PFedEdit: Personalized Federated Learning via Automated Model Editing

Haolin Yuan[1], William Paul[2], John Aucott[3], Philippe Burlina[2], and Yinzhi Cao[1]

[1] Johns Hopkins University {hyuan4, yinzhi.cao}@jhu.edu
[2] Johns Hopkins Applied Physics Lab {william.paul, philippe.burlina}@jhuapl.edu
[3] Johns Hopkins University School of Medicine jaucott2@jhmi.edu

Abstract. Federated learning (FL) allows clients to train a deep learning model collaboratively while maintaining their private data locally. One challenging problem facing FL is that the model utility drops significantly once the data distribution gets heterogeneous, or non-i.i.d, among clients. A promising solution is to personalize models for each client, e.g., keeping some layers locally without aggregation, which is thus called personalized FL. However, previous personalized FL often suffer from suboptimal utility because their choice of layer personalization is based on empirical knowledge and fixed for different datasets and distributions. In this work, we design PFedEdit, the first federated learning framework that leverages automated model editing to optimize the choice of personalization layers and improve model utility under a variety of data distributions including non-i.i.d. The high-level idea of PFedEdit is to assess the effectiveness of every global model layer in improving model utility on local data distribution once edited, and then to apply edits on the top-k most effective layers. Our evaluation shows that PFedEdit outperforms six state-of-the-art approaches on three benchmark datasets by 6% on the model's performance on average, with the largest accuracy improvement being 26.6%. PFedEdit is open-source and available at this repository: https://github.com/Haolin-Yuan/PFedEdit

1 Introduction

Federated learning (FL) [13,15,20,31,33] has emerged as a widely used training algorithm of deep learning, especially in the image domain, due to its practical applications in real-world scenarios and its impressive performance. A federated learning training scenario entails multiple local clients and a global server. Each client trains their local model using their private image data. After performing local training, each client updates their local model weights to the global server for aggregation with other clients' model weights. Federated learning allows clients to train a collaborative model that performs well on multi-class tasks—even if each client only possesses a few classes of images—via only sharing models but not local training images.

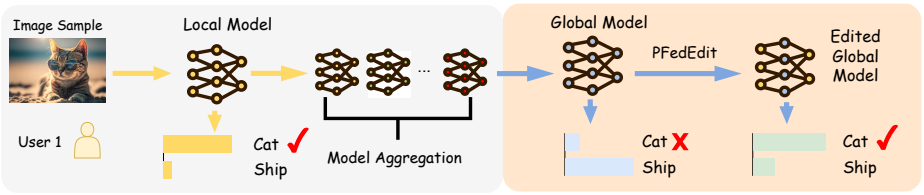


Fig. 1: High-level idea of PFedEdit (PFedEdit edits the global model returned from FL server to fit the local client's distribution.)

As the goal of federated learning is to learn a global model for all devices cooperatively, if the data distributions among clients are similar, the global model is supposed to perform better than local models that are solely trained on local data. However, once the data distribution becomes skewed, or non-i.i.d., e.g., some clients may possess image samples of only one or two classes or some possess many more image samples from one class than others, the global model becomes ineffective to be employed as local models. The reason is that the local data distribution at one client could be dramatically different from others, thus causing local model weights to adversely affect each other during the model aggregation.

Multiple works [11, 14, 16] have been proposed to improve the FL model accuracy under the non-i.i.d. distribution by optimizing local models. They either regularize local models to maximize the agreement between clients [14], or reduce clients' weights drift from the server end [11, 16] for a better global model that generalizes on each local distribution. While such works improved model's accuracy, the model is still uniform across different clients, and cannot be tailored based on different clients' data distribution. Therefore, one natural step is a so-called personalized FL [2, 4, 18, 36], which enables each user to have personalized layers that help customize local models to their data distributions. Those personalized layers are retained locally each communication round and are not updated from the global model, thereby preventing being aggregated with others and avoiding bias towards other clients' data distributions. The choices of those personalized layers are various but are all pre-defined before the training, e.g., it can be linear layer [2], batch normalization layer [4], or specially designed layer [36].

However, there are still two challenges that remain for personalized FLs despite the accuracy improvement. First, the personalized layers chosen in previous personalized FL works [2, 4, 18, 36] are inspired by empirical findings and fixed, implying that those choices may not be optimal when given different data distributions or datasets. That is, while the weights of those layers might be personalized, the locations and types of those layers are not, e.g., either linear or batch normalization layers in certain locations of a learning model. Second, previous personalized works are limited in their application scenarios as they assume certain model structure [18], which may not be used for certain clients.

In this paper, to address these two challenges, we design a framework that automatically locates and chooses layers to be personalized. We propose the first personalized federated learning framework, called PFedEdit, which leverages automated model editing [3,5,21,23] to optimize the layer choosing and improve model utility under different settings including non-i.i.d. The high-level idea of PFedEdit, as shown in Figure 1, is to look for model layers that contain knowledge about local distribution that benefits clients the most, and by editing those layers, PFedEdit mitigates the negative influence brought by other clients, thus tailoring the global model weights towards the local model weights in each round. Specifically, PFedEdit assesses which layers of the global model are the most effective in maintaining the local distribution and then marks them as target layers. PFedEdit then edits the top-k target layers to tailor the global model weights closer to local models thus improving the model utility. PFedEdit does not make any assumptions on model selection and thus is model-structure agnostics.

We performed extensive experiments on three benchmark datasets, namely CIFAR-10 [12], CIFAR-100 [12], and Lyme dataset [35], and we compared PFedEdit with FedAvg [20] and other six state-of-the-art personalized federated learning approaches. Our experimental results demonstrate that PFedEdit outperforms all SOTA personalized federated learning works on all the datasets, with an average 6% accuracy improvement.

In summary, our contributions are as follows:

- We design PFedEdit, the first personalized federated learning framework that optimizes personalized layer selections for each client using model editing.
- PFedEdit is adaptable to all model structures because it does not impose any assumption on model selection.
- We conduct extensive experiments to show that PFedEdit outperforms other SOTA personalized federated works on different benchmark datasets.

2 Related Work

Federated Learning. Federated learning [13, 15, 20, 31] (FL) is first proposed to predict user's text input on portable devices and it has been widely deployed in other industries, such as medical imaging [19], Internet of Things (IoT) [24], and block-chain [25]. It allows multiple clients to collaboratively train a model that can perform on samples from any clients' data distribution. Two common approaches in federated learning (FL) are FedSGD [28] and FedAvg [20]. FedSGD optimizes global weights by back-propagating each local gradient, while FedAvg allows for multiple local training epochs and client-side optimization. In FedAvg, the server's role is limited to averaging and distributing the aggregated weights. While FedAvg reduces communication costs compared to FedSGD, it lacks a convergence guarantee, and heterogeneous data distribution among clients can significantly reduce the global model's utility Therefore, various works [2, 4, 14, 18, 32, 36] have been proposed to solve such utility degradation caused by data heterogeneity. One direction [14] is to optimize local

model either via personalized objective functions; yet, another, called personalized FL [2,4,18,36], allows clients to have personalized model layers that can be kept locally. As a comparison, PFedEdit is the first work that optimizes and personalizes the choice of model layers with further improved utilities of FL, especially under non-i.i.d. settings.

Model Editing. Model editing [3, 5, 17, 23, 26] involves adding, removing, or adjusting layers and parameters of a model architecture to improve its performance or adapt it to new tasks. Specifically, let h_i be the i-th hidden states of any model, $\{h_l | l \in [1, L]\}$, where L is the number of layers in model, model editing allows users to find each hidden state's contribution towards a specific prediction. It mainly contains three runs: i). clean run, in which the statistics of each state are recorded when fed a factual prompt x; ii). corrupted run, in which each hidden layer is added Gaussian noise such that $h_l := h_l + \epsilon$. Because the hidden layer is obfuscated by the added noise, the model will output incorrect predictions; iii). restoration run, in which the model gives predictions with noised hidden states recovered to clean one at a time to find out which layer is related to producing factual knowledge. Following all three steps, users can find out the causal importance of all model layers in the computation graph and edit weights of those knowledge-related layers to alter the final model output.

There are different applications of model editing. For example, model editing can be employed on large language models (LLMs) to facilitate the interpretation of LLMs as black boxes when generating particular content or to guide LLMs in producing desired outputs [22, 30]. ROME [22] edits LLMs' linear layers to modify some learned factual knowledge and make LLMs output desired content such as "Eiffel Tower is located in the city of Rome". Meng et al. [23] utilizes Gaussian noise to perturb predictions to achieve mass editing in a transformer.

As a comparison, PFedEdit is different from prior model editing works in two aspects. First, PFedEdit is a layer-wise model editing approach, which replaces the entire model layer from the local model instead of editing specific neuron stats. The advantage is that PFedEdit is able to perform well against potential images from various data distributions. Second, PFedEdit considers the global model as corrupted instead of one with Gaussian noise to reduce computation overhead.

3 Overview

We start by describing some backgrounds in defining personalized federated learning and then present a motivating example.

3.1 Background: Personalized Federated Learning

We describe some backgrounds of personalized federated learning. Consider the initial weights for all local models as $\{\theta_1, \theta_2, \theta_3, ..., \theta_n\}$. Let $\mathcal{P}_i$ be the data distribution over domain $\mathcal{X} \times \mathcal{Y}$ and l_i be the loss function for client $i \in \{1, 2, 3.., n\}$.

The goal of personalized federated learning is to find the set of personalized weights Θ for each client such that it minimizes empirical loss with respect to each $\mathcal{P}_i$ as shown in Equation 1.

$$\Theta = argmin_{\theta_i} \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{x,y \sim \mathcal{P}_i}[l_i(\theta_i, x, y)] \tag{1}$$

3.2 A Motivating Example

In this part, we run through an example to motivate PFedEdit. We start with a challenging question. Note that previous personalized FL work [2,4,18,36] has noted the advantages of keeping some layers locally under non-i.i.d. scenarios, e.g., all batch normalization layers for FedBN [18]. This observation raises an intriguing question: How do we choose personalized layers to improve FL's performance with heterogeneous data?

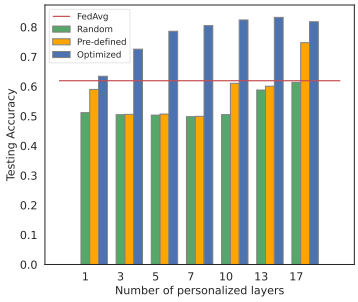


Fig. 2: A motivating example: We consider three different ways to personalize layers of local models in a federated learning scenario: random choosing, fixed choosing, and optimized layer-choosing based on the loss value. For the fixed choosing algorithm, we allow clients to keep certain numbers of batch normalization layers locally each round, and this approach becomes equivalent to FedBN [18] when the number of personalized layers gets 17.

Our motivating example provides insight into this question. Specifically, we consider a real-world federated learning scenario where there are 10 clients collaboratively training their models on the CIFAR-100 dataset [12]. Each client is assigned 20 classes of image samples following Yuan et al. [36] to form a non-i.i.d. scenario, and we choose ResNet18 [9] as their classifier. To inspect the correlation between the personalized layers and the model utility, we pick $\{1, 3, 5, 7, 10, 13, 17\}$ as the number of personalized layers, and choose three personalized algorithms for comparison. The first algorithm randomly chooses certain numbers of layers from local model and keeps those layers locally. The second algorithm follows FedBN [18] to personalize batch normalization layers, and we denoted it as fixed choosing. The third personalized algorithm optimizes layer-choosing based on loss values and it considers layers that, once kept locally, yield the least loss values personalized layer and keeps them locally each round.

Figure 2 shows the results: The optimized-personalized algorithm clearly outperforms both fixed and random personalized algorithms in all cases. This motivates our approach of using model editing to choose the optimal layers for personalization. Note that the random-personalized algorithm fails to outperform FedAvg [20] in all cases, implying that random-personalized layers do not grant utility improvement as it comes with significant randomness. The fixed-personalized algorithm fails to outperform FedAvg [20] when with a small number

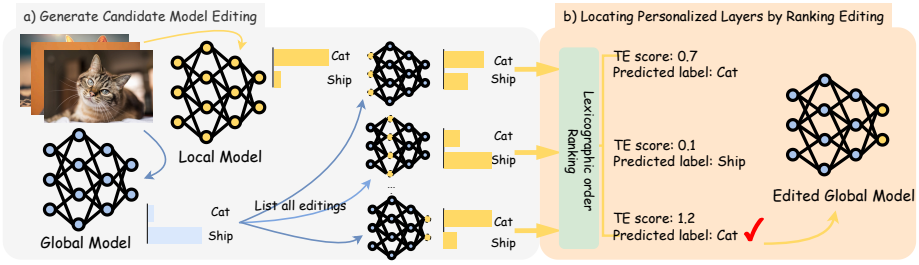


Fig. 3: Framework of PFedEdit. The federated learning still works as usual. That is, first, clients train their local models and upload their model weights to the global server. Then, the global server aggregates all model weights and returns the new global model weight to clients. Then, the main change is how PFedEdit merges global model with the local, which has two steps: i) PFedEdit generates a list of candidate model editings, and ii) PFedEdit locates the top-k important model layers and outputs the edited model for the next-round local training.

of personalized layers, and its utility is quite similar to the random-choosing algorithm. Once the number of personalized layer gets 17, which means all batch normalization layers are locally personalized, the utility improvement increases significantly and it outperforms FedAvg [18] by 13%, which implies that fixed-personalized algorithm may require all certain layers to be involved to show any utility improvement.

4 Method

In this section, we introduce our method, PFedEdit, which leverages model editing to improve model accuracy for heterogeneous data distributions. Figure 3 shows the entire workflow of PFedEdit. The training still follows a traditional FL, i.e., each client uploads their local model to the server for aggregation, and then the server returns the aggregated model. Then, when the client receives the global model, there are mainly two steps in PFedEdit, model editing and locating personalized layers, to update the client model based on the global model.

4.1 Step I: Generating Candidate Model Editings

The purpose of this step is to generate a list of possible editings of the local model based on the global model returned from the FL server. Specifically, PFedEdit considers the local model "clean model" as it is trained on local distribution. On the opposite, it considers the global model "corrupted model" as in non-i.i.d. scenarios, different models bias global model weights towards their local distributions, causing it not generalizing well on each local distribution. In the model editing step, as Figure 3 shows, PFedEdit hypothesizes all editings that tailor the "corrupted model" towards local distribution.

We show the pseudocode of this step in Algorithm 1, Line 5-6. For each editing, PFedEdit replaces one layer $\{h_t^g | t \in (1, T)\}$, where T is the total number of model layers, from the global model f_{θ^g} with the corresponding layer h_t^l from the local model f_{θ^l}. Then the edited model is defined as $S(\theta^*) = (S(\theta^g) \backslash \{h_j^g\}_{j=0}^k) \cup \{h_j^l\}_{j=0}^k$. As the global model is equipped with the local model layer, which is optimized on local distribution, the global model is tailored towards local distribution.

The first editing happens after the first round (Line 1-2), right after each client receive the aggregated global model weights (Line 4) and before they start next-round local training (Line 14). The reason is that the global model starts to bias towards other distributions starting from the second round as its weights is uniformly initialized in the first round (Line 1).

Algorithm 1 PFedEdit

Input: client i's initialized global model θ^g, communication round R, number of model layers T , number of layers to be edited k, local training set D_{train}, subset D_p, $z = (x, y) \in D_p$, model layer set $S(\cdot)$, dictionary W
Output: Edited global model weights θ^* for R-th round
1: $\theta^l \leftarrow$ train θ^g on D_{train}
2: $\theta^g \leftarrow \frac{1}{n} \sum^n \theta_i^l$
3: **for** r in range(1,$R+1$) **do**
4: **for** h_t^g in $S(\theta^g)$ **do** ▷ model editing
5: $S(\theta^*) \leftarrow (S(\theta^g) \backslash h_t^g) \cup h_t^l$
6: $g(\theta_i^l, \theta_i^*, z), (f_{\theta^*}(x), y) \leftarrow f_{\theta^*}(z)$
7: $\varphi \leftarrow (g(\theta_i^l, \theta_i^*, z), (f_{\theta^*}(x), y))$ ▷ make up all cases for prediction list φ
8: $W[h_t^g] \leftarrow \varphi$
9: **end for**
10: $W \leftarrow Lexicographic\_rank(W)$
11: **for** $j \in range(0, k)$ **do**
12: $m \leftarrow W_j.key()$ ▷ get model layer index
13: $S(\theta^*) \leftarrow (S(\theta^g) \backslash h_m^g) \cup h_m^l$
14: **end for** ▷ update local weights
15: **if** r == R **then**
16: **return** θ^*
17: **else**
18: $\theta^l \leftarrow$ train θ^* on D_{train}
19: $\theta^g \leftarrow \frac{1}{n} \sum^n \theta^l$
20: **end if**
21: **end for**

4.2 Step II: Locating Personalized Layers by Ranking Editings

The purpose of this step is to access all possible editings from Step I and then locate a list of personalized layers. The detailed substeps are two-fold. First, PFedEdit evaluates model editing's effectiveness based on a Total Effectiveness (TE) Score (Step II-a). Second, PFedEdit generates prediction lists based on TE score and ground-truth mask to reflect changes in probability and predicted labels (Step II-b). Lastly, PFedEdit ranks all prediction lists following the lexicographic order and then selects the top-k best layer editings as personalized layers for each client based on the ranking (Step II-c).

Step II-a: Evaluating model editing effectiveness. This substep is to calculate a so-called Total Effectiveness (TE) score to evaluate mode editing effectiveness. Specifically, PFEDEDIT queries local model $f(\theta^l)$ using a representative subset of the training set, denoted as D_p, which maintains the same distribution as the full training set D_{train} and $|D_p| = p * |D_{train}|$, where $p \in (0, 1)$. PFEDEDIT considers that model weights θ^l represents local distribution as it is trained on pure local data. PFEDEDIT then queries each edited model $f(\theta^*)$ using the same subset to see the prediction of tailored weights on local distribution. Then by comparing probability outputs from both local and edited weights on the same image samples, PFEDEDIT measures the effectiveness of that editing as well as the capability of edited layer in improving model accuracy. Specifically, when querying the edited model $f(\theta^*)$, PFEDEDIT collects both TE scores and ground-truth masks, Line 7 in Algorithm 1. The Total Effectiveness (TE) score, as defined below, reflects the probability change on the ground-truth class while the ground truth mask, which is a boolean list indicating whether the predicted label matches the ground truth label, reflects the predicted label change.

Now we describe the definition of TE score. Given model weight θ and training dataset D with c classes, we denote by $P(y|\theta, x)$ the probability (softmax) of the ground-truth class y. We define the total effectiveness (TE) score of editing the model weight θ into the new weight θ^* given dataset D is defined as:

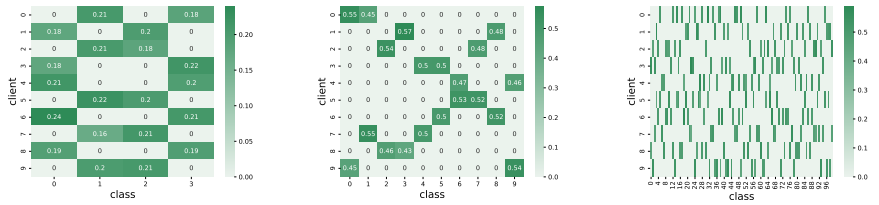
$$g(\theta, \theta^*, D) = \frac{1}{N} \sum_{x,y \in D} \frac{P(y|\theta^*, x)}{P(y|\theta, x)} - 1 \tag{2}$$

A positive TE score represents an increased probability on ground-truth label while a negative TE score signifies a decrease.

Step II-b: Generating prediction lists based on TE and ground-truth mask. This sub-task is to generate prediction lists based on TE and ground-truth mask. Specifically, the TE score, together with the ground-truth mask, makes up four combinations in total, and we call the list of all four cases a prediction list, denoted by φ. The prediction list then serves as a reference for evaluating the editing as well as the capability of such layer in improving model's accuracy. Below we show the prediction list that contains all four combinations of TE score and ground-truth mask in Equation 3.

$$\varphi = \begin{cases} f_{\theta^*}(x) = y, g(\theta, \theta^*, z) > 0 \\ f_{\theta^*}(x) = y, g(\theta, \theta^*, z) < 0 \\ f_{\theta^*}(x) \neq y, g(\theta, \theta^*, z) > 0 \\ f_{\theta^*}(x) \neq y, g(\theta, \theta^*, z) < 0 \end{cases} \tag{3}$$

The top case of the prediction list φ shows the the best editing (predicted label same as ground-truth label, TE score greater than 0) and bottom case shows the worst editing effect (predicted label different to ground-truth label, TE score less than 0). PFEDEDIT collects values for such prediction list for each editing by splitting which case each image prediction belongs to, and then ranks all prediction lists using Lexicographic order defined below.



(a) Lyme **(b)** CIFAR-10 **(c)** CIFAR-100

Fig. 4: Non-i.i.d. data distribution (2 classes per client and 20 classes for CIFAR-100) among clients in Section 5.2.

Step II-c: Locating top-k personalized layers using lexicographic order. This sub-step is to rank all prediction lists (φ) using lexicographic order and select the top-k as the personalized layers. Here is the definition of lexicographic order, which is used to rank φ. That is, given two prediction lists φ_1, φ_2 for two model editings on layer l_1, l_2, respectively, say $\varphi_1 = a_1, ..., a_4$, $\varphi_2 = b_1, ..., b_4$, where a and b are possible cases from prediction lists, the order of those two lists depends on the numeric order of the cases in the first place i where the two case counts differ. That is, $\varphi_1 < \varphi_2$ if and only if $a_i < b_i$ in the natural order of numbers.

Note that the advantages of using such metric here are two-folds: i) We record $\varphi_j(\theta, \theta^*, z)$ by counting the number of each case and compare any two prediction lists following the Lexicographic order instead of a summation manner, i.e., $\sum_{|D|}^{i} \varphi_j(\theta, \theta^*, z)$, so that the impact of one case is not overridden by another, even if there are insufficient instances. ii) The case order listed in Equition 3, from top to the bottom, is intuitive and aligns with accuracy metric, thus implying that the editing with highest prediction list ranking comes with a largest accuracy improvement. We also discuss and compare our metrics with other potential ones, and evaluation details are shown in Section 5.4.

5 Evaluation

In this section, we compare PFedEdit with baselines in terms of the accuracy and convergence. We then perform a study on parameters used in PFedEdit. The evaluation on computation overhead and complexity of PFedEdit can be found in Appendix A.4.

5.1 Experimental Setup

We describe four aspects, i.e., setup of federated learning, datasets, models, and then baselines, for our experimental setup.

FL Setup. Our default settings follow state-of-the-art works [2, 6] in evaluating federated learning. That is, there are 10 clients to participate in a FL scenario with 100 communication rounds. The global sampling rate is set to be 1.0 in each communication round and each client has one local training epoch. The default

value (k) in selecting and editing top layers is 7%, and the default representative subset ratio (p) is 0.1. Both values are selected based on our parameter study (Section 5.4). By default, personalized layers are chosen each round, and the comparison with fixed personalized layer setting can be found in Appendix A.2.
Datasets. We evaluate PFedEdit on three benchmark datasets to show its performance on different domains. Specifically, we choose i). CIFAR-10 [12], a benchmark dataset that contains 60,000 32×32 color images in 10 classes; ii). CIFAR-100 [12], a benchmark dataset that contains 60,000 32×32 color images in 100 classes; and iii). Lyme [35] dataset, a skin disease dataset that contains 3,027 skin disease images in 4 classes that are confusers for Lyme. We resize CIFAR-10 image samples to 64×64, CIFAR-100 samples to 224×224 following [36], and Lyme samples to 255×255 following [35].

To show that PFedEdit can perform on various non-i.i.d. scenario, we adopt the Distributional Heterogeneity level (DH-level) from Yuan et al. [36] to split data distribution into different levels. Specifically, we first assign certain numbers of classes to each client i, and then for each local class c, we sample $D_{i,c} \in (0.4, 0.6)$ from D_c. Each client is assigned $\frac{D_{i,c}}{\sum D_{n,c}}$ of samples in class c. The DH-level ranges from 0%, which represents an i.i.d. scenario where each client gets all classes of data, all the way to 100%, which means that each client only gets one class of data. (10 classes for CIFAR-100). For Lyme [35] dataset, since it only possesses four classes, we only conduct evaluations on 50% (2 classes per clients) and 100% (4 classes per clients) DH-level distribution scenarios. Unless specified otherwise, the default data distribution among clients is 2 classes per client (20 classes per client for CIFAR-100) in our evaluation. We demonstrate the default heterogeneous data distribution for each dataset in Figure 4. We also consider feature space non-i.i.d. scenario, and the evaluation, in comparison with Cd2-pfed [27] and Partialfed-fix [29], can be found in Appendix A.3.
Models. For model selection, we choose ResNet18 [9] for evaluation on CIFAR 10 and CIFAR 100, and ViT [8] for Lyme. We allow ResNet18 model to have pretrained weights and allow ViT to have b16 weights that is fine-tuned on Image1k. We set the learning rate for ViT to be 1e-3 and we follow FedPer [2] to set the learning rate as 1e-2 for ResNet18.
Baselines. We compare PFedEdit with FedAvg [20], MOON [14], and other five SOTA personalized FL works as baselines, namely, FedPer [2], FedRep [4], FedBN [18], APFL [6], and DisTrans [36]. We allow each baseline to have the same data distribution and model selection as PFedEdit. We consider all layer normalization layers personalized for FedBN [18] when employing ViT. For other parameters, we follow their default paper settings to assign values. The accuracy comparison with more baselines (Fedfa [38], FedALA [37],Fed-LAMB [10], FedRolex [1], and HeteroFL [7]) can be found in Appendix A.1.

5.2 Accuracy Comparison

As Table 1 shows, PFedEdit outperforms other state-of-the-art works on all three datasets under different DH-level data distributions. Compared to the

baseline FedAvg, the accuracy improvement ranges from 0.007 to 0.266, with an average 0.09 accuracy improvement, and the largest accuracy improvement gets to 0.156 and average improvement gets to 0.05 when compared to other personalized FL works. The accuracy gap between PFedEdit and other STOA personalized works in CIFAR-10 is not as huge as in other two dataset, and the reason is that we allow all works to employ models with pretrained weights, which provides a greater lower bound for model accuracy and reduces the accuracy gap between different methods.

We then evaluate PFedEdit, together with other SOTA personalized FL methods under different DH levels. According to Table 1, PFedEdit consistently achieves superior accuracy compared to other SOTA works under all data distributions, and we have two observations here. First, some personalized FL methods, e.g., FedPer, FedBN, and FedRep, fail to outperform FedAvg under i.i.d. scenarios, which is attributed to their natures as personalized FL works: they are strategically designed to align the global model with local distributions under non-i.i.d. scenarios. If the data distribution is i.i.d., where there is a close alignment between the global model distribution and each local one, the personalized layers may block favorable updates from other clients. On the other hand, PFedEdit outperforms FedAvg under i.i.d. scenarios, and the reason is that PFedEdit allow clients to update the weights of personalized layers so that they can favor other clients when local distributions are similar. Second, the accuracy improvement brought by PFedEdit compared to FedAvg increases as the DH-level increases. The reason is that as the distribution becomes more non-i.i.d., the gap between the global and local model distributions widens. Personalized layers help bridge this gap, tailoring the global model towards the local distribution and improving its generalization on local data.

It is noteworthy to mention that PFedEdit gets 0.989 accuracy on CIFAR-10 when DH-level gets to 100%. The reason is that, in an extreme non-i.i.d. scenario where clients only get one class of data samples, the local model is very likely to get 1.0 testing accuracy and the model editing effectively improves the model accuracy as the local model distribution is highly matched with the local data distribution.

5.3 Convergence

In this section we evaluate the convergence rate of PFedEdit. Specifically, we compare PFedEdit with FedAvg and we demonstrate their best accuracy on each dataset and the numbers of rounds to achieve it. As Table 2 shows, PFedEdit converges much faster than FedAvg on all three datasets as it only requires 18 epoches to converge on CIFAR-100, 13 epoches on CIFAR-10, and 25 epochs on Lyme. On the opposite, FedAvg takes around three times numbers of rounds to get a suboptimal accuracy on both CIFAR-10 and CIFAR-100. For Lyme, it takes 87 epochs to get accuracy above 0.7, which only takes 4 epochs for PFedEdit. The reason is that FedAvg does not provide convergence guarantee as it aggregates all model weights each round without having any optimization,

Table 1: Evaluation of PFedEdit with other SOTA works on different datasets.

Dataset	DH-level	FedAvg [20]	FedPer [2]	FedBN [18]	FedRep [4]	DisTrans [36]	MOON [14]	APFL [6]	PFedEdit
CIFAR-10	0 % (i.i.d)	0.942	0.890	0.9272	0.889	0.947	0.922	0.891	**0.955**
	40%	0.923	0.873	0.926	0.884	0.944	0.931	0.889	**0.953**
	60%	0.939	0.880	0.940	0.920	0.960	0.927	0.910	**0.965**
	80%	0.806	0.887	0.945	0.933	0.954	0.938	0.915	**0.962**
	100%	0.830	0.902	0.961	0.980	0.972	0.940	0.927	**0.989**
CIFAR-100	0 % (i.i.d)	0.783	0.733	0.746	0.742	0.788	0.746	0.744	**0.790**
	40%	0.779	0.753	0.776	0.747	0.779	0.762	0.753	**0.825**
	60%	0.772	0.748	0.802	0.793	0.830	0.768	0.802	**0.855**
	80%	0.737	0.769	0.862	0.859	0.887	0.770	0.833	**0.890**
	100%	0.664	0.790	0.904	0.880	0.903	0.773	0.859	**0.930**
Lyme	0%	0.707	0.700	0.733	0.730	0.745	0.699	0.722	**0.765**
	50 % (i.i.d)	0.716	0.808	0.821	0.833	0.850	0.768	0.781	**0.912**

Table 2: Best Accuracy of PFedEdit/FedAvg and number of rounds to achieve it. (- represents not achieved within 100 rounds)

Dataset	PFedEdit/FedAvg	# Rounds for target accuracy (PFedEdit/FedAvg)					
		>0.6	>0.7	>0.75	>0.8	>0.85	>0.9
CIFAR-10	**0.962**/0.806	**4**/5	**6**/8	**7**/13	**9**/33	11/-	13/-
CIFAR-100	**0.890**/0.737	**6**/9	**8**/24	10/-	12/-	18/-	-/-
Lyme	**0.912**/0.716	**3**/32	**4**/87	5/-	5/-	9/-	25/-

Table 3: Accuracy of PFedEdit with different editing ratios (in comparison with FedAvg).

Dataset	FedAvg	PFedEdit				
		1%	3%	5%	7%	9%
CIFAR-10	0.806	0.933	0.947	0.948	**0.962**	0.952
CIFAR-100	0.748	0.860	0.878	0.885	**0.890**	0.874
Lyme	0.716	0.868	0.880	0.889	**0.912**	0.910

and PFedEdit edits the global model and tailors it towards the local distribution, which makes it faster to generalize on local distributions.

5.4 Parameter Study

Model Editing Ratio (k) PFedEdit allows users to pre-define k, the number of layers to be edited. It then edits the top-k layers, which are considered to be the most effective in improving model accuracy. In this section we set up an experiment to show the correlation between the number of modified layers and the model accuracy. Intuitively, if the number of edited layers increases, the model accuracy gets higher as greater part of the model is tailored towards local distribution. However, such increase may not hold all the time as there are layers that do not possess any learnable parameters and the model may converge.

We choose the ratios of edited layers to be 1%, 3%, 5%, 7%, and 9% to the total model layers and we round up the number in each case. Shown in Table 3, as the editing ratio increases, the overall testing accuracy gets higher. The accuracy improvement is greater from improving the ratio from 1% to 3% and the increasing rate gets flatter when keep improving the ratio from 3% to

5%, or 5% to 7%. At the end, when the ratio is set from 7% to 9%, the accuracy starts to drop a little bit, implying that the model may converge on that accuracy. Compared to the baseline, PFedEdit improves the model accuracy from 0.13, with editing 1% layers, to 0.16, with editing 7% layers, on three datasets on average.

Table 4: Evaluation of PFedEdit with representative subset ratio (i.e., p).

Dataset	Subser ratio				
	p=0.01	p=0.025	p=0.05	p=0.08	p=0.1
CIFAR-10	0.951	0.953	0.954	0.954	**0.962**
CIFAR-100	-	0.875	0.884	0.886	**0.890**

Table 5: Evaluation of PFedEdit using different metrics in assessing model editing

Dataset	Different metrics			
	Accuracy	Loss	TE score	Prediction list
CIFAR-10	0.540	0.905	0.885	**0.962**
CIFAR-100	0.705	0.886	0.821	**0.890**

Representative Subset Ratio (p) As mentioned in Section 4, PFedEdit queries the model using a representative subset D_p of the training set D_{train}. PFedEdit employs a ratio p to pre-define the subset size compared to the training set, $|D_p| = p * |D_{train}|$. We want to ensure that the subset is representative enough for potential testing image sample while not introducing too much computation overhead. Therefore, we conduct a study on the size of the subset versus the performance of PFedEdit. Intuitively, a larger size covers a broader local data distribution and leads to a more accurate layer locating and thus a higher model accuracy. However, on the other hand, taken a larger subset from the original training set reduces the size of the remaining set. We want to find an optimal size of the subset that is large enough to give a complete representation of local data distribution but also leaves enough samples for the local training. We split the local training set into a subset that takes up of $p * |D_{train}|$ image samples, and a training set that takes the rest $|D^*_{trian}| = |D_{train}| - |D_p|$ image samples. To control the variable, we fix the size of D^*_{trian}, and we choose different p values from $\{0.1, 0.08, 0.05, 0.025, 0.01\}$ to adjust the subset size. We conduct the evaluation on CIFAR-10 and CIFAR-100 [12], with 10 clients and each client is assigned 2 classes of image samples. Each client possesses 4,500 image samples as training set, and the subset size varies from 50 image samples (p value 0.01) to 500 image samples (p value 0.1). Note that the p value starts from 0.025 for CIFAR-100 because a subset that is 0.01 times the size of training set does not include image samples from all classes, i.e, 50 image samples do not cover 100 classes thus the subset is not representative.

As Table 4 shows, the overall trend of the testing accuracy keeps increasing as p value gets larger from 0.01 times the training set to 0.1 times the training set for both datasets. However, the overall accuracy improvement is minimal compared to the subset size increase, demonstarting the capability of PFedEdit in improving model accuracy with a small size subset. For CIFAR-10, as p value increases from 0.01 to 0.08, which is an 8 times increase, the model accuracy is improved from 0.951 to 0.954, which is a 0.3% improvement. As p value increases from 0.08 to 0.1, the accuracy is further improved from 0.954 to 0.962, which is a 0.8% increase. For CIFAR-100, the accuracy improvement brought by increasing p value is greater than that in CIFAR-10 as the model accuracy is improved from

0.875 to 0.890 when p value changes from 0.025 to 0.1. Therefore, PFedEdit is computation-friendly as it requires only 50 image samples to get a 0.951 model accuracy and 500 image samples to achieve a 0.962 accuracy on CIFAR-10. Similarly, PFedEdit requires only 125 image samples as the subset to achieve a 0.875 accuracy and 500 image samples to gets a 0.890 accuracy on CIFAR-100.

Metrics Selection In this section, we conduct experiments to explore possible metric selections for PFedEdit when assessing each model editing. Besides the prediction list that demonstrates all four combinations of TE score and ground-truth mask, we also choose model accuracy, loss value, and TE score. We select cross-entropy loss as our loss metric, which is defined as $H(P,Q) = -\sum_{x\in\mathcal{X}} p(x)logq(x)$, given two probability distribution P and Q, and the TE score is introduced in Section 4.

As Table 5 shows, the prediction list employed by PFedEdit outperforms other metrics and the accuracy gap ranges from 0.057 to 0.422. Among all possible metrics, using accuracy to assess model editings leads to the lowest accuracy, 0.540 on CIFAR-10 and 0.705 on CIFAR-100, lower than the baseline FedAvg [20]. The reason is that the effects of some editings are subtle and they only cause the probability change but not necessarily results in label change. As the way that accuracy is computed only counts on samples that are correctly classified, small probability change that do not alter the final prediction will be considered as no improvement at all. On the other hand, the TE score reflects the probability change on ground-truth label, but it does not reflects any label change. In other words, an increase in the probability of the ground-truth label suggests a positive editing, but a decrease in probability does not necessarily indicate a negative editing. From the accuracy perspective, the edited model performs identically to the original model despite these probability changes.

While employing loss functions as the metric provides insights into the probability of the ground-truth class, the averaged loss value may exhibit bias if outliers have a disproportionate impact on the probability of the ground-truth class. For example, If an editing improves the model accuracy in correctly classifying two samples with a slight increase in the ground-truth class and misclassifies one sample with a notable drop in the ground-truth class, the improvement introduced by the correct classifications might be nullified by the incorrect one, which leads to a biased averaged loss value with respect to each data sample.

6 Conclusion

In this paper, we design the first personalized federated learning framework that utilizes automated model editing as a core strategy to improve the model's utility. Our approach leverages layer-specific editing, which automatically identifies specific layers to be edited based on local data distribution, allowing for more effective adaptation of the global model to heterogeneous data distributions across various clients. Our evaluation demonstrates that PFedEdit significantly improves the accuracy of the global model from non-i.i.d. to i.i.d. data distribution.

Acknowledgments

This work was supported in part by National Science Foundation (NSF) under grants OAC-23-19742 and Johns Hopkins University Institute for Assured Autonomy (IAA) with grants 80052272 and 80052273. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF or JHU-IAA.

References

1. Alam, S., Liu, L., Yan, M., Zhang, M.: Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction (2023)
2. Arivazhagan, M.G., Aggarwal, V., Singh, A.K., Choudhary, S.: Federated learning with personalization layers. arXiv preprint arXiv:1912.00818 (2019)
3. Basu, S., Zhao, N., Morariu, V., Feizi, S., Manjunatha, V.: Localizing and editing knowledge in text-to-image generative models (2023)
4. Collins, L., Hassani, H., Mokhtari, A., Shakkottai, S.: Exploiting shared representations for personalized federated learning. In: Proceedings of the 38th International Conference on Machine Learning (2021)
5. Dai, D., Dong, L., Hao, Y., Sui, Z., Chang, B., Wei, F.: Knowledge neurons in pretrained transformers. arXiv preprint arXiv:2104.08696 (2021)
6. Deng, Y., Kamani, M.M., Mahdavi, M.: Adaptive personalized federated learning (2020)
7. Diao, E., Ding, J., Tarokh, V.: Heterofl: Computation and communication efficient federated learning for heterogeneous clients (2021)
8. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale (2021)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
10. Karimi, B., Li, P., Li, X.: Layer-wise and dimension-wise locally adaptive federated learning (2022)
11. Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S., Stich, S., Suresh, A.T.: Scaffold: Stochastic controlled averaging for federated learning. In: International conference on machine learning (2020)
12. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep. (2009)
13. Li, L., Fan, Y., Tse, M., Lin, K.Y.: A review of applications in federated learning. Computers & Industrial Engineering (2020)
14. Li, Q., He, B., Song, D.: Model-contrastive federated learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2021)
15. Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: Challenges, methods, and future directions. IEEE signal processing magazine (2020)
16. Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks. Proceedings of Machine learning and systems **2**, 429–450 (2020)

17. Li, X., Li, S., Song, S., Yang, J., Ma, J., Yu, J.: Pmet: Precise model editing in a transformer. arXiv preprint arXiv:2308.08742 (2023)
18. Li, X., Jiang, M., Zhang, X., Kamp, M., Dou, Q.: Fed{bn}: Federated learning on non-{iid} features via local batch normalization. In: International Conference on Learning Representations (2021), https://openreview.net/pdf?id=6YEQUn0QICG
19. Liu, Q., Chen, C., Qin, J., Dou, Q., Heng, P.A.: Feddg: Federated domain generalization on medical image segmentation via episodic learning in continuous frequency space. The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
20. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics (2017)
21. Meng, K., Bau, D., Andonian, A., Belinkov, Y.: Locating and editing factual associations in GPT. Advances in Neural Information Processing Systems (2022)
22. Meng, K., Bau, D., Andonian, A., Belinkov, Y.: Locating and editing factual associations in GPT. Advances in Neural Information Processing Systems **35** (2022)
23. Meng, K., Sen Sharma, A., Andonian, A., Belinkov, Y., Bau, D.: Mass editing memory in a transformer. The Eleventh International Conference on Learning Representations (ICLR) (2023)
24. Nguyen, D.C., Ding, M., Pathirana, P.N., Seneviratne, A., Li, J., Vincent Poor, H.: Federated learning for internet of things: A comprehensive survey. IEEE Communications Surveys & Tutorials (2021)
25. Nguyen, D.C., Ding, M., Pham, Q.V., Pathirana, P.N., Le, L.B., Seneviratne, A., Li, J., Niyato, D., Poor, H.V.: Federated learning meets blockchain in edge computing: Opportunities and challenges. IEEE Internet of Things Journal (2021)
26. Nori, H., Jenkins, S., Koch, P., Caruana, R.: Interpretml: A unified framework for machine learning interpretability. arXiv preprint arXiv:1909.09223 (2019)
27. Shen, Y., Zhou, Y., Yu, L.: Cd^2-pfed: Cyclic distillation-guided channel decoupling for model personalization in federated learning (2022)
28. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security (2015)
29. Sun, B., Huo, H., YANG, Y., Bai, B.: Partialfed: Cross-domain personalized federated learning via partial initialization. In: Advances in Neural Information Processing Systems (2021)
30. Wang, P., Zhang, N., Xie, X., Yao, Y., Tian, B., Wang, M., Xi, Z., Cheng, S., Liu, K., Zheng, G., et al.: Easyedit: An easy-to-use knowledge editing framework for large language models. arXiv preprint arXiv:2308.07269 (2023)
31. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST) (2019)
32. Yang, Y., Hui, B., Yuan, H., Gong, N., Cao, Y.: PrivateFL: Accurate, differentially private federated learning via personalized data transformation. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 1595–1612 (2023)
33. Yang, Y., Yuan, H., Hui, B., Gong, N., Fendley, N., Burlina, P., Cao, Y.: Fortifying federated learning against membership inference attacks via client-level input perturbation. In: 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (2023)
34. Yang, Z., Zhang, Y., Zheng, Y., Tian, X., Peng, H., Liu, T., Han, B.: Fedfed: Feature distillation against data heterogeneity in federated learning. In: Thirty-seventh Conference on Neural Information Processing Systems (2023)

35. Yuan, H., Aucott, J., Hadzic, A., Paul, W., Villegas de Flores, M., Mathew, P., Burlina, P., Cao, Y.: Edgemixup: Embarrassingly simple data alteration to improve lyme disease lesion segmentation and diagnosis fairness. In: Medical Image Computing and Computer Assisted Intervention – MICCAI 2023 (2023)
36. Yuan, H., Hui, B., Yang, Y., Burlina, P., Gong, N.Z., Cao, Y.: Addressing heterogeneity in federated learning via distributional transformation. In: Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVIII. p. 179–195 (2022)
37. Zhang, J., Hua, Y., Wang, H., Song, T., Xue, Z., Ma, R., Guan, H.: Fedala: Adaptive local aggregation for personalized federated learning. In: Proceedings of the AAAI Conference on Artificial Intelligence (2023)
38. Zhou, T., Konukoglu, E.: FedFA: Federated feature augmentation. In: The Eleventh International Conference on Learning Representations (ICLR) (2023)

A Appendix

A.1 Accuracy comparison with more baselines

We compare PFedEdit with more SOTA personalized FL works. The results show that PFedEdit outperforms *all* of them by 0.2%–24.8% in terms of accuracy.

Table 6: Performance (Accuracy) of PFedEdit vs. Baselines.

	Fedfa [38]	Fedfed [34]	FedALA [37]	Fed-LAMB [10]	FedRolex [1]	HeteroFL [7]	PFedEdit
CIFAR-10	0.887±0.060	0.923±0.058	0.936±0.023	0.917±0.024	0.792±0.032	0.735±0.030	0.938±0.043
CIFAR-100	0.673±0.032	0.696±0.023	0.840 ± 0.022	0.822±0.020	0.674±0.042	0.620±0.038	0.868 ±0.018
Lyme	0.938±0.029	0.943±0.038	0.932±0.035	0.938±0.043	0.872±0.046	0.831±0.040	0.970±0.042

A.2 Comparison on personalized layer strategies

Fixed vs. Flexible Personalized Layers. We conduct an ablation study in Table 7 to show the advantage of flexible layer personalization vs. fixed one on CIFAR-100 and CIFAR-10.

Table 7: Ablation Study. (Convergence round / Accuracy)

	Fixed	Flexible (i.e., PFedEdit's default)
CIFAR-100	30/0.885±0.019	26/0.890±0.018
CIFAR-10	25/0.920±0.025	23/0.962±0.018

A.3 More data distribution

Feature space non-i.i.d. We also demonstrate that PFedEdit performs better on feature space non-i.i.d. FL scenario. We assign light- and dark-skinned samples of the Lyme dataset to different clients as a feature Space non-i.i.d setting. Table 8 (Last Row) shows that PFedEdit outperforms both Cd2-pfed and Partialfed.

Table 8: Accuracy of PFedEdit vs. Baselines.

	Cd2-pfed [27]	Partialfed-fix [29]	PFedEdit
CIFAR-10 (non-i.i.d)	0.915±0.050	0.903±0.040	0.938±0.043
CIFAR-100 (non-i.i.d)	0.833±0.028	0.825±0.023	0.868±0.018
Lyme (non-i.i.d)	0.903±0.040	0.926±0.045	0.970±0.042
Lyme (feature non-i.i.d)	0.897±0.032	0.908±0.020	0.921±0.027

A.4 Discussion on overhead and complexity

Overall Overhead (Caused by Layer Traversal). We discuss the computation overhead brought by generating candidate model editing step introduced in Section 4.1. We argue that the overall overhead introduced by traversing all the layers is minimum because (i) the traversal only involves forward (not backward) propagation, and (ii) PFedEdit reduces the total number of communication rounds. We compared PFedEdit with other SOTA works utilizing different model structures and Table 9 shows that PFedEdit has a shorter or similar training time compared with baselines on CIFAR-10.

Table 9: Training Time (Seconds) of PFedEdit vs. Baselines.

	FedAvg	pFedALA	Fed-LAMB	PFedEdit
MLP (4 Layers)	60.50	50.27	54.31	33.74
ResNet18(54 Layers)	187.09	157.63	175.80	150.98
ResNet50 (107 Layers)	301.23	168.59	280.53	272.08
DenseNet121 (370 Layers)	642.54	581.03	606.40	768.17

Computational Complexity. Theoretically, the computational complexity of PFedEdit's training is $O(R{\cdot}E{\cdot}|D_{train}|{\cdot}C(\theta^g){+}R{\cdot}T{\cdot}C'(\theta^g){\cdot}|D_{train}|{+}R{\cdot}n{\cdot}T)$, where θ^g represents the model, $C'(\theta^g)$ and $C(\theta^g)$ the complexity of a single forward and both forward and backward pass in θ^g, T the number of layers in θ^g, R the number of communication rounds, E the number of local training epochs, n the number of clients, and D_{train} the training set. Practically, the training time of PFedEdit is shown in Table 9, which is on par with prior works.