning Approach. *ArXiv*, abs/2002.07948.

Ghosh, A.; Chung, J.; Yin, D.; and Ramchandran, K. 2020. An Efficient Framework for Clustered Federated Learning. *ArXiv*, abs/2006.04088.

Hanzely, F.; and Richtárik, P. 2020. Federated learning of a mixture of global and local models. *arXiv preprint arXiv:2002.05516*.

He, C.; Annavaram, M.; and Avestimehr, S. 2020a. Fednas: Federated deep learning via neural architecture search. *arXiv e-prints*, arXiv–2004.

He, C.; Annavaram, M.; and Avestimehr, S. 2020b. Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge. *arXiv: Learning*.

He, C.; Li, S.; So, J.; Zeng, X.; Zhang, M.; Wang, H.; Wang, X.; Vepakomma, P.; Singh, A.; Qiu, H.; et al. 2020. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*.

Hu, S.; Xie, S.; Zheng, H.; Liu, C.; Shi, J.; Liu, X.; and Lin, D. 2020. Dsnas: Direct neural architecture search without parameter retraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12084–12092.

Jaafra, Y.; Laurent, J. L.; Deruyver, A.; and Naceur, M. S. 2019. Reinforcement learning for neural architecture search: A review. *Image and Vision Computing*, 89: 57–66.

Jiang, Y.; Konecný, J.; Rush, K.; and Kannan, S. 2019. Improving Federated Learning Personalization via Model Agnostic Meta Learning. *ArXiv*, abs/1909.12488.

Karimireddy, S. P.; Kale, S.; Mohri, M.; Reddi, S. J.; Stich, S. U.; and Suresh, A. T. 2020. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *ICML*.

Li, C.; Yu, Z.; Fu, Y.; Zhang, Y.; Zhao, Y.; You, H.; Yu, Q.; Wang, Y.; and Lin, Y. 2021a. Hw-nas-bench: Hardware-aware neural architecture search benchmark. *arXiv preprint arXiv:2103.10584*.

Li, T.; Hu, S.; Beirami, A.; and Smith, V. 2021b. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, 6357–6368. PMLR.

Li, T.; Sahu, A. K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; and Smith, V. 2018. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*.

Liang, P. P.; Liu, T.; Ziyin, L.; Salakhutdinov, R.; and Morency, L.-P. 2020. Think Locally, Act Globally: Federated Learning with Local and Global Representations. *ArXiv*, abs/2001.01523.

Lin, T.; Kong, L.; Stich, S. U.; and Jaggi, M. 2020a. Ensemble distillation for robust model fusion in federated learning. *arXiv preprint arXiv:2006.07242*.

Lin, T.; Kong, L.; Stich, S. U.; and Jaggi, M. 2020b. Ensemble Distillation for Robust Model Fusion in Federated Learning. *ArXiv*, abs/2006.07242.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Liu, Y.; Sun, Y.; Xue, B.; Zhang, M.; Yen, G. G.; and Tan, K. C. 2021. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*.

Reddi, S.; Charles, Z.; Zaheer, M.; Garrett, Z.; Rush, K.; Konečnỳ, J.; Kumar, S.; and McMahan, H. B. 2020. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*.

Reddi, S. J.; Charles, Z. B.; Zaheer, M.; Garrett, Z.; Rush, K.; Konecný, J.; Kumar, S.; and McMahan, H. B. 2021. Adaptive Federated Optimization. *ArXiv*, abs/2003.00295.

Santra, S.; Hsieh, J.-W.; and Lin, C.-F. 2021. Gradient Descent Effects on Differential Neural Architecture Search: A Survey. *IEEE Access*, 9: 89602–89618.

Sattler, F.; Müller, K.-R.; and Samek, W. 2020. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*.

Sattler, F.; Müller, K.-R.; and Samek, W. 2021. Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 32: 3710–3722.

Singh, I.; Zhou, H.; Yang, K.; Ding, M.; Lin, B.; and Xie, P. 2020. Differentially-private federated neural architecture search. *arXiv preprint arXiv:2006.10559*.

Smith, V.; Chiang, C.-K.; Sanjabi, M.; and Talwalkar, A. 2017a. Federated multi-task learning. *arXiv preprint arXiv:1705.10467*.

Smith, V.; Chiang, C.-K.; Sanjabi, M.; and Talwalkar, A. S. 2017b. Federated Multi-Task Learning. In *NIPS*.

Wang, J.; Liu, Q.; Liang, H.; Joshi, G.; and Poor, H. V. 2020. Tackling the objective inconsistency problem in heterogeneous federated optimization. *arXiv preprint arXiv:2007.07481*.

Wang, K.; Mathews, R.; Kiddon, C.; Eichner, H.; Beaufays, F.; and Ramage, D. 2019. Federated Evaluation of On-device Personalization. *ArXiv*, abs/1910.10252.

Wang, R.; Cheng, M.; Chen, X.; Tang, X.; and Hsieh, C.-J. 2021. Rethinking architecture selection in differentiable NAS. *arXiv preprint arXiv:2108.04392*.

Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; and Keutzer, K. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10734–10742.

Xu, M.; Zhao, Y.; Bian, K.; Huang, G.; Mei, Q.; and Liu, X. 2020. Federated neural architecture search. *arXiv preprint arXiv:2002.06352*.

Yang, L.; Yan, Z.; Li, M.; Kwon, H.; Lai, L.; Krishna, T.; Chandra, V.; Jiang, W.; and Shi, Y. 2020. Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6. IEEE.

Yu, T.; Bagdasaryan, E.; and Shmatikov, V. 2020a. Salvaging federated learning by local adaptation. *arXiv preprint arXiv:2002.04758*.

Yu, T.; Bagdasaryan, E.; and Shmatikov, V. 2020b. Salvaging Federated Learning by Local Adaptation. *ArXiv*, abs/2002.04758.

Yurochkin, M.; Agarwal, M.; Ghosh, S.; Greenewald, K.; Hoang, N.; and Khazaeni, Y. 2019. Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning*, 7252–7261. PMLR.

Zhu, H.; and Jin, Y. 2021. Real-time federated evolutionary neural architecture search. *IEEE Transactions on Evolutionary Computation*.