FedIN: Federated Intermediate Layers Learning for Model Heterogeneity

Anonymous authors
Paper under double-blind review

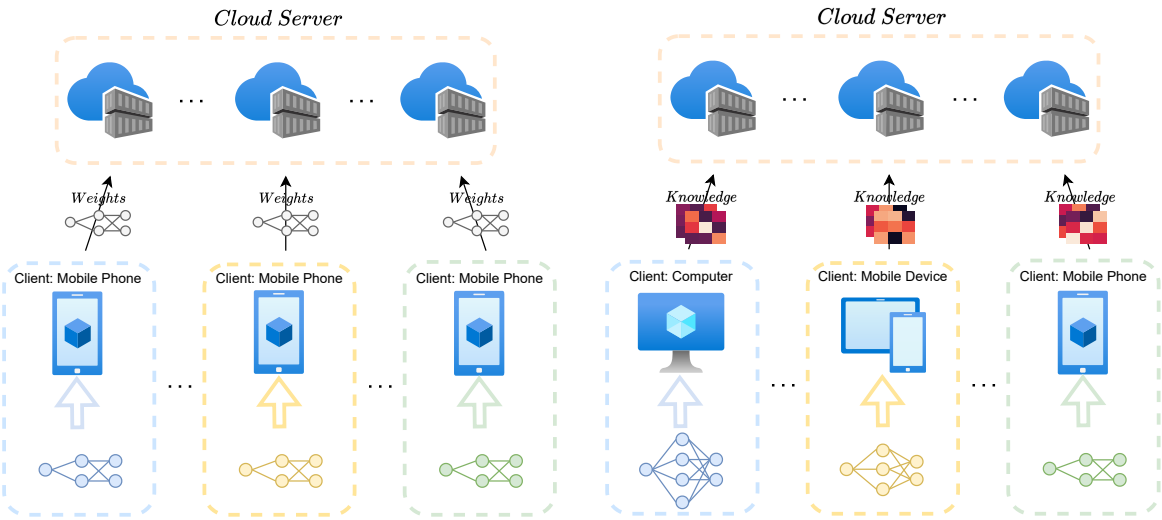
Abstract

Federated learning (FL) facilitates edge devices to cooperatively train a global shared model while maintaining the training data locally and privately. However, a prevalent yet impractical assumption in FL requires the participating edge devices to train on an identical global model architecture. Recent research endeavors to address this problem in FL using public datasets. Nevertheless, acquiring data distributions that closely match to those of participating users poses a significant challenge. In this study, we propose an FL method called Federated Intermediate Layers Learning (FedIN), which supports heterogeneous models without relying on any public datasets. Instead, FedIN leverages the inherent knowledge embedded in client model features to facilitate knowledge exchange. To harness the knowledge from client features, we propose Intermediate Layers (IN) training to align intermediate layers based on features obtained from other clients. IN training only needs minimal memory and communication overhead by employing a single batch of client features. Additionally, we formulate and resolve a convex optimization problem to mitigate the challenge of gradient divergence stemming from model heterogeneity. The experimental results demonstrate the superior performance of FedIN in heterogeneous model settings compared to state-of-the-art algorithms. Furthermore, the experiments discuss the details of how to protect user privacy leaked from IN features, and our ablation study illustrates the effectiveness of IN training.

1 Introduction

The substantial surge in Internet-of-Things (IoT) device utilization has led to the generation of vast quantities of user data (Song et al., 2022). Effectively managing this IoT big data without compromising user privacy has emerged as a significant concern. **Federated Learning** (FL) (McMahan et al., 2017) is proposed as a distributed machine learning paradigm that facilitates collaborative training on IoT data while keeping user data locally. Within FL, each client transmits model weights from their local models to the server following a few local training epochs. Subsequently, the server aggregates these weights to update the global model, and sends this model back to clients.

While Federated Learning (FL) has demonstrated success in various applications, such as recognizing human activities (Chen et al., 2019b; Ouyang et al., 2021) and learning sentiment (Smith et al., 2017; Qin et al., 2021), numerous practical challenges persist within the FL domain (Kairouz et al., 2021). One of the most crucial and practical challenges is system heterogeneity, characterized by varying resources among client devices participating in FL training (Li et al., 2020a; Chan et al., 2024). Many existing FL schemes (Li et al., 2021a; Karimireddy et al., 2020) assume that the client devices with distinct resources possess the same architecture as the global shared model for global aggregation. Nevertheless, clients with limited computation resources may struggle to complete local training in time, dragging the training speed of the entire communication round. The clients hindering the training process are called stragglers. To combat this issue, some research has proposed asynchronous FL (Xie et al., 2020; Chen et al., 2020; Chai et al., 2021), adjusting local training epochs dynamically and clustering clients according to their available resources in order to mitigate the problem of stragglers. Nevertheless, given that all clients keep the same model architecture, less capable clients may lack the sufficient memory to deploy the shared global model. In this case, the global model must be adjusted to a smaller size, leading to the resource waste of more capable



(a) System homogeneity. (b) System heterogeneity.

Figure 1: All clients have the same model architectures in system homogeneous FL as shown in Figure 1a. In system heterogeneity, the clients participate in the federated learning with different available resources, inducing different model architectures in Figure 1b.

clients and diminishing the performance of FL training. Therefore, A straightforward way to facilitate system heterogeneity is to deploy different model architectures based on the available resources of the clients, as shown in Figure 1b. However, the server can not aggregate the weights directly like Figure 1a under the heterogeneous model architectures. Recent works addressing this challenge through knowledge distillation (Hinton et al., 2015) using a public dataset, such as RHFL (Fang & Ye, 2022) and FedMD (Li & Wang, 2019). While these methods allow for diverse model architectures on clients, it is challenging to collect a suitable public dataset with a similar distribution to the local datasets.

Therefore, to support system heterogeneity without relying on a public dataset, we propose a method called Federated Intermediate Layers Learning (**FedIN**), training the intermediate layers according to a single batch of features obtained from other clients. In FedIN, a local model architecture consists of three components: an extractor, intermediate layers, and a classifier, as depicted in Figure 2. Client features are derived from the outputs of the extractor and the inputs to the classifier. Notably, clients only need to transmit **one batch** of features to the server, in addition to weight updates. The intermediate layers are updated through a combination of local training and **IN** training process, where IN training leverages a single batch of features to extract latent knowledge from other clients. However, directly deploying these two training processes can induce a critical problem called gradient divergence (Wang et al., 2020; Zhao et al., 2018), as the latent information from the local dataset and the features collected from other clients varies, particularly in a model heterogeneous environment. To alleviate the effect of this problem, we formulate and address a convex optimization problem to obtain the optimal updated gradients. Moreover, we use a simple yet efficient method, adding Gaussian noise to the client features to protect user privacy. The experiment results reveal that FedIN outperforms the baselines in terms of both accuracy and overhead.

Our contributions are summarized as follows.

- We proposed a novel FL method called **FedIN**, utilizing local training and IN training for intermediate layers, which is a flexible and reliable FL method addressing the system heterogeneity problem.
- To alleviate the effects of the gradient divergence, we formulate a convex optimization problem to derive the optimal updated gradient. The ablation study shows its effectiveness in handling the gradient divergence problem.
- To protect user privacy within FedIN, we utilize Gaussian noise in the IN training process. The experiments demonstrate the effectiveness of this approach in ensuring user privacy.
- Our experiments reveal that FedIN achieves the best performances in the IID and non-IID data compared with the state-of-the-art algorithms. Moreover, we conduct a thorough analysis to investigate the factors contributing to the improvements attained by FedIN.

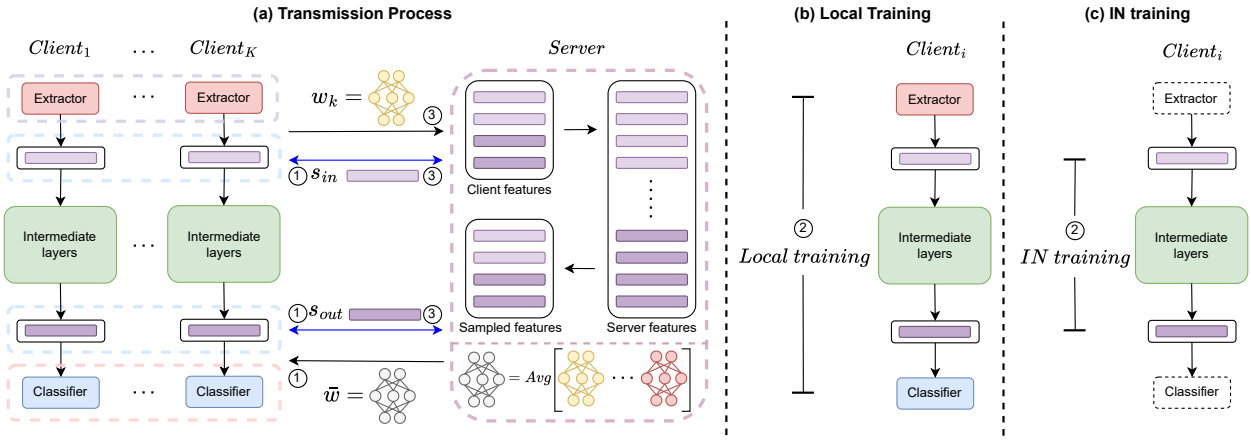


Figure 2: **Details of (a) the transmission process, (b) local training and (c) IN training process for FedIN.** The process for FedIN is described as follows. ① First, clients receive client features and global weights $\bar{w}$ from the server. ② After updating client weights by global weights, the clients complete the local training from the local private dataset and complete the IN training for the client features inputs and outputs (s_{in}, s_{out}) from the server. ③ Upon completing the training process, clients transmit the model weights and new client features, denoted as (w_k, s_{in}, s_{out}), to the server. The aggregation methods for system heterogeneity are discussed in section 4.4.

2 Related Work

2.1 Federated Learning

Federated Learning (FL) was proposed in 2017 to organize cooperative model training among edge devices and servers (McMahan et al., 2017). In FL, numerous clients train models jointly while retaining training data locally to maintain privacy protection. Various methods have been proposed and achieved good performance in different scenarios. In (Xie et al., 2020), FedAsyn utilizes coordinators and schedulers to create an asynchronous training process, handling the stragglers in the FL training process. FedProx (Li et al., 2020b) regularizes and re-parametrizes FedAvg, guaranteeing convergence when learning over non-IID data. To share local knowledge among clients with different model architectures, FCCL (Huang et al., 2022) generates a cross-correlation matrix based on the unlabeled public dataset.

2.2 Heterogeneous Models

Our work focuses on supporting heterogeneous models in FL. This subsection classifies recent research contributing to model heterogeneity into three categories.

Public and Auxiliary Data. If a server has a public dataset, clients can exploit the general knowledge from this dataset, constructing a simple and efficient bridge to exchange knowledge among clients. FedAUX (Sattler et al., 2021) utilizes unsupervised pre-training and unlabeled auxiliary data to initialize heterogeneous models. FedGen (Zhu et al., 2021) simulates the prior knowledge from all the clients according to a generator. To dig out the latent knowledge from the public dataset, several studies (Li & Wang, 2019; Li et al., 2021b; He et al., 2020) propose addressing the system heterogeneity problem, inspired by knowledge distillation (Hinton et al., 2015). In FedMD (Li & Wang, 2019), a large public dataset is deployed in a server, while the clients distill and transmit logits from this dataset to learn the knowledge from both logits and local private datasets. In FedH2L (Li et al., 2021b), clients extract the logits from a public dataset consisting of small portions of local datasets from other clients. In RHFL (Fang & Ye, 2022), a server calculates the weights of clients by the symmetric cross-entropy loss function, and clients distilled knowledge from the unlabeled dataset. FCCL (Huang et al., 2022) computed a cross-correlation matrix also based on the unlabeled public dataset. MocoSFL (Li et al., 2023) proposes a mechanism, replay memory on features to assist the MoCo functions (Chen et al., 2021), a contrastive framework, in model heterogeneous FL.

Data-free Knowledge Distillation. The basic ideas of data-free KD are to optimize noise inputs to minimize the distance to prior knowledge (Nayak et al., 2019), and Chen et al. (Chen et al., 2019a) train Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) to generate training data for the entire KD process, utilizing the knowledge distilled from the teacher model. To free the limitation from a public dataset, some research works consider data-free KD in FL. In FedML (Shen et al., 2020), latent knowledge from homogeneous models is applied to train heterogeneous models. In FedHe (Chan & Ngai, 2021), logits belonging to the same class are directly averaged in a server. In FedGKT (He et al., 2020), a neural network is split into a client and a server, while the server completes the entire training process based on the features and logits collected from all clients. FedMK (Liu et al., 2023) utilizes dataset distillation to transmit latent knowledge between clients in FL.

Splitting Models. To adapt to the available resources of different clients, several studies split the large models into small sub-models. HeteroFL (Diao et al., 2021) divides a large model into local models with different sizes. However, the architectures of local and global models are still restricted by the same model architecture. SlimFL (Baek et al., 2022) integrates slimmable neural network (SNN) architectures (Yu & Huang, 2019) into FL, adapting the widths of local neural networks based on resource limitations. In (Horvath et al., 2021), FjORD leverages Ordered Dropout and a self-distillation method to determine the model widths. ScaleFL (Ilhan et al., 2023) splits a server model along two dimensions, and local models are trained using the cross-entropy and KL-divergence loss functions. InCo (Chan et al., 2024) proposes three splitting methods with convex optimization problems to solve the gradient divergence problem in heterogeneous FL.

3 Problem Formulation

The goal of FL is to collaborate with the clients to train a shared global model while keeping their local data private. We briefly summarize the optimization problem below. We assume that K clients participate in FL. Each client has a private dataset $D_k = \{(x_{i,k}, y_{i,k}) | i = 1, 2, ..., |D_k|\}$, where $k \in \{1, ..., K\}$ is the index of a client, and $|D_k|$ denotes the size of a dataset D_k. Private dataset D_k is only accessible to client k, guaranteeing data privacy. In traditional FL, the clients share identical model architecture. We denote a training model by $f(x; w)$, where w are the training weights and x are the inputs. The loss function l_k of client k is shown as follows,

$$\min_{w} \quad l_k(w) = \frac{1}{|D_k|} \sum_{i=1}^{|D_k|} l(f(x_{i,k}; w), y_{i,k}), \tag{1}$$

where $l(\cdot, \cdot)$ is a loss function for each data sample $(x_{i,k}, y_{i,k})$. Nevertheless, it may not be possible to deploy an identical model architecture for all the clients due to system heterogeneity. One potential solution is to allow clients to select different model architectures according to their capabilities in heterogeneous FL. The problem of heterogeneous FL is described as follows. We denote w_k as the model weights of client k. If the total size of all datasets is $N = \sum_{k=1}^{K} |D_k|$, the global optimization function is described as follows,

$$\min_{w_1, w_2, ..., w_K} \quad L(w_1, ..., w_K) = \sum_{k=1}^{K} \frac{|D_k|}{N} l_k(w_k), \tag{2}$$

where the optimized model weights $\{w_1, w_2, ..., w_K\}$ have different sizes. Thus, the direct aggregation of entire model weights becomes unfeasible when dealing with heterogeneity among models. Therefore, we adopt layer-wise heterogeneous aggregation (Liu et al., 2022; Chan et al., 2024) as an alternative approach to aggregate the layer weights of heterogeneous models instead of the entire model weights in our experiments.

4 FedIN: Federated Intermediate Layers Learning

In this section, we describe the details of FedIN, focusing on addressing system heterogeneity by deploying clients with diverse model architectures that align with their available resources. Figure 2 illustrates the workflow of FedIN. The client model consists of three key components: an extractor, intermediate layers, and a classifier. The outputs of the extractor, referred to as feature inputs (s_{in}), serve as inputs to the

intermediate layers. Similarly, the outputs of the intermediate layers, referred to as feature outputs (s_{out}), act as inputs to the classifier. The client features are the pair of feature inputs and outputs, denoted as (s_{in}, s_{out}). To be specific, FedIN encompasses two training processes: local training, which leverages the private dataset, and IN training, which relies on the feature inputs and outputs (s_{in}, s_{out}). Moreover, to address the challenge of gradient divergence arising from conflicts from model heterogeneity, we propose a convex optimization problem formulation to obtain the optimal updated gradients.

4.1 Local Training and IN Training

The clients receive a single batch of feature inputs and feature outputs, denoted as $S = \{(s^c_{i,in}, s^c_{i,out})|i = 1, 2, ..., |S|\}$, from the server. These samples are utilized for training the intermediate layers during the IN training process. The superscript c means that these feature inputs and outputs are from the central server. The clients begin their local training after receiving a batch of client features from the server. For an instance $(x_{i,k}, y_{i,k}) \in D_k$, client k conducts local training on its private dataset. The loss function of the local training is shown as follows,

$$l_{local,k} = \quad l_{CE}(f(x_{i,k}; w^t_k), y_{i,k}) + \frac{\mu}{2}||w^t_k - w^{t-1}_k||^2, \tag{3}$$

where w^t_k are the weights of client k at time t, and l_{CE} is the cross-entropy loss function for the local training. To ensure client consistency, we add a proximal regularization term (Li et al., 2020b) in Eq. 3.

The second training process is IN training, which is training the intermediate layers from the features dataset S. It is worth mentioning that the sample number of S is one batch size. We denote the weights of the extractor and the classifier by $w_{e,k}$ and $w_{c,k}$ for client $k \in \{1, ..., K\}$. Moreover, the weights of the intermediate layers are denoted by $w_{in,k}$. The relations among the data sample $(x_{i,k}, y_{i,k}) \in D_k$, client weights, and $(s^k_{i,in}, s^k_{i,out})$ are shown as follows,

$$s^k_{i,in} = f(x_{i,k}; w_{e,k}), \tag{4}$$

$$s^k_{i,out} = f(s^k_{i,in}; w_{in,k}), \tag{5}$$

$$f(x_{i,k}; w_k) = f(s^k_{i,out}; w_{c,k}). \tag{6}$$

Eq. 4 shows that the feature input $s^k_{i,in}$, the light purple components in Figure 2, is the output of the extractor $w_{e,k}$ of an instance $(x_{i,k}, y_{i,k})$ from client k. Eq. 5 describes that the feature output $s^k_{i,out}$, the deep purple components in Figure 2, is the output of the intermediate layers $w_{in,k}$ with the feature input $s^k_{i,in}$. Eq. 6 proves the equivalence between the output of the classifier $w_{c,k}$ and the output of the whole client model w_k. Eq. 5 shows the main function of the IN training, as shown in Figure 2(c). After the client receives the feature dataset $S = \{(s^c_{i,in}, s^c_{i,out})|i = 1, 2, ..., |S|\}$, it begins the IN training for the intermediate layers. The feature inputs $s^c_{i,in}$ from the server are the inputs of the intermediate layers, while the $s^c_{i,out}$ are the targets of the IN training. The loss function of IN training is defined as follows,

$$l_{IN,k} = l_{MSE}(f(s^c_{i,in}; w_{in,k}), s^c_{i,out}), \tag{7}$$

where l_{MSE} is a mean-square error loss function. The weights $w_{in,k}$ are updated by the loss functions of the local training $l_{local,k}$ and the IN training $l_{IN,k}$. We use MSE as the loss function due to its effectiveness in this learning method. Moreover, and s_{in} and s_{out} do not represent probability distributions, making it difficult to incorporate other losses such as KL-divergence and cross-entropy losses.

4.2 Gradient Alleviation

However, local training is based on the local data, while IN training is based on the features from other clients' data. Different local datasets lead to varied distributions, resulting in dissimilar optimized directions. Moreover, in our scenario, deploying distinct model architectures in clients emphasizes differences in feature spaces, as shown in Figure 3. These combined factors result in divergent gradients between local training and IN training, impeding the pace of convergence and disturbing the model to achieve the optimum point (Wang et al., 2020; Zhao et al., 2018). Therefore, mitigating this gradient divergence is imperative for the effectiveness of our method. To address this problem, inspired by (Chan et al., 2024), we formulate a convex optimization problem as follows.

We define the gradients from the local training as a matrix G_{local} and the gradients from the IN training as a matrix G_{IN}, both for the intermediate layers. To guarantee the optimized direction of the models, we design a constraint for the gradient as follows,

$$\langle G_{IN}, G_{local} \rangle \geq 0, \tag{8}$$

where $\langle \cdot, \cdot \rangle$ is the dot product, which ensures the optimized direction for G_{local} and G_{IN} to be the same. In the optimization problem, we denote the new optimized gradients by a matrix Z and model the following convex optimization primal problem,

$$\min_{Z} ||G_{IN} - Z||_F^2, \quad s.t. \ \langle Z, G_{local} \rangle \geq 0, \tag{9}$$

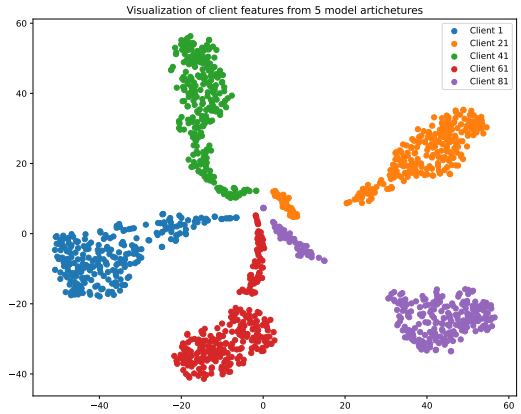


Figure 3: T-SNE visualization depicts IN feature outputs s_{out} derived from five distinct model architectures, with each color representing a unique model architecture.

where we maintain the optimized direction between Z and G_{local} to be the same and minimize the distance between Z and G_{in}. We consider that the information from the feature inputs and outputs is more fruitful than the local private dataset which is easier to have over-fitting in the training process. We solve this convex optimization problem by the Lagrange dual problem (Bot et al., 2009; Chan et al., 2024). The Lagrangian is shown as,

$$\begin{aligned} L(Z, \lambda) = tr(G_{IN}^T G_{IN}) - tr(Z^T G_{IN}) \\ -tr(G_{IN}^T Z) + tr(Z^T Z) - \lambda tr(G_{local}^T Z), \end{aligned} \tag{10}$$

where $tr(A)$ means the trace of the matrix A, and the λ is a Lagrange multiplier associated with $\langle Z, G_{local} \rangle \geq 0$. To derive the dual problem, we first get the optimum of Z for the Lagrangian Eq. 10, and then obtain the Lagrange dual function $g(\lambda) = \inf_Z L(Z, \lambda)$. Thus, the Lagrange dual problem is described as follows,

$$\max_{\lambda} \ g(\lambda) = -\frac{\lambda^2}{4} tr(G_{local}^T G_{local}) - \lambda tr(G_{local}^T G_{IN}), \quad s.t. \ \lambda \geq 0, \tag{11}$$

where the optimum of the Lagrangian Eq. 10 is $Z = G_{IN} + \frac{\lambda}{2} G_{local}$. If the λ is large enough, it is obvious that $\langle Z, G_{local} \rangle > 0$, which means this convex optimization problem holds strong duality because it satisfies the Slater's constraint qualification(Boyd et al., 2004), i.e., the optimum of the primal problem Eq. 9 is also $Z = G_{IN} + \frac{\lambda}{2} G_{local}$. Furthermore, the dual problem Eq. 11 can be solved to obtain the analytic solution for λ and Z, which is shown as follows,

$$Z = \begin{cases} G_{IN}, & \text{if } b \geq 0 \\ G_{IN} - \frac{b}{a} G_{local}, & \text{if } b < 0 \end{cases} \tag{12}$$

where $a = tr(G_{local}^T G_{local})$ and $b = tr(G_{local}^T G_{IN})$. However, one crucial point is that the clients will handle this optimization process. If we calculate each gradient matrix following Eq. 12, this process would occupy lots of computing resources because of the matrix multiplication. Therefore, to mitigate the computational pressure on the clients, we simplified the updated gradient matrix as,

$$Z = G_{IN} + \frac{\lambda}{2} G_{local}, \tag{13}$$

where $\lambda = 1$ is set for the optimum point of the primal problem in our experiment settings. The detailed derivation process for this section is shown in Appendix A.

4.3 Privacy Consideration

In our methods, clients are required to transmit feature inputs and outputs to the server, raising privacy concerns regarding the potential leakage of private data through transmitted features. We investigate two recent related attack methods, the Gradient Inversion Attack and the Model Inversion Attack. The Gradient Inversion Attack relies on the strong assumption that the server knows the private statistic of BatchNorm (Huang et al., 2021), which is not appropriate to FedIN as such information is unnecessary to transmit to the server. Additionally, the Model Inversion Attack poses a greater risk of stealing private information in our scenario, but one strong assumption for this attack is that the server needs to have prior knowledge of the client input images (Li et al., 2022), which is impractical in our scenario as the server does not receive any images from the clients. However, as the server accesses the model parameters and the IN feature inputs and outputs, we explore an alternative method known as dataset distillation (Wang et al., 2018; Lei & Tao, 2023) to potentially reconstruct the private images from the clients. We randomly initialize and train a batch of noise $\hat{x}$ with the same size as the input images x, aiming to optimize $\hat{x}$ following the following reconstruction objective: $l_{rec} = l_{MSE}(f(\hat{x}; w_e), s_{in})$, where w_e represents the freezing weights of the extractor on the server and s_{in} denotes the feature inputs.

Figure 4: The comparison between privacy with protection and without protection.

To enhance user privacy within FedIN, the clients can easily **add Gaussian noise** followed the standard deviation σ of IN feature inputs and outputs in training. Specifically, we define σ_{in} as the standard deviation of IN feature inputs and σ_{out} for feature outputs. The Gaussian noises are represented as $z_{in} \sim \mathcal{N}(0, \sigma_{in})$ and $z_{out} \sim \mathcal{N}(0, \sigma_{out})$. For simplicity in notations, we use σ to denote z, as we solely adjust σ within this privacy protection mechanism. Throughout the training phase, We apply 0.8σ to the IN feature inputs and outputs, i.e., the inputs of Eq. 5 are $\hat{s}^k_{i,in} = s^k_{i,in} + 0.8\sigma_{in}$, and for Eq. 6, they become $\hat{s}^k_{i,out} = s^k_{i,out} + 0.8\sigma_{out}$. The results of the privacy protection are shown in Figure 4, indicating the efficiency of this mechanism in protecting user privacy in FedIN. More details on the privacy experiments are further provided in Section 5.5.

4.4 Weight Aggregation

If client models have different numbers of layers, FedIN adopts layer-wise heterogeneous aggregation (Liu et al., 2022; Chan et al., 2024), enabling the server to aggregate weights from the same layer rather than the same model. Similarly, when client models have different architectures, FedIN aggregates model weights only from models with identical architectures, the same as the homogeneous aggregation method used in FedAvg (McMahan et al., 2017) and FedDF (Lin et al., 2020). The effectiveness of FedIN with these two distinctive aggregation methods is further demonstrated in Section 5.2, and the detailed demonstrations for these two aggregation methods are shown in Appendix B.

4.5 Limitation

FedIN can handle different model architectures with various numbers of layers. However, one limitation is that FedIN is unable to directly manage the models with different widths, leading to different sizes of s_{in} and s_{out}, and also the models from different family, such as CNNs and Transformers. We have two ways to alleviate these environments in FedIN. One method is aggregating model weights only from models with identical shapes, which we have conducted experiments in Section 5.2. The other method involves utilizing linear layers to project the dimension of s_{in} and s_{out} to the same size, enabling the deployment of IN training after these projections. We will investigate this method further in our future work.

Table 1: Model accuracy for IID and non-IID data of FashionMNIST, SVHN, and CIFAR-10, with target accuracy established at 85, 80, and 60, respectively. "Round" denotes the round at which the method first achieves the target accuracy in Non-IID. "Speed" refers to the convergent speed compared to the speed of FedAvg. We **bold** the best results, and underline the second best results in this table.

Methods	FashionMNIST (ACC=85)				SVHN (ACC=80)				CIFAR-10 (ACC=60)			
	IID	Non-IID	Round↓	Speed↑	IID	Non-IID	Round↓	Speed↑	IID	Non-IID	Round↓	Speed↑
FedAvg(2017)	90.3	89.2±1.1	47	×1.0	89.2	84.3±1.2	82	×1.0	76.8	65.7±1.8	109	×1.0
FedProx(2020b)	89.7	87.8±0.5	40	×1.2	90.6	87.1±1.0	45	×1.8	77.6	72.5±1.2	72	×1.5
Scaffold(2020)	88.3	87.1±0.9	25	×1.9	91.1	86.0±1.5	72	×1.1	79.0	67.9±1.7	120	×0.9
FedNova(2020)	87.5	87.3±1.4	36	×1.3	87.3	86.5±1.7	106	×0.8	62.9	61.4±2.5	229	×0.5
MOON(2021a)	89.5	89.0±1.0	34	×1.4	89.5	86.1±1.3	55	×1.5	74.1	67.4±1.8	129	×0.8
HeteroFL(2021)	89.3	89.5±0.6	140	×0.3	**93.8**	89.3±0.6	107	×0.8	72.1	62.5±1.3	273	×0.4
FedGen(2021)	89.1	88.7±1.2	39	×1.2	91.7	89.2±1.4	64	×1.3	78.3	69.6±2.2	134	×0.8
FedET(2022)	90.2	89.5±1.5	29	×1.6	89.8	88.6±1.6	70	×1.2	75.4	71.1±2.6	117	×0.9
InclusiveFL(2022)	88.4	89.1±1.0	31	×1.5	90.9	88.7±0.9	67	×1.2	75.0	67.2±1.7	160	×0.7
FedRolex(2022)	90.9	88.7±1.3	100	×0.4	91.3	87.0±1.2	81	×1.0	79.8	68.0±1.6	165	×0.6
ScaleFL(2023)	91.1	90.1±0.7	95	×0.5	93.7	90.2±0.7	100	×0.8	76.4	72.0±2.0	108	×1.0
InCoAvg(2024)	90.6	89.5±1.2	22	×2.1	90	87.4±1.8	55	×1.5	78.7	67.5±2.6	127	×0.8
FedIN	91.2	90.2±1.2	20	×2.4	91.8	89.4±1.3	29	×2.8	80.5	74.8±2.3	54	×2.0
FedIN (+Noise)	**91.3**	**90.6±1.3**	**18**	**×2.6**	92.9	**91.0±1.5**	**26**	**×3.1**	**83.2**	**77.3±2.5**	**52**	**×2.1**

5 Experiments

5.1 Experiment Settings

Federated Settings. In this section, we conduct experiments to evaluate the performances of FedIN on the CIFAR-10 (Krizhevsky et al., 2009), Fashion-MNIST (Xiao et al., 2017), and SVHN (Netzer et al., 2011) datasets. We establish two distributions for these datasets, independent and identically distributed (IID), and non-IID. The non-IID data is generated using a Dirichlet distribution with a parameter $\alpha = 0.5$. We have 100 clients in the FL training process. The model architectures are ResNet10, ResNet14, ResNet18, ResNet22, and ResNet26 from PyTorch source codes, and they are evenly distributed among 100 clients. We use one CNN layer at the beginning as an extractor, and one MLP layer at the end for a classifier. The number of communication rounds is set to 500. The batch size is 16 during the training process. For all datasets, the clients complete five epochs of local training during each communication round. Our code will be released on Github.

Baselines. We have 12 baselines, including two classic algorithms, FedAvg (McMahan et al., 2017) and FedProx (Li et al., 2020b), and ten state-of-the-art methods, Scaffold (Karimireddy et al., 2020), FedNova (Wang et al., 2020), MOON (Li et al., 2021a), HeteroFL (Diao et al., 2021), FedGen (Zhu et al., 2021), FedET (Cho et al., 2022), InclusiveFL (Liu et al., 2022), FedRolex (Alam et al., 2022), ScaleFL (Ilhan et al., 2023), and InCo (Chan et al., 2024). FedIN and these baselines, FedAvg, FedProx, Scaffold, FedNova, and MOON, utilize the layer-wise aggregation technique proposed in (Chan et al., 2024; Liu et al., 2022) under our heterogeneous model environment. FedIN (+Noise) is a privacy-protected version of FedIN. More discussions on user privacy are provided in Section 5.5. We first focus on FedIN without adding noises in the experiments. Since HeteroFL, FedRolex, and ScaleFL require model splitting based on their own methodology, they cannot utilize this aggregation technique. To maintain a similar number of parameters as the other baselines, we deploy ResNet152 in these baselines instead of using the largest model, ResNet26, as in other methods. The model split mode in these baselines is "dynamic_a1-b1-c1-d1-e1" from the source code because of five heterogeneous models in all other methods. The hyper-parameter $\frac{\mu}{2}$ for FedProx and FedIN is 0.05. We use Adam optimizer with default parameter settings in PyTorch for all methods. All experiments are conducted with one Nvidia RTX3090 GPU. We show average results from three random seeds in Non-IID experiments.

5.2 Accuracy Analyses.

Accuracy of IID and non-IID Data. We conduct experiments on the IID and non-IID data in Fashion-MNIST, SVHN, and CIFAR-10 datasets. The experiment results are shown in Table 1. From Table 1, FedIN

Table 2: Training overheads for different methods. "Params" indicates the communication overheads, ("M" means million). "Memory" refers to the memory occupied by methods in the training process.

Metrics	Methods					
	FedAvg	Scaffold	MOON	HeteroFL	FedRolex	FedIN
Params(M) ↓	12.28	24.56	12.28	16.29	16.29	12.35
Memory(MB) ↓	235.0	470.0	705.0	445.6	445.6	235.3

Table 3: Model accuracy with heterogeneity models with FedAvg aggregations.

Fashion-MNIST	Methods						
	FedAvg	FedProx	Scaffold	FedNova	MOON	InclusiveFL	FedIN
IID	86.1	83.4	87.7	84.2	87.0	88.1	**88.9**
Non-IID	85.4	82.1	86.3	83.9	86.5	86.4	**88.0**

(+Noise) achieves the highest accuracy among all methods in FashionMNIST, CIFAR-10, and gets the second highest accuracy in SVHN with IID. More discussions for FedIN (+Noise) are shown in Section 5.5. FedIN also gets the second-best results among different datasets. These results demonstrate the effectiveness of FedIN. Additionally, Figure 5 shows the smoothed test accuracy on non-IID data of CIFAR-10. FedIN (red line) achieves the highest accuracy and exhibits the fastest convergent speed throughout the training process. It is the first method to achieve the target accuracy (red dot line). Moreover, FedIN incurs only a small additional overhead of one batch of feature inputs and outputs compared to FedAvg, as shown in Table 2.

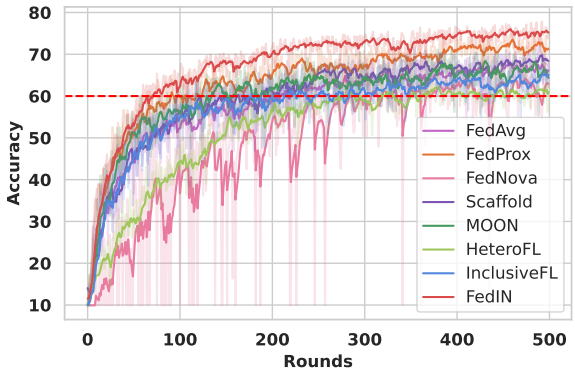


Figure 5: The smoothed test accuracy on non-IID data of CIFAR-10. The red dot line denotes the target accuracy in Table 1.

Accuracy of Homogeneous Models. While FedIN primarily addresses the system heterogeneity challenge in FL, we also conduct experiments in a homogeneous model environment using CIFAR-10. All client models are ResNet18 in this experiment, and the remaining federated settings are the same as those in the system heterogeneity experiments. As presented in Table 4a, FedIN still outperforms state-of-the-art baselines, specifically for the baselines that are designed to enhance FL performance in homogeneous model environments.

Accuracy with FedAvg Aggregation. It is worth noting that FedIN can be deployed in scenarios with extreme heterogeneity, where layer-wise aggregation is not feasible. In such cases, model weights can only be aggregated with models that share the same shape. Therefore, when the server aggregates weights from a specific model shape, it disregards weights from models with different shapes. This setup simulates a scenario where clients have completely different model architectures, such as CNNs and Transformers, and aggregation can only occur among clients with identical shapes. To demonstrate the effectiveness of FedIN in such extreme environments, we conducted experiments on the Fashion-MNIST dataset, utilizing FedAvg aggregation. The remaining federated settings are the same in this experiment. As indicated in Table 3, FedIN still achieves the highest accuracy, 88.9% on IID data and 88.0% on non-IID data. These results further emphasize the effectiveness of FedIN in extreme system heterogeneity environments.

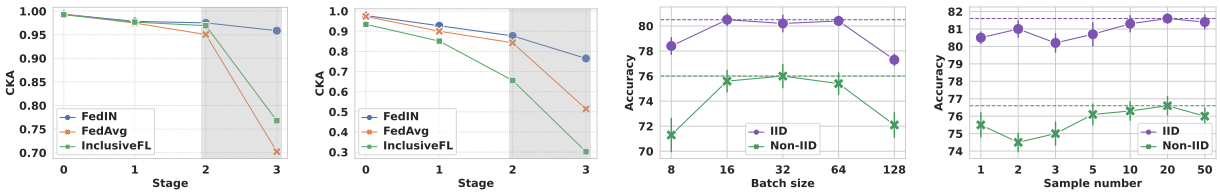
Table 4: Model accuracy with different settings in CIFAR-10.

(a) Model accuracy with homogeneous models.

CIFAR-10	Methods				
	FedProx	Scaffold	FedNova	MOON	FedIN
IID	83.5	84.3	82.0	84.2	**84.7**
Non-IID	77.5	76.8	75.4	78.2	**79.2**

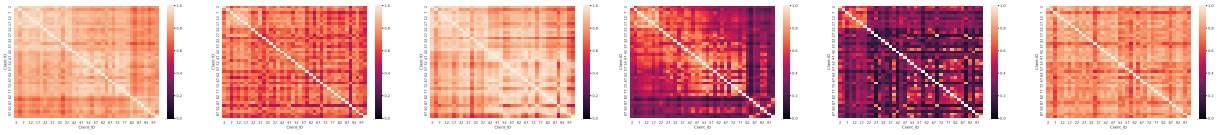
(b) Model accuracy with ablation studies.

CIFAR-10	Methods				
	FedAvg	w/o IN	w/o Prox	w/o Opt	FedIN
IID	76.8	77.6	78.8	79.4	**80.5**
Non-IID	66.2	72.0	66.4	74.9	**75.9**



(a) CKA similarity for IID data. (b) CKA similarity for non-IID data. (c) Effects from different batch sizes. (d) Effects from different sample numbers.

Figure 6: Illustrations for CKA similarity of IID data in Figure 6a and non-IID data in Figure 6b with CIFAR-10. The effects from different batch sizes and different sample numbers are shown in Figure 6c and Figure 6d under non-IID CIFAR-10.



(a) Stage 2 of FedAvg. (b) Stage 2 of InclusiveFL. (c) Stage 2 of FedIN. (d) Stage 3 of FedAvg. (e) Stage 3 of InclusiveFL. (f) Stage 3 of FedIN.

Figure 7: Heatmaps of CKA similarity from stage 2 and 3 among different clients in CIFAR-10.

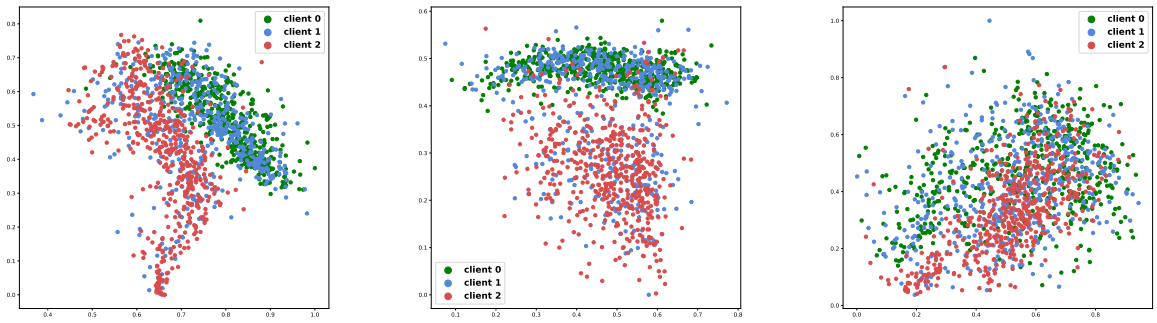
5.3 The Reason for the Improvements

CKA Similarity for Different Stages. Inspired by (Luo et al., 2021) and (Raghu et al., 2021), we use CKA similarity (Kornblith et al., 2019) to examine the layer similarity among different clients across different methods. In our analysis, stage i indicates the i_{th} block in the ResNet architecture. We define all residual blocks with the same shape as one stage. Figure 6a and Figure 6b illustrates the CKA similarity of different stages under IID and non-IID. Notably, in these results, FedIN exhibits the highest similarity even in the deepest stage (stage 3), while FedAvg and InclusiveFL struggle to maintain high similarity levels in stage 3, as evidenced by the gray area in the figure. To gain further insights into the dissimilarities between FedIN and the other methods, we present heatmaps of similarity from stage 2 and stage 3 among clients in Figure 7, indicating that the average similarity of FedIN surpasses that of FedAvg and InclusiveFL. These results and analyses suggest that FedIN ensures consistency among the deep layers of client models, indicating that client models learn more general knowledge in FedIN compared to other baselines.

T-SNE Visualization. We conduct t-SNE visualizations (Van der Maaten & Hinton, 2008) on features extracted from stage 3 in Figure 8, focusing on data belonging to the same class. The objective is to observe the clustering behavior of these data points. In Figure 8a and Figure 8b, it is evident that the features from client 0 and client 1 and features from client 2 are separated. However, the features from these three clients form a singular cluster in FedIN, as depicted in Figure 8c, validating that the features from data with the same class from different model architectures are consistent.

5.4 Ablation Study

We conduct an ablation study to evaluate the contributions of the key components in FedIN. Our ablation study includes the following methods: (i) FedAvg, (ii) FedIN w/o IN (FedIN without IN loss), (iii) FedIN w/o



(a) FedAvg. (b) InclusiveFL. (c) FedIN.

Figure 8: T-SNE visualization of features learned by different methods from stage 3 on CIFAR-10. We select data from the same class and utilize three models with different architectures (Client0: ResNet10, Client1: ResNet14, Client2: ResNet26).

Table 5: Model accuracy with different client numbers on CIFAR-10.

Methods	IID					Non-IID				
	$N_c = 10$	$N_c = 20$	$N_c = 50$	$N_c = 100$	$N_c = 200$	$N_c = 10$	$N_c = 20$	$N_c = 50$	$N_c = 100$	$N_c = 200$
FedAvg	79.3	79.2	78.7	76.8	74.0	68.3	67.9	66.9	66.2	62.5
InclusiveFL	77.5	76.7	79.1	75.0	73.4	66.8	68.4	67.1	66.1	61.2
FedIN	**82.8**	**83.1**	**81.0**	**80.5**	**74.3**	**76.7**	**76.3**	**74.1**	**75.9**	**72.2**

Prox (FedIN without Prox regularized term), (iv) FedIN w/o Opt (FedIN without the gradient alleviation (optimization)). Table 4b and Figure 9 illustrates the results of the ablation studies.

Effects of the Gradient Alleviation and the Loss Function. In this experiment, we highlight that our solution is advantageous and effective in solving the gradient divergence problem. Figure 9 illustrates the results of considering the gradient divergence problem and ignoring this problem. The accuracy achieved by FedIN surpasses that of FedIN w/o Opt, and the convergent speed of FedIN is also accelerated, as observed in Figure 9. Moreover, after 200 rounds, FedIN w/o Prox becomes unstable and its performance deteriorates during the subsequent training process. At last, FedIN w/o Prox only achieves the performance like FedAvg, as shown in Table 4b, hinting that the improvement from IN loss is eliminated at the end of the training process. Therefore, the inclusion of a regularized term becomes essential to maintain the effectiveness of IN loss throughout the training process.

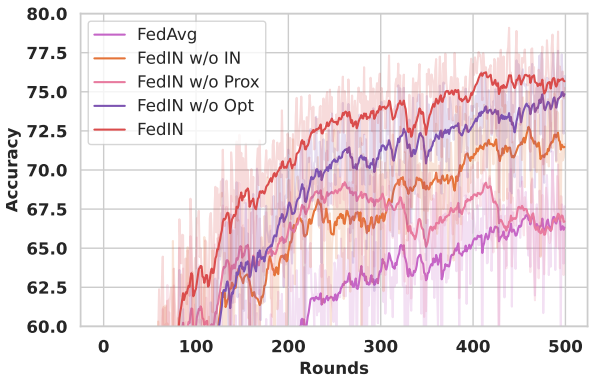
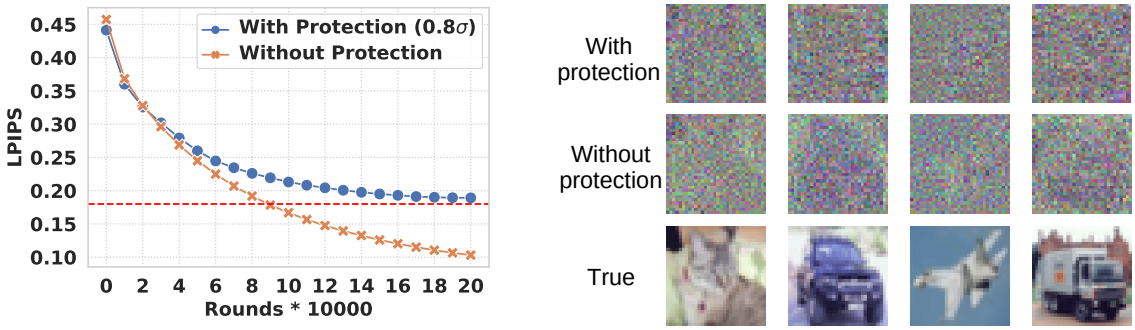


Figure 9: Smoothed test accuracy for non-IID data of CIFAR-10 in the ablation study.

Effects of Client Numbers, Batch Sizes, and Sample Numbers. To investigate the effects of varying client numbers, we conduct experiments on CIFAR-10, as presented in Table 5. N_c denotes the number of clients. Notably, FedIN outperforms the other methods across different numbers of clients. We also conduct analysis on different batch sizes and sample numbers on CIFAR-10 to verify the effects of these hyperparameters. As shown in Figure 6c, batch sizes 16, 32, and 64 are the best selections, but the batch sizes of 8 and 128 still outperform HeteroFL and InclusiveFL. Considering the communication overhead, a batch size of 16 is the optimal choice. From Figure 6d, it is clear that increasing the sample numbers has little impact on accuracy improvement.

Table 6: Model accuracy with adding noise in FedIN to protect privacy.

Dataset	Noise Level (+FedIN)								
	w/o Noise	$+0.1\sigma$	$+0.2\sigma$	$+0.5\sigma$	$+0.8\sigma$	$+1.0\sigma$	$+2.0\sigma$	$+3.0\sigma$	$+5.0\sigma$
SVHN	89.3	90.1	90.2	90.0	**91.0**	89.2	83.7	79.8	65.3
Fashion-MNIST	90.3	89.8	90.4	90.6	**90.6**	90.4	88.7	84.9	69.4
CIFAR-10	75.9	75.2	76.4	77.2	**77.3**	76.2	73.5	70.0	30.6



(a) Similarities of reconstruction images.

(b) Reconstruction images and ground truths.

Figure 10: The results from the reconstruction experiments.Figure 10a evaluates the quality of reconstruction images, using LPIPS (Balle et al., 2022), which lower value indicates more clear images. Figure 10b illustrates the reconstructed images from protected IN inputs and original IN inputs.

5.5 Privacy Analysis

We assume that the server reconstructs user images from IN inputs, originating from the outputs of the first layer, as images are more easily reconstructed from features extracted from shallow layers. The experimental results presented in Table 6 and Figure 10 provide a comprehensive analysis of the impact of how privacy-preserving mechanisms, adding Gaussian noise, ensure user privacy. We use LPIPS (Balle et al., 2022) to represent the quality of reconstruction images.

Accuracy with Different Noise Levels. Table 6 illustrates the impact of adding noise at varying levels in FedIN on the model accuracy. Moderate noise levels, especially at $+0.8\sigma$ (the setting of FedIN (+Noise)), obtain superior performance compared to FedIN without adding noise. However, as noise levels exceeding $+1.0\sigma$, there is a noticeable decline in accuracy across all datasets, indicating a degradation in model performances due to excessive noises. These results suggest that appropriate noise levels can protect user privacy while also aiding in model generalization. The features with adding appropriate noises enrich the feature spaces, implying more diversified s_{in} and s_{out} for IN training.

Reconstruction Images. In Figure 10, we add 0.8σ to IN inputs for privacy protection. The LPIPS of reconstruction images from models trained with privacy protection stabilizes at a higher value (0.18) compared to those trained without protection, as shown in Figure 10a. Figure 10b displays the reconstructed images at the server using these two methods. These results indicate that the server encounters challenges in reconstructing user images when IN inputs are protected.

6 Conclusions

We propose a method called FedIN, which conducts local training based on the private dataset and IN training from the client features, requiring only one batch of features. Moreover, we formulate a convex optimization problem to tackle the gradient divergence problem. To protect user privacy, we further propose a simple yet effective method, adding Gaussian noise during the IN training process. We conduct extensive experiments on three public datasets with ten baselines to demonstrate the superior performances of FedIN.

References

Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=OtxyysUdBE.

Hankyul Baek, Won Joon Yun, Yunseok Kwak, Soyi Jung, Mingyue Ji, Mehdi Bennis, Jihong Park, and Joongheon Kim. Joint superposition coding and training for federated learning over multi-width neural networks. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pp. 1729–1738. IEEE, 2022.

Borja Balle, Giovanni Cherubin, and Jamie Hayes. Reconstructing training data with informed adversaries. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1138–1156. IEEE, 2022.

Radu Ioan Bot, Sorin-Mihai Grad, and Gert Wanka. *Duality in vector optimization*. Springer Science & Business Media, 2009.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. Fedat: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421.

Yun Hin Chan and Edith Ngai. Fedhe: Heterogeneous models and communication-efficient federated learning. *IEEE International Confer- ence on Mobility, Sensing and Networking (MSN 2021)*, 2021.

Yun-Hin Chan, Rui Zhou, Running Zhao, Zhihan JIANG, and Edith C. H. Ngai. Internal cross-layer gradients for extending homogeneity to heterogeneity in federated learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Cc0qk6r4Nd.

Hanting Chen, Yunhe Wang, Chang Xu, Zhaohui Yang, Chuanjian Liu, Boxin Shi, Chunjing Xu, Chao Xu, and Qi Tian. Data-free learning of student networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3514–3522, 2019a.

Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9640–9649, 2021.

Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE transactions on neural networks and learning systems*, 31(10):4229–4238, 2019b.

Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pp. 15–24. IEEE, 2020.

Yae Jee Cho, Andre Manoel, Gauri Joshi, Robert Sim, and Dimitrios Dimitriadis. Heterogeneous ensemble knowledge transfer for training large models in federated learning. In Lud De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 2881–2887. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/399. URL https://doi.org/10.24963/ijcai.2022/399. Main Track.

Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*, 2021.

Xiuwen Fang and Mang Ye. Robust federated learning with noisy and heterogeneous clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10072–10081, 2022.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. *Advances in Neural Information Processing Systems*, 33:14068–14080, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning and Representation Learning Workshop*, 2015.

Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34:12876–12889, 2021.

Wenke Huang, Mang Ye, and Bo Du. Learn from others and be yourself in heterogeneous federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10143–10153, 2022.

Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. Evaluating gradient inversion attacks and defenses in federated learning. *Advances in neural information processing systems*, 34:7232–7241, 2021.

Fatih Ilhan, Gong Su, and Ling Liu. Scalefl: Resource-adaptive federated learning with heterogeneous clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24532–24541, 2023.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. 2021.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pp. 5132–5143. PMLR, 2020.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pp. 3519–3529. PMLR, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Shiye Lei and Dacheng Tao. A comprehensive survey of dataset distillation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Daliang Li and Junpu Wang. FedMD: Heterogenous federated learning via model distillation. *NeurIPS Workshop on Federated Learning for Data Privacy and Confidentiality*, 2019.

Jingtao Li, Adnan Siraj Rakin, Xing Chen, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Ressfl: A resistance transfer framework for defending model inversion attack in split federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10194–10202, 2022.

Jingtao Li, Lingjuan Lyu, Daisuke Iso, Chaitali Chakrabarti, and Michael Spranger. MocoSFL: enabling cross-client collaborative self-supervised learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=2QGJXyMNoPz`.

Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10713–10722, 2021a.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020a.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of the 3rd MLSys Conference*, 2020b.

Yiying Li, Wei Zhou, Huaimin Wang, Haibo Mi, and Timothy M Hospedales. FedH2L: Federated learning with model and statistical heterogeneity. *arXiv preprint arXiv:2101.11296*, 2021b.

Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, 33:2351–2363, 2020.

Ping Liu, Xin Yu, and Joey Tianyi Zhou. Meta knowledge condensation for federated learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=TDf-XFAwc79`.

Ruixuan Liu, Fangzhao Wu, Chuhan Wu, Yanlin Wang, Lingjuan Lyu, Hong Chen, and Xing Xie. No one left behind: Inclusive federated learning over heterogeneous devices. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3398–3406, 2022.

Mi Luo, Fei Chen, Dapeng Hu, Yifan Zhang, Jian Liang, and Jiashi Feng. No fear of heterogeneity: Classifier calibration for federated learning with non-iid data. *Advances in Neural Information Processing Systems*, 34:5972–5984, 2021.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Gaurav Kumar Nayak, Konda Reddy Mopuri, Vaisakh Shaj, Venkatesh Babu Radhakrishnan, and Anirban Chakraborty. Zero-shot knowledge distillation in deep networks. In *International Conference on Machine Learning*, pp. 4743–4751. PMLR, 2019.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 54–66, 2021.

Han Qin, Guimin Chen, Yuanhe Tian, and Yan Song. Improving federated learning for aspect-based sentiment analysis via topic memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3942–3954, 2021.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems*, 34:12116–12128, 2021.

Felix Sattler, Tim Korjakow, Roman Rischke, and Wojciech Samek. Fedaux: Leveraging unlabeled auxiliary data in federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Tao Shen, Jie Zhang, Xinkang Jia, Fengda Zhang, Gang Huang, Pan Zhou, Kun Kuang, Fei Wu, and Chao Wu. Federated mutual learning. *arXiv preprint arXiv:2006.16765*, 2020.

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. *31st Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Xuan Song, Haoran Zhang, Rajendra Akerkar, Huawei Huang, Song Guo, Lei Zhong, Yusheng Ji, Andreas L. Opdahl, Hemant Purohit, André Skupin, Akshay Pottathil, and Aron Culotta. Big data and emergency management: Concepts, methodologies, and applications. *IEEE Transactions on Big Data*, 8(2):397–419, 2022. doi: 10.1109/TBDATA.2020.2972871.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33: 7611–7623, 2020.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *12th Annual Workshop on Optimization for Machine Learning*, 2020.

Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1803–1811, 2019.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. Data-free knowledge distillation for heterogeneous federated learning. In *International Conference on Machine Learning*, pp. 12878–12889. PMLR, 2021.

A Derivation for Convex Optimization Problem

In Section 4.2, we propose the convex optimization problem to alleviate the gradient divergence caused by local training and IN training, which is shown as follow,

$$\min_{Z} ||G_{IN} - Z||_F^2, \quad s.t. \ \langle Z, G_{local} \rangle \geq 0, \tag{14}$$

which we solve this convex problem by the Lagrange dual problem (Bot et al., 2009; Chan et al., 2024). First, the Lagrangian of Eq. 14 is illustrated as,

$$\begin{aligned} L(Z, \lambda) =& tr((G_{IN} - Z)^T (G_{IN} - Z)) - \lambda tr(Z^T G_{local}) \\ =& tr(G_{IN}^T G_{IN} - Z^T G_{IN} - G_{IN}^T Z + Z^T Z) - \lambda tr(Z^T G_{local}) \\ =& tr(G_{IN}^T G_{IN}) - 2tr(Z^T G_{IN}) + tr(Z^T Z) - \lambda tr(G_{local}^T Z). \end{aligned} \tag{15}$$

To get the optimum point for $L(Z, \lambda)$, we set $\frac{\partial L(Z,\lambda)}{\partial Z} = 0$, and then we get

$$Z = G_{IN} + \lambda G_{local}/2, \tag{16}$$

which is the optimum point for $L(Z, \lambda)$ and also for the primal problem Eq. 14. Moreover, the dual problem is obtained by $L(\lambda) = \inf_Z L(Z, \lambda)$. We replace $Z = G_{IN} + \lambda G_{local}/2$ in $L(Z, \lambda)$ to get $L(\lambda)$ as shown below,

$$\begin{aligned} L(\lambda) =& tr(G_{IN}^T G_{IN}) - 2tr((G_{IN} + \frac{\lambda G_{local}}{2})^T G_{IN}) + tr((G_{IN} + \frac{\lambda G_{local}}{2})^T (G_{IN} + \frac{\lambda G_{local}}{2})) \\ & - \lambda tr((G_{IN} + \frac{\lambda G_{local}}{2})^T G_{local}) \\ =& tr(G_{IN}^T G_{IN}) - 2tr(G_{IN}^T G_{IN}) - \lambda tr(G_{local}^T G_{IN}) + tr(G_{IN}^T G_{IN} + \frac{\lambda G_{local}^T G_{IN}}{2} + \frac{\lambda G_{IN}^T G_{local}}{2} \\ & + \frac{\lambda^2 G_{local}^T G_{local}}{4}) - \lambda tr(G_{IN}^T G_{local}) - \frac{\lambda^2 tr(G_{local}^T G_{local})}{2} \\ =& tr(G_{IN}^T G_{IN}) - 2tr(G_{IN}^T G_{IN}) + tr(G_{IN}^T G_{IN}) - \lambda tr(G_{local}^T G_{IN}) + \lambda tr(G_{local}^T G_{IN}) \\ & + \frac{\lambda^2 tr(G_{local}^T G_{local})}{4} - \frac{\lambda^2 tr(G_{local}^T G_{local})}{2} - \lambda tr(G_{IN}^T G_{local}) \\ =& -\frac{\lambda^2}{4} tr(G_{local}^T G_{local}) - \lambda tr(G_{local}^T G_{IN}). \end{aligned} \tag{17}$$

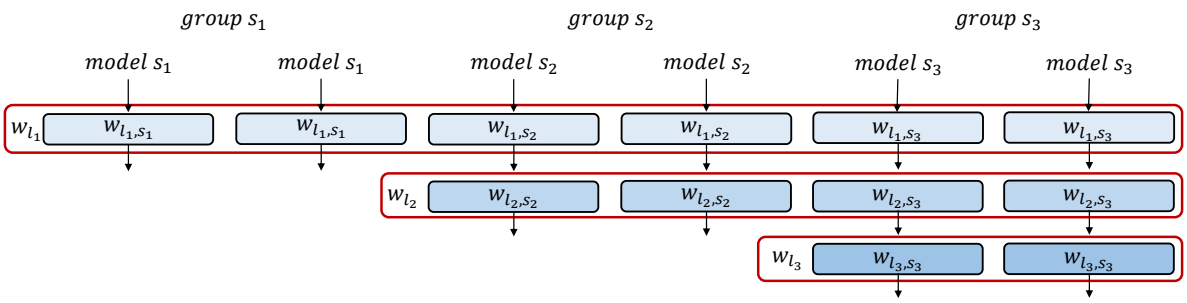
Therefore, we have the dual problem, as follows,

$$\max_{\lambda} \ L(\lambda) = -\frac{\lambda^2}{4} tr(G_{local}^T G_{local}) - \lambda tr(G_{local}^T G_{IN}), \quad s.t. \ \lambda \geq 0, \tag{18}$$

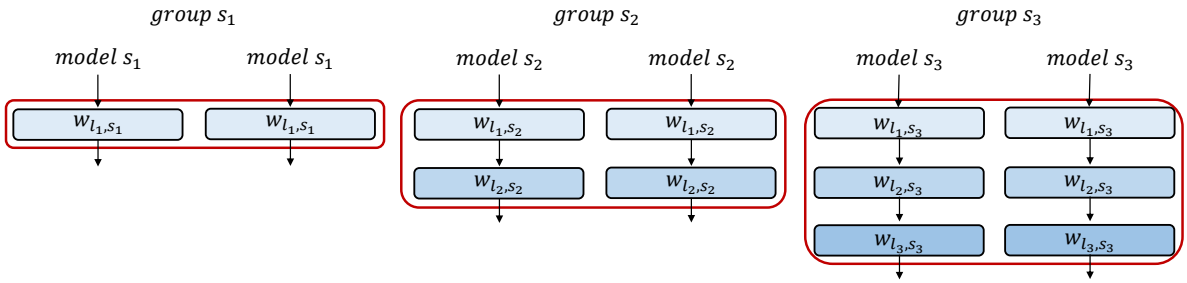
then we complete the derivation process for convex optimization problem using Lagrange dual problem.

B Details for Weight Aggregation

In Section 4.4, we present two aggregation methods for FedIN. One is layer-wise heterogeneous aggregation (Liu et al., 2022; Chan et al., 2024), and the other one is FedAvg aggregation from FedAvg (McMahan et al., 2017) and FedDF (Lin et al., 2020). In this section, we demonstrate the details for these two aggregation methods.



(a) Layer-wise heterogeneous aggregation.



(b) FedAvg aggregation with heterogeneous models.

Figure 11: Two aggregation methods for FedIN: (a) Figure 11a: Layer-wise heterogeneous aggregation. This method aggregates model weights layer by layer. (b) Figure 11b: FedAvg aggregation with heterogeneous models. This method aggregates model weights only from models with identical shapes.

B.1 Layer-wise Heterogeneous Aggregation

To facilitate heterogeneous models in FL, we utilize layer-wise aggregation. Specifically, the weights of the same layer from different models would be averaged. For example, Figure 11a illustrates the heterogeneous aggregation for n groups, $s_1, ..., s_n$. The clients in the same group share the same model architecture. w_{l_i,s_j} denotes the weights of layer i in group s_j. We further use a superscript to denote that $w^k_{l_i,s_j}$ belongs to the k-th client in group s_j. The group indexes are ordered by the size of the model architectures, i.e., the size of the client models in group s_1 is the smallest. As shown in the example in Figure 11a, all groups share layer 1, denoted by $w_{l_1,s_j}, j = 1, ..., n$. Then, layer w_{l_i} only appears in group s_j when $i \leq j$. Following the above definition, the heterogeneous aggregation for layer w_{l_i} is expressed as follows,

$$w_{l_i} = \frac{\sum_{j=i}^{n} \sum_{k=1}^{|s_j|} w^k_{l_i,s_j}}{\sum_{j=i}^{n} |s_j|}, \tag{19}$$

where $|s_j|$ denotes the number of clients in group s_j.

B.2 FedAvg Aggregation

When client models have different shapes, FedIN aggregates model weights exclusively from models with identical shapes, the same as the aggregation method employed in FedAvg, as illustrated in Figure 11b. Similarly, the environment has n different model groups, denoted as $s_1, ..., s_n$. For layer w_{l_i,s_j} in group s_j, this aggregation method is defined as follows,

$$w_{l_i,s_j} = \frac{\sum_{k=1}^{|s_j|} w^k_{l_i,s_j}}{|s_j|}. \tag{20}$$

Considering different sizes for local datasets in group s_j, and simplify the notation l_i for model weights in this methods, the aggregation method for w_{s_j} in group $s_j, j = 1, ...n$ is formulated as follows,

$$w_{s_j} = \frac{\sum_{k=1}^{|s_j|} d_k w_{s_j}^k}{\sum_{k=1}^{|s_j|} d_k}, \tag{21}$$

where d_k is the size of local datasets for client k in group s_j.