Personalized Federated Learning in Real-world Healthcare Applications across Different Frameworks

Chenyang Ma* University of Cambridge cm2196@cam.ac.uk

Yuwei Zhang* University of Cambridge yz798@cam.ac.uk

Abstract

Personalized Federated Learning (pFL) serves as a promising solution for the collaborative training of deep neural networks between healthcare institutions to protect privacy and address data heterogeneity. Existing works commonly evaluate proposed pFL strategies on synthesized datasets but their effectiveness in realistic healthcare settings remains under-investigated. This work evaluates four FL/pFL strategies on three real-world medical datasets with implementation under two FL frameworks. New findings are acquired through extensive experiments. Code is made available at our GitHub repo.

1 Introduction

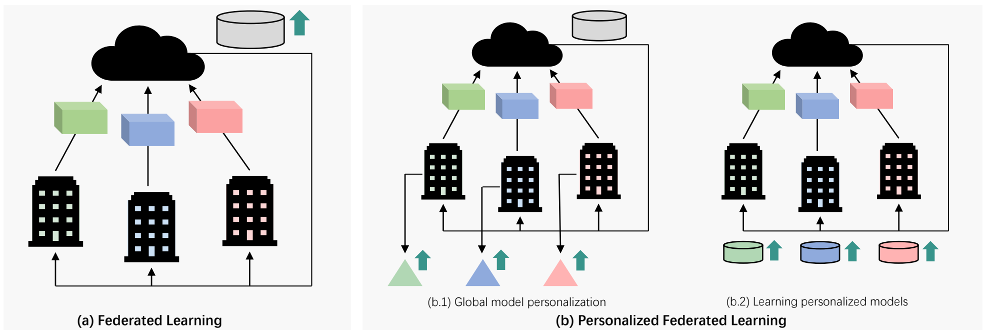
(a) Federated Learning
(b.1) Global model personalization
(b.2) Learning personalized models
(b) Personalized Federated Learning

Figure 1: (a) FL vs. (b) pFL. FL optimizes the global model while pFL optimizes personalized models.

The prominence of deep learning in recent years raises the collective interest of healthcare institutions to train deep neural networks on various medical downstream tasks. However, deep networks are always data-hungry and their performance depends heavily on both the quality and the diversity of the training dataset. In some cases, the local dataset of an individual healthcare institution is too trivial to build a robust network. The privacy-sensitive nature of medical data also prevents multiple healthcare institutions from contributing and migrating their local data to a single storage place [24, 1].

The newly emerged concept of Federated Learning (FL) [38] serves as a promising solution for the above scenario. FL enables the training of a global model utilizing decentralized and locally stored datasets in a privacy-preserving manner. Contrary to the first proposed cross-device FL [38], in which the clients consist of massively parallel Internet of Things (IoT) and mobile devices that can possibly reach 10^{10} [22], our setting is named as cross-silo FL. The clients typically are 2 to 100 [22] legal organizations with mutual incentives to share their dataset and conduct specific tasks.

Compared to cross-device FL, cross-silo FL has the additional challenge of high inter-client dataset heterogeneity [17] since each client possesses a dataset that is collected in different and isolated

* Indicates equal contribution

environments. This issue is exaggerated in medical datasets due to the underlying bias incurred by factors such as population shift, prevalence shift, annotation shift, etc [4]. Thus, instead of training and deploying a single global model, it is more reasonable to build a personalized model for each participating healthcare institution. This approach is known as personalized Federated Learning (pFL). Fig. 1 illustrates the key difference between pFL and FL.

The approaches developed by existing research on pFL can be divided into two main categories (Fig. 1.b.1 and 1.b.2): 1) global model personalization and 2) learning personalized models [44]. In 1), a global model is firstly trained and then locally adapted to each client [30, 18, 48, 49, 29, 23, 28]. In 2), personalized models are learned by modifying the model aggregation strategy [43, 3, 37, 32, 27, 20, 14]. These works commonly used synthesized FL datasets such as FEMNIST [7] and CIFAR-10/100 [26] with disparate partition manners and evaluation metrics. The effectiveness of these approaches in real-world settings (e.g., healthcare), in which the datasets owned by the participating clients can be severely biased with extreme Non-IID distribution, remains under-investigated.

The goal of this work is to evaluate the performance of existing FL/pFL strategies on **real-world** medical datasets. More specifically, we select and evaluate four FL/pFL strategies with increasing personalization capacity: FedAvg [38] $\rightarrow$ FedProx [29] $\rightarrow$ FedBN [30] $\rightarrow$ FedAP [37] on three real-world medical datasets with different characteristics and downstream tasks. Additionally, we are also interested in the performance of these strategies under different FL frameworks. We implement our methods under two FL frameworks: Microsoft PersonalizedFL (MsPFL) [36] and Flower [5].

To the best of our knowledge, this is one of the first few works which conducts cross-silo FL research in a real-world healthcare setting [17, 34, 19]. Overall, the main contributions of this work can be summarized as the following three aspects.

- We evaluate four FL/pFL strategies on three real-world medical datasets in cross-silo settings with nature data partition and extreme Non-IID distribution.
- We implement our methods under two FL frameworks for comparison. We modify, optimize, and add new files to both frameworks' source code.
- We acquire new findings regarding FL/pFL strategies and FL frameworks through extensive experiments.

2 Personalized Federated Learning Strategies

In the section, we provide in-depth descriptions and explain the insight of the four FL/pFL strategies with increasing personalization power. We recommend the readers refer to the original papers for complete technical details.

2.1 FedAvg

McMahan *et al.* [38] introduced FL and proposed FedAvg in 2016, which is the most basic FL strategy and remains to be widely used. FedAvg averages the local stochastic gradient descent model updates of clients. The training algorithm repeats the following procedures: 1. The server sends the global model to sampled clients (with some model initialization method at round 1); 2. Sampled clients train and update models with local datasets; 3. Sampled clients send locally updated models to the server; 4. The server aggregates clients' models by averaging model weights. The core of FedAvg, which is the server aggregation, can be expressed as:

$$w^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{t+1} \tag{1}$$

where K is the number of sampled clients, w_k and n_k are the local weights and size of the local dataset of client k. w is the global model weight, $n = \sum_{k=1}^{K} n_k$, and $\frac{n_k}{n}$ gives the weighted averaging.

Although FedAvg is proved to be empirically effective, it does not provide convergence guarantees when local datasets are heterogeneous [29] and may lead to objective inconsistency, which means the training global model may converge to the stationary point of a mismatched objective function [47].

2.2 FedProx

To solve the above issue, Li *et al.* [29] proposed FedProx in 2018. The idea behind FedProx is simple yet effective: if the updates made on heterogeneous local datasets lead to divergent models and the averaged global model fails to reach the global optimum, we want to restrict the number of local updates. This can be done by adding a proximal term to the objective function of FedAvg such that each sampled client finds a w_k^{t+1} that is a γ_k^t-inexact minimizer:

$$w_k^{t+1} \approx F_k(w) + \frac{\mu}{2}\|w - w^t\|_2 \quad \text{and} \quad F_k(w) := \mathbb{E}_{x_k \sim X_k}[\ell(x_k; w)] \tag{2}$$

where $F_k(w)$ is the local objective of client k over local data distribution X_k, $\ell(x_k; w)$ is the loss of x_k with model weights w. The proximal term, $\frac{\mu}{2}\|w - w^t\|_2$, calculates the normalized weight difference between the local and global model weights. Thus, it imposes more penalties on updates with higher heterogeneity. Intuitively, we can imagine tension ropes between local and global optimums. Experimental results of FedProx suggest it is better to finetune the penalty constant μ based on the model and the dataset, but a reasonable pool can be: $[0.001, 0.01, 0.1, 1]$. FedProx is also proven to grant more robust convergence theoretically compared to FedAvg [29].

2.3 FedBN

Unlike FedProx which tackles data heterogeneity from client-shift, FedBN, proposed by Li *et al.*[30] in 2021, considers feature space and optimizes feature-shift of local datasets. FedBN can be seen as a pFL strategy that fits in the category of global model personalization, as illustrated in Fig. 1.b.1.

FedBN takes on the inspirations from previous works [31, 8] using batch normalization (BN) layers to alleviate domain shifts in domain adaptation. BN is long known for its effectiveness in feature scaling and correcting internal covariate shifts between neural network layers. Similarly, FedBN hypothesized that domain-related knowledge is stored in and can be represented by the statistics of BN layers. Thus, clients can retain their own BN layers to synchronize the feature space distribution. The sole implementation difference between FedBN from FedAvg is that BN layers are excluded from aggregation during communication rounds.

2.4 FedAP

Lu *et al.* [37] (2022) argued that FedBN does not fully use information stored in BN layers and proposed FedAP. FedAP further considers client similarities, measured by distance between local datasets distributions, leveraging BN layers information. Same as other similarity-based pFL strategies [20, 6], the insight of FedAP is that clients with higher similarity should acquire similar models and vice versa. FedAP is a pFL strategy fitting in the category of learning personalized models (Fig. 1.b.2).

FedAP quantifies the client similarities using a weight matrix W. Given K clients, $\mathrm{W} \in \mathbb{R}^{K \times K}$, where $w_{ij} \in [0, 1]$ represents the similarity between client i and j and a value closer to 1 denotes higher similarity. Each row of W sums to 1. W is calculated from the BN layers of a pre-trained network, which is either pre-trained on a large dataset such as ImageNet [13] or pre-trained by FL using several rounds of FedBN [37].

At the beginning of round 1, a pre-trained model is sent to the K clients. Each client computes the BN statistics using the pre-trained network and sends it to the server. The server then computes W. The BN statistics of each channel are treated as Gaussian distribution and the distance is calculated by Wasserstein distance. The server only calculates W once and fixes W for the entire training.

During each training round, the server aggregates $\psi_j^{t+1}, j = 1...K$, containing only non-BN layers. Each client i, $i = 1...K$, obtains unique model $\psi_i^{(t+1)^*}$ according to the weighted sum between i-th row of W and ψ_j^{t+1}, which can be written as:

$$\psi_i^{(t+1)^*} = \sum_{j=1}^{K} w_{ij}\psi_j^{t+1} \tag{3}$$

At the end of each round, the server sends the corresponding personalized model to each client.

3 Federated Learning Frameworks

In the section, we introduce the two FL frameworks from a high level. The framework architectures are shown in Fig. 2.a and 2.b.

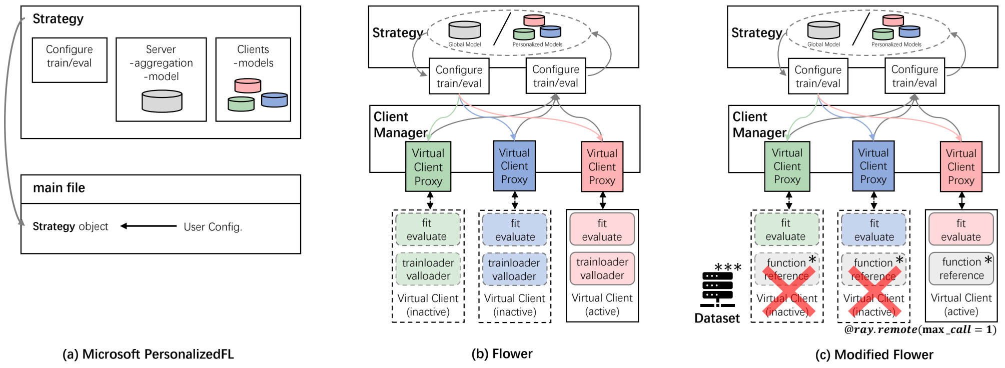

Figure 2: (a) MsPFL vs. (b) Flower. (c) We modify and optimize Flower framework in our implementation, which will be explained in Section 4.2. For (b), we adopt the structure of Fig. **2** of the original Flower [5] paper with revisions. We appreciate the authors of Flower for making the good figure.

3.1 Microsoft PersonalizedFL

MsPFL [1] [36] (Fig. 2.a) is an open-source federated learning framework based on PyTorch [40]. It aims for personalization with the design goals of easy to learn and extend, and has a very simple architecture. MsPFL treats the FL strategy as the highest-level abstraction. Each strategy class stores all training configurations, the global model, and all clients' local models. Each strategy class also defines methods for client train/evaluation, server aggregation, and server evaluation. The main function initializes the strategy class and starts federated training. MsPFL currently supports five strategies: FedAvg, FedProx, FedBN, FedAP, and MetaFed [9]. We run experiments on all five strategies but only include results of the first four for comparison.

3.2 Flower

Flower [2] [5] (Fig. 2.b) is an end-to-end FL framework that is customizable, extendable, and understandable. The architecture of the Flower framework can be partitioned into the server side and the client side. The server side consists of three major components. The *Strategy* abstraction is responsible for training configuration, client selection, model updates, and global model evaluation. The *ClientManager* organizes and communicates with *ClientProxy* objects. Each *ClientProxy* object is associated with a Flower Client. The FL loop coordinates the entire FL process. The client-side mainly comprises *Client* abstraction. Individual Flower Clients have access to the local datasets and predefined models. Flower Clients can be created by either Virtual Client Engine (VCE) or Edge Client Engine. VCE enables the virtualization of Flower Clients and the maximized usage of hardware by allocating the appropriate amount of resources to Flower Clients according to their computation requirements. This functionality is especially helpful in realistic cross-silo FL settings in which the local dataset held by each client is extremely divergent.

In addition to the robustness of Flower's single-node (machine) FL simulation, Flower also supports other functionalities such as multi-node execution, experiment on heterogeneous clients, high scalability, and multi-framework workloads by facilitating language-agnostic, communication-agnostic, and privacy-agnostic design principles. We do not fully exploit these in our cross-silo FL setting.

4 Implementation Details

In this section, we explain the implementation details. We put emphasis on the new codes, modifications, and optimizations made by us within the two FL frameworks.

4.1 Microsoft PersonalizedFL

Using MsPFL Framwork. We utilize the *communication* function inside *PersonalizedFL/alg/core*, in which strategies are defined as Python classes with *server_model* and *client_models* as attributes. In each round, we use *algclass.client_train* to train models locally and *algclass.server_aggre* to average and set model weights leveraging *state_dict* attributes of PyTorch [40] models.

We load our own *evaluate_model_on_tests* function to evaluate the model performance on both local and global test sets as we use real-world medical datasets (see Section 5.1).

[1] https://github.com/microsoft/PersonalizedFL [2] https://flower.dev

MsPFL Codebase Modifications and Optimizations. We modify the source code of *algclass* definitions so that we can load customized models, loss functions, optimizers, and evaluation metrics. We also make substantial changes to the source code of FedAP for obtaining the pre-trained models.

FedAvg, FedProx, FedBN. In every communication round, FedAvg averages the client model weights, generates a new server model, and sends it back to all clients. A different training function *train_prox* is defined in the framework for FedProx where the normalized weight difference is added to the loss. FedBN skips all BN layers during communication by checking if the keyword "bn" or "norm" is in *model.state_dict().keys()*.

FedAP. We need to calculate features using a pre-trained model. Different from the original implementation of MsPFL, in which a model is trained on part of the centralized dataset and may cause privacy concerns, we follow the practice of the original FedAP [37] paper. We have a different model for each dataset (three in total, see Section 5.1). We use EfficientNet-B0 [45] pre-trained on ImageNet [13]. We train two other models in FedBN for several rounds and use the server-side aggregated model as the pre-trained model.

For all three pre-trained models, we define unique *getallfea* functions to generate runtime BN layers outputs as a feature list (the input of *set_client_weight*). For the 2-layer MLP, we extract the only BN layer. For EfficientNet-B0 [45], we implement depth-first search (DFS) to traverse over all layers and extract the outputs of 38 BN layers. Our approach can be generalized to all models that are forwarded sequentially. For 3D U-Net [10], DFS is not applicable because of the skip connections. We first traverse over all layers in encoding blocks and store the skip connections. We then traverse over the bottom block and the decoding blocks.

4.2 Flower

Using Flower Framework. We use L46 lab 3 code as our starting point. We have a main file named *L46_code.py*. We create local models of type *torch.nn.Module* with *get_weights* and *set_weights* functions to get model weights as and set model weights from a list of *flwr.common.NDArrays*. We define our own *train* and *test* functions for model training and testing.

We create a custom Flower Client of type *flwr.client.Client*, in which we define our own *get_parameters*, *set_parameters*, *fit*, and *evaluate* functions. These four functions depend on the functions we mentioned in the above paragraph. We create a custom server-side evaluation function to evaluate the global test set. We also create a function *start_experiment* which sets all parameters, configures *flwr.server.strategy*, creates *client_fn*, and calls *flwr.simulation.start_simulation*.

Flower Codebase Modifications and Optimizations. We make changes in the Flower source code in Anaconda3 environment python site-packages: *anaconda3/envs/***/lib/python3.8/site-packages/flwr*. We implement our own FedProx, FedBN, and FedAP strategies.

Although Flower's VCE, implemented with Ray [39] framework, is resource-aware when it launches Flower Clients either for fitting or evaluation, we experience SSD memory and CUDA out-of-memory issues with Ray because one of our datasets is large in size (see Section 5.1). To solve these two issues, we modify and optimize both our main file and Flower source code, as shown in Fig. 2c.

For SSD memory issue, we think the reason is Ray moves Flower Clients, which are temporary objects, that are too large in size once the object store (RAM) is full to external storage (disk space). Ray refers to this behavior as "object spilling" [3]. If too much spilling takes place, the disk space may be out of memory. To tackle this, when we create custom Flower Client in *L46_code.py*, instead of declaring *self.trainloader* and *self.valloader* within *__init__*, we pass a function reference named as *self.client_datasets*. The global function *client_datasets* returns corresponding trainloader and valloader to each Flower Client based on *self.cid* only when Client *fit/evaluate* are called.

For the causes of CUDA out-of-memory, *@ray.remote* in *flwr/simulation/ray_transport/ray_client_proxy.py* is repeatedly called during simulation [4]. Flower Clients still exist in GPUs after *fit/evaluate* finish, which accumulate and drain CUDA memory. To tackle this, we change *@ray.remote* to *@ray.remote(max_calls=1)* in *ray_client_proxy.py* so that ray worker is removed after each Flower Client terminates.

[3] https://docs.ray.io/en/releases-1.12.1/ray-core/objects/object-spilling.html

[4] https://github.com/adap/flower/pull/1384

Table 1: Summary of datasets and FL partition.

Dataset	Data No.	Clients No.	Client Train Partition	Client Test Partition	Input Dimension	Size
Fed-Heart-Disease	740	4	[199, 172, 30, 85]	[104, 89, 16, 45]	13	40K
Fed-IXI	566	3	[249, 145, 59]	[62, 36, 15]	$48 \times 60 \times 48$	444M
Fed-ISIC2019	23,247	6	[9930, 3163, 2691, 1807, 655, 351]	[2483, 791, 672, 452, 164, 88]	$200 \times 200 \times 3$	9G

Table 2: Experimental setup.

Dataset	Model	Optimizer	Batch Size	LR	μ	Local Iter	Round	Metric	Loss
Fed-Heart-Disease	1-layer MLP 2-layer MLP + BN	Adam [25]	4	0.001	0.01	50	30	Accuracy	BCE Loss
Fed-IXI	3D U-Net [10]	AdamW [35]	2	0.001	0.01	10	75	DICE [15]	DICE Loss
Fed-ISIC2019	EfficientNet-B0 [45]	Adam	64	0.005	0.01	20	47	Balanced Accuracy	Weighted Focal Loss [33]

FedAvg. We use Flower's built-in FedAvg strategy without modifications. We configure FedAvg in *start_experiment* function and pass it to *flwr.simulation.start_simulation*.

FedProx. We modify the *fit* function of the custom Flower Client. We first call *self.set_parameters* and *train* inside *fit*. After training, we can calculate the proximal term, which is the normalized weight difference between the global model and the local trained model as they are both *flwr.common.NDArrays*. Then, we reset the model weights by calling *self.set_parameters*, and train the model again by concatenating *torch.nn.Identity(proximal_term)* to the predefined loss criterion.

FedBN. We modify the *get_weights* and *set_weights* functions of the models to skip all BN layers in *self.state_dict().keys()*. This can be done by checking if the keyword "bn" or "norm" is in the keys.

FedAP. We create a new function *aggregate_ap* in *flwr.server.strategy.aggregate.py*. For simplicity and fair comparison, we hardcode the three weight matrices calculated in the MsPFL framework inside *aggregate_ap*. We aggregate the model weights from all clients using these matrices and return a list of *flwr.common.NDArrays* with length equal to the number of clients.

We create a new strategy *flwr.server.strategy.fedap.py* with class *FedAP(Strategy)*. We make changes in *aggregate_fit*. We call *aggregate_ap* and convert the returned list of *flwr.common.NDArrays* into a list of parameters. We return the list of parameters in *aggregate_fit*. Since we change the return type of *aggregate_fit*, in the *set_parameters* function of our custom Flower Client, we index the parameters from the list of aggregated parameters according to *self.cid*.

5 Experiments

In this section, we first introduce the datasets and benchmarks of our experiments in Section 5.1. We then report our experimental setup, comparison methods, and training procedure in Section 5.2. Finally, we reveal our experimental results in Section 5.3.

5.1 Datasets and benchmarks for real-world healthcare settings

Few benchmarks are available for cross-silo FL. We use FLamby (Federated Learning AMple Benchmark of Your cross-silo strategies) [17], a very recently released and the first benchmark for realistic healthcare cross-silo FL setting. FLamby has seven real-world medical datasets with natural partitions (i.e., the local datasets are from different real-world hospitals). Each client has a predefined partition of the local train and test set. The server has a predefined global test set.

For each medical dataset, FLamby provides a preprocessing approach (if required), baseline model, evaluation metric, loss function, and training hyperparameters summarized and inherited from previous research works. These can be loaded by FLamby's helper functions implemented with PyTorch [40]. We directly use these for a fair comparison.

We select three datasets with different characteristics and downstream tasks for experiments. Datasets information is summarized in Table. 1. The following describes the content of each chosen dataset.

Fed-Heart-Disease. This tabular dataset [2] contains patient info to predict the presence of heart disease, which is a binary classification task. The baseline model is a 1-layer MLP. To include a BN layer that is necessary for FedBN and FedAP, we add a hidden layer. The new model is a 2-layer MLP with hidden dimension 32. We put ReLU after BN.

Table 3: Quantitative results. Ms = Microsoft PersonalizedFL, Flwr = Flower. The centralized baseline given by FLamby is in **bold** at the rightmost column. We highlight performance differences of more than 5.00 between the two frameworks in red. For each dataset, strategies with the best and the second-best average results are in blue and violet.

Dataset	Strategy	Client 0		Client 1		Client 2		Client 3		Client 4		Client 5		Average		Server		Baseline
		Ms	Flwr	Ms	Flwr	Ms	Flwr	Ms	Flwr	Ms	Flwr	Ms	Flwr	Ms	Flwr	Ms	Flwr	FLambly
Fed-Heart-Disease (1-layer MLP)	FedAvg	75.96	78.20	76.40	69.66	75.00	75.00	64.44	68.89	-	-	-	-	72.95	72.94	77.17	78.48	**79.13**
	FedProx	75.00	78.85	75.28	73.03	81.25	75.00	64.44	68.89	-	-	-	-	73.99	73.94	76.77	77.56	
	FedBN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	FedAP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Fed-Heart-Disease (2-layer MLP + BN)	FedAvg	66.99	67.95	59.18	59.17	91.67	87.50	83.70	79.55	-	-	-	-	75.38	73.54	70.87	71.75	**-**
	FedProx	66.03	66.35	61.80	58.41	89.58	81.25	78.52	77.27	-	-	-	-	73.98	70.82	68.77	71.49	
	FedBN	72.12	68.27	78.65	72.19	93.75	93.75	75.56	81.82	-	-	-	-	80.02	79.01	-	-	
	FedAP	72.12	68.59	78.65	76.36	93.75	93.75	77.78	78.79	-	-	-	-	80.57	79.37	-	-	
Fed-IXI	FedAvg	98.92	98.36	98.98	98.58	98.69	98.38	-	-	-	-	-	-	98.86	98.44	98.90	98.40	**98.68**
	FedProx	98.81	98.40	98.89	98.48	98.68	98.32	-	-	-	-	-	-	98.79	98.40	98.82	98.28	
	FedBN	98.91	98.41	98.96	98.55	98.71	98.40	-	-	-	-	-	-	98.86	98.45	-	-	
	FedAP	98.92	98.50	98.96	98.57	98.74	98.41	-	-	-	-	-		98.88	98.49	-	-	
Fed-ISIC2019	FedAvg	79.13	77.35	71.16	67.68	64.86	63.15	54.70	52.36	45.87	46.46	73.35	74.34	64.84	63.56	72.70	74.05	**75.55**
	FedProx	76.38	75.53	66.94	62.81	65.15	64.29	53.64	53.92	47.76	50.58	76.46	77.38	64.39	64.09	70.81	73.35	
	FedBN	77.46	77.47	58.63	61.97	73.89	74.15	70.67	73.39	68.79	68.18	73.99	73.16	70.57	71.39	-	-	
	FedAP	82.61	83.80	71.60	69.24	81.62	79.81	73.40	74.06	69.25	69.44	75.15	74.62	75.60	75.16	-	-	

Fed-IXI. This 3D imaging dataset comprises a collection of T1WI data (a type of magnetic resonance imaging) for brain mask prediction, which is a 3D segmentation task. The baseline model is a 3D U-Net [10]. This dataset is extracted from the IXI database [5], and was previously released by Perez *et al.* [41] under the name IXITiny.

Fed-ISIC2019. This 2D imaging dataset [46, 11, 12] consists of dermoscopy images for melanoma class prediction, which is a multi-class classification task. The baseline model is EfficientNet-B0 [45] pre-trained on ImageNet [13].

5.2 Experimental setup, comparison methods, and training

Table. 2 concludes our experimental setup. All follow FLamby [17] benchmarks as explained in Section 5.1 and random seed is set to 0. We use centralized training results from FLamby [17] as our baseline. For all experiments, we train on one NVIDIA A100 GPU. For Adam [25] and AdamW [35] optimizers, all parameters except learning rate (LR) follow the default setting of *torch.optim* [40].

5.3 Quantitative results

Table. 3 demonstrates the quantitative results of the four pFL strategies on three medical datasets across two FL frameworks. Discussion of the results is in Section 6. We also show the weight matrices computed within FedAP [37] in Fig. 3.

$$\begin{vmatrix} 0.5000 & 0.2085 & 0.1173 & 0.1741 \\ 0.2036 & 0.5000 & 0.1196 & 0.1768 \\ 0.1574 & 0.1645 & 0.5000 & 0.1781 \\ 0.1784 & 0.1856 & 0.1360 & 0.5000 \end{vmatrix}$$

Fed-Heart-Disease

$$\begin{vmatrix} 0.5000 & 0.2090 & 0.2910 \\ 0.2842 & 0.5000 & 0.2158 \\ 0.3236 & 0.1764 & 0.5000 \end{vmatrix}$$

Fed-IXI

$$\begin{vmatrix} 0.5000 & 0.0781 & 0.0303 & 0.0528 & 0.0504 & 0.2885 \\ 0.0778 & 0.5000 & 0.0482 & 0.1561 & 0.1405 & 0.0774 \\ 0.0603 & 0.0965 & 0.5000 & 0.1396 & 0.1440 & 0.0595 \\ 0.0285 & 0.0845 & 0.0378 & 0.5000 & 0.3212 & 0.0280 \\ 0.0277 & 0.0776 & 0.0397 & 0.3274 & 0.5000 & 0.0276 \\ 0.2895 & 0.0780 & 0.0299 & 0.0521 & 0.0504 & 0.5000 \end{vmatrix}$$

Fed-ISIC2019

Figure 3: The weight matrices computed within FedAP on three medical datasets. In each weight matrix, $[i, j]$ represents the similarity between client i and client j.

6 Discussion

In this section, we explain our observations and findings from the experimental results.

6.1 Performance difference between frameworks

The results given by MsPFL and Flower generally agree. The average performance of clients differ little across all three datasets. The best and the second-best strategies are consistent in MsPFL and

[5] https://brain-development.org/ixi-dataset

Flower. The differences between individual clients are the largest in Fed-Heart-Disease with several results differing by more than 5.00. We think the reason is Fed-Heart-Disease is a small tabular dataset with small batch size, which makes the training more stochastic.

Interestingly, for all three datasets and strategies, Flower gives slightly lower average client performance and slightly higher server-side evaluation ($\sim$1.00). Normally, we assume better client models will lead to a better aggregated global model. We are not certain about the reasons. We hypothesize it may be due to: 1) Flower provides a much more realistic simulation environment than MsPFL, which comes at a cost. VCE saves the computation resources used by clients but also imposes restrictions. On the other hand, Flower has stronger server-side infrastructures and implementations; 2) Our implementation differences; 3) Although we use the same type of GPU, the specific GPU provided by Cambridge HPC may be in different status; 4) Purely training fluctuations (less likely).

6.2 Effectiveness of personalized federated learning strategies

Comparing two FL strategies, FedProx performs worse than FedAvg in all three datasets for both average client performance and server evaluation with one exception (original Fed-Heart-Disease with 1-layer MLP), which is unexpected. We think it is because the data heterogeneity caused by client-shift is less significant than feature-shift in real-world medical datasets.

For the two pFL strategies, FedBN and FedAP gain large improvements on average client performance compared to FedAvg on Fed-Heart-Disease and Fed-ISIC2019. FedAP proves to be very robust as it outperforms FedBN by a large margin, and its average client performance is comparable to the centralized baseline. The effectiveness of pFL strategies, FedBN and FedAP, reveal the issue of feature-shift in real-world medical datasets. The bias in medical datasets should not be overlooked.

For Fed-IXI, the differences between four FL/pFL strategies are almost negligible. We think the reason is the segmentation task performed on Fed-IXI is too simple.

Comparing the results of using two models (1-layer MLP and 2-layer MLP with BN) on Fed-Heart-Disease, the server-side evaluation of FedAvg and FedProx decreases a lot after adding extra layers. We explain this as BN layers are designed to deal with the vanishing gradient problem in deep neural networks and they do not work well for shallow network designs; however, it is interesting that the average client performance becomes higher than the server performance, and it increases for FedAvg. Also, FedBN and FedAP have great performance gain compared to FedAvg and FedProx using 2-layer MLP with BN. Thus, we conclude that by adding BN layers, the benefits outweight the adverse effects for personalization even for shallow networks.

6.3 Limitations and Future Directions

Due to limited time and computing resources, we are only able to compare four FL/pFL strategies on three datasets. There are considerable amount of existing pFL strategies including MetaFed [9], pFedMe [16], APFL [14], etc. that are yet to be evaluated on real-world datasets.

In addition, our implementations in MsPFL and Flower frameworks are sufficient for our experiments; however, modifications and refinements need to be made if we want to merge our implementations into the official codebase.

Future directions of this work include the evaluation of more pFL strategies and study their application scenarios. In this way, we are able to provide suggestions for researchers regarding strategy choices and provide insights to develop new pFL strategies.

Meanwhile, quantifying bias in medical datasets [4, 21, 42] is an important topic in the research community. By observing the amount of performance gains using pFL strategies, this may be a new method to quantify the underlying bias.

7 Conclusion

We evaluate four FL/pFL strategies of different personalization capacity on three real-world medical datasets across two frameworks. We acquire new findings through extensive experiments.

Acknowledgements. We thank Tong Xia for recommending the FLamby benchmark and Alexandru-Andrei Iacob for the helpful discussion on how to solve the SSD memory and CUDA out-of-memory issues with Ray inside the Flower framework.

References

[1] Sheri A. Alpert. Protecting medical privacy: Challenges in the age of genetic information. *Journal of Social Issues*, pages 301–322, 2003.

[2] Janosi Andras, Steinbrunn William, Pfisterer Matthias, and Detrano Robert. Heart disease data set. 1988.

[3] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers. *arXiv*, abs/1912.00818, 2019.

[4] Mélanie Bernhardt, Charles Jones, and Ben Glocker. Potential sources of dataset bias complicate investigation of underdiagnosis by machine learning algorithms. *Nature Medicine*, pages 1157–1158, 2022.

[5] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. Flower: A friendly federated learning research framework. *arXiv*, abs/2007.14390, 2020.

[6] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *IEEE International Joint Conference on Neural Networks*, pages 1–9, 2020.

[7] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. B. McMahan, Virginia Smith, and Ameet S. Talwalkar. Leaf: A benchmark for federated settings. *arXiv*, abs/1812.01097, 2018.

[8] Woong-Gi Chang, Tackgeun You, Seonguk Seo, Suha Kwak, and Bohyung Han. Domain-specific batch normalization for unsupervised domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7354–7362, 2019.

[9] Yiqiang Chen, Wang Lu, Xin Qin, Jindong Wang, and Xing Xie. Metafed: Federated learning among federations with cyclic knowledge distillation for personalized healthcare. In *FL-IJCAI Workshop*, 2022.

[10] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 424–432, 2016.

[11] Noel CF Codella, David Gutman, M Emre Celebi, Brian Helba, Michael A Marchetti, Stephen W Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, et al. Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic). In *IEEE International Symposium on Biomedical Imaging*, pages 168–172, 2018.

[12] Marc Combalia, Noel C. F. Codella, Veronica Rotemberg, Brian Helba, Verónica Vilaplana, Ofer Reiter, Allan C. Halpern, Susana Puig, and Josep Malvehy. BCN20000: dermoscopic lesions in the wild. *arXiv*, abs/1908.02288, 2019.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern*, pages 248–255, 2009.

[14] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. Adaptive personalized federated learning. *arXiv*, abs/2003.13461, 2020.

[15] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, pages 297–302, 1945.

[16] Canh T. Dinh, Nguyen Hoang Tran, and Tuan Dung Nguyen. Personalized federated learning with moreau envelopes. In *Advances in Neural Information Processing*, 2020.

[17] Jean Ogier du Terrail, Samy-Safwan Ayed, Edwige Cyffers, Felix Grimberg, Chaoyang He, Regis Loeb, Paul Mangold, Tanguy Marchand, Othmane Marfoq, Erum Mushtaq, Boris Muzellec, Constantin Philippenko, Santiago S. Silva R., Maria Telenczuk, Shadi Albarqouni, Salman Avestimehr, Aurélien Bellet, Aymeric Dieuleveut, Martin Jaggi, Sai Praneeth Karimireddy, Marco Lorenzi, Giovanni Neglia, Marc Tommasi, and Mathieu Andreux. Flamby: Datasets and benchmarks for cross-silo federated learning in realistic healthcare settings. *arXiv*, abs/2210.04620, 2022.

[18] Moming Duan, Duo Liu, Xianzhang Chen, Renping Liu, Yujuan Tan, and Liang Liang. Self-balancing federated learning with global imbalanced data in mobile systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 59–71, 2021.

[19] Pengfei Guo, Puyang Wang, Jinyuan Zhou, Shanshan Jiang, and Vishal M. Patel. Multi-institutional collaborations for improving deep learning-based magnetic resonance image reconstruction using federated learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2423–2432, 2021.

[20] Yutao Huang, Lingyang Chu, Zirui Zhou, Lanjun Wang, Jiangchuan Liu, Jian Pei, and Yong Zhang. Personalized cross-silo federated learning on non-iid data. In *AAAI*, pages 7865–7873, 2021.

[21] Jeremy Irvin, Pranav Rajpurkar, Michael Ko, Yifan Yu, Silviana Ciurea-Ilcus, Chris Chute, Henrik Marklund, Behzad Haghgoo, Robyn L. Ball, Katie S. Shpanskaya, Jayne Seekins, David A. Mong, Safwan S. Halabi, Jesse K. Sandberg, Ricky Jones, David B. Larson, Curtis P. Langlotz, Bhavik N. Patel, Matthew P. Lungren, and Andrew Y. Ng. Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *AAAI*, pages 590–597, 2019.

[22] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, pages 1–210, 2021.

[23] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143, 2020.

[24] Mehmet Kayaalp. Patient privacy in the era of big data. *Balkan Medical Journal*, pages 8–17, 2018.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2012.

[27] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv*, abs/1910.03581, 2019.

[28] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 10713–10722, 2021.

[29] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Machine Learning and Systems*, 2020.

[30] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. Fedbn: Federated learning on non-iid features via local batch normalization. In *International Conference on Learning Representations*, 2021.

[31] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. In *International Conference on Learning Representations, Workshop Track*, 2017.

[32] Paul Pu Liang, Terrance Liu, Ziyin Liu, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *arXiv*, abs/2001.01523, 2020.

[33] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision,*, pages 2999–3007, 2017.

[34] Quande Liu, Cheng Chen, Jing Qin, Qi Dou, and Pheng-Ann Heng. Feddg: Federated domain generalization on medical image segmentation via episodic learning in continuous frequency space. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1013–1023, 2021.

[35] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[36] Wang Lu and Jindong Wang. Personalizedfl: Personalized federated learning toolkit. `https://github.com/microsoft/PersonalizedFL`.

[37] Wang Lu, Jindong Wang, Yiqiang Chen, Xin Qin, Renjun Xu, Dimitrios Dimitriadis, and Tao Qin. Personalized federated learning with adaptive batchnorm for healthcare. *IEEE Transactions on Big Data*, page 1, 2022.

[38] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[39] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *Operating Systems Design and Implementation*, pages 561–577, 2018.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[41] Fernando Pérez-García, Rachel Sparks, and Sébastien Ourselin. Torchio: A python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning. *Computer Methods and Programs in Biomedicine*, page 106236, 2021.

[42] Laleh Seyyed-Kalantari, Haoran Zhang, Matthew B A McDermott, Irene Y Chen, and Marzyeh Ghassemi. Underdiagnosis bias of artificial intelligence algorithms applied to chest radiographs in under-served patient populations. *Nature Medicine*, 2021.

[43] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017.

[44] Alysa Ziying Tan, Han Yu, Li zhen Cui, and Qiang Yang. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[45] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114, 2019.

[46] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data*, pages 1–9, 2018.

[47] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. In *Advances in Neural Information Processing Systems*, 2020.

[48] Qiong Wu, Xu Chen, Zhi Zhou, and Junshan Zhang. Fedhome: Cloud-edge based personalized federated learning for in-home health monitoring. *IEEE Transactions on Mobile Computing*, pages 2818–2832, 2022.

[49] Miao Yang, Ximin Wang, Hongbin Zhu, Haifeng Wang, and Hua Qian. Federated learning with class imbalance reduction. In *IEEE European Signal Processing Conference*, pages 2174–2178, 2021.