MODULAR FEDERATED CONTRASTIVE LEARNING WITH PEER NORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite recent progress in federated learning (FL), the fundamental challenge of training a global model across multiple clients having heterogeneous and class imbalanced (CIB) data has not been fully resolved. Furthermore, most of the existing works for FL with heterogeneous data assume that the clients have fully labeled data, which might not be practical in real-world scenarios due to the challenges of labeling, especially at the clients. In this paper, we provide a solution for the realistic FL setting in which the clients have unlabeled, heterogeneous, and CIB data. To address the issue of biased gradients in training on heterogeneous and CIB data, we develop a new FL framework, called the Modular Federated Contrastive Learning (MFCL). Instead of federally training a whole deep network across the clients, we propose to train two separate and different network modules at the clients and the server. One is a sensor module that is federally trained across the clients to extract the data representations from the clients' unlabeled data, which are sent to the server. The other is a discriminator module at the server, which is trained with contrastive loss on the data representations received from the clients. We also propose a new normalization technique, Peer Normalization (PN), which is customized for the contrastive FL to reduce the biases of the gradients resulting from training on the heterogeneous and CIB data across the clients. Our experiments show that the proposed MFCL with PN provides high and stable accuracy, achieving state-of-the-art performance when the clients have (severe) heterogeneous and CIB data.

1 INTRODUCTION

Federated Learning (FL) has been introduced as a framework to make it possible to learn from the data located at multiple clients without any need to bring the raw data from the clients to one central location, called the server (McMahan et al., 2016; 2017; Konečnỳ et al., 2016a;b). FL starts with broadcasting the initial model parameters to the (selected) clients. Each client's local model is trained on its own local data set for some iterations and the locally trained model parameters are sent to the server. The server averages the local models into a global model, which is broadcast back to the (selected) clients. This completes one FL round and the training process continues until convergence or achieving an accuracy threshold.

For real-world scenarios of FL, it is reasonable to assume that the clients collect their own data individually, which leads to non-identical data distributions, namely heterogeneous data sets. Besides, even for each individual client's data set, the numbers of data samples are not necessarily the same for all classes in practice. This means that each client' data might be often class-imbalanced (CIB), rather than class-balanced (CB). For these challenges, the actual training of FL might be suboptimal, might slowly converge, or even might diverge. Figure 4 in Appendix C.2 shows the four different scenarios of data distribution of clients in FL: heterogeneous/homogeneous and CIB/CB data.

Over the past several years, the FL research community has paid a lot of attention to addressing the issue of data heterogeneity of clients, mostly based on supervised learning, by applying different tricks such as regularization techniques in local objectives of the clients as in Luo et al. (2021); Dieuleveut et al. (2021); Mu et al. (2021); Li et al. (2020b; 2021); Durmus et al. (2021); Yu et al. (2020b), adaptive aggregation in the server as in Wang et al. (2020), data sharing on the clients or server as in Zhao et al. (2018); Hao et al. (2021), and data augmentation as in Shin et al. (2020) to

reduce the divergence between the local and global model. Apart from addressing the issue of data heterogeneity, some efforts have been devoted to addressing the issue of CIB data in the clients by regularization techniques as in Wang et al. (2021), data level approaches such as data augmentation in Duan et al. (2020), or data distribution estimation in Yang et al. (2021). In the literature, most of the existing FL methods including all algorithms mentioned above are based on supervised learning, which implies that all clients must have fully labeled data. However, labeling data is costly and requires expert knowledge, which is challenging especially for the end users, such as the clients in FL (Oord et al., 2018; Chen et al., 2020; He et al., 2020).

Addressing the challenges of labeling in FL, some self-supervised methods have recently been proposed considering data heterogeneous clients (Lubana et al., 2022; Lu et al., 2022; Makhija et al., 2022; Huang et al., 2022; Zhang et al., 2020b; Miao & Koyuncu, 2022; Yu et al., 2020a). In particular, contrastive learning Chen et al. (2020), which is one of the most successful self-supervised learning algorithms, has been applied to the framework of FL. In those works, contrastive learning is used for training the local models of the clients to ensure that clients can collaboratively learn from the unlabeled data (Zhang et al., 2020b; Shi et al., 2021; Miao & Koyuncu, 2022; Yu et al., 2020a).

In real-world scenarios of FL, however, it is often difficult or ineffective to apply contrastive learning directly across those multiple clients as in the existing works in the literature. First, contrastive learning relies on a large amount of data because a contrastive loss needs a large number of similar (positive) and dissimilar (negative) samples to achieve good performance. Furthermore, training with a contrastive loss typically demands large models (i.e., deep and wide neural networks) with large batch sizes, which means that high computing power and large memory footprints are needed. This might not be possible on the side of clients in FL. This motivated our work.

In this paper, based on contrastive learning, we propose a new FL method considering the practical setting in which the clients have unlabeled, heterogeneous, and CIB data. In particular, considering the restrictions of computing power and memory on the side of clients, we propose to train two separate network modules for FL, which is called the modular FL: one across the clients and the other at the server. On the side of the clients, a sensor module, which is an autoencoder, is first trained federally across the clients to extract the data representations from the clients' local unlabeled data. When this training is done, the data representations are extracted by the sensor module and then sent to the server in which a discriminator module, which is a convolutional neural network (CNN), is trained with a contrastive loss.

As another major contribution, we propose a new normalization method, namely Peer Normalization (PN). The widely adopted method of Batch Normalization (BN) fails when there exist (strong) biases of the model gradients resulting from the heterogeneous and CIB data of the clients. To address the issue, our proposed PN leverages the rich and diverse features extracted from different augmentations for each data sample, which can be considered as a normalization scheme customized for the contrastive learning in FL. Our contributions are summarized as follows:

- Considering the computing and memory limitations of the clients, we propose the modular federated contrastive learning (MFCL), which performs contrastive learning at the server using the data representations extracted by the clients. The proposed scheme is the first self-supervised learning framework in FL that trains two separate modules: one for extracting data representations at the clients and the other for discriminating the data at the server.

- We propose a new normalization technique, PN, specifically for contrastive training in FL. PN is similar to Group Normalization (GN) (Wu & He, 2018) in that channels are divided into groups, but different from GN in that PN normalizes a group of channels of *two* positive examples together, rather than one.

- Through experiments, we demonstrate the effectiveness of the proposed MFCL, especially with PN, which achieves the state-of-the-art performance. The proposed scheme provides robust and stable performance on (severe) heterogeneous and CIB data whilst only a small size sensor module is trained federally across the clients, which results in much smaller communication burden for training than the existing FL methods.

2 PROPOSED MODULAR FEDERATED CONTRASTIVE LEARNING

We aim to develop an effective FL mechanism, namely the modular federated contrastive learning (MFCL), to learn from the distributed clients having CIB and heterogeneous data, which are unlabeled. To this end, we propose to use two different types of network modules: (i) *sensor modules* at the clients, which are trained federally across clients to learn the representations of unlabelled data, and (ii) a *discriminator module* at the server, which is trained on the representation vectors received from the clients with a contrastive loss.

In the literature, there are few works on contrastive learning-based FL, and in those works, the contrastive loss was optimized at each client. However, we believe it is more practical to optimize the contrastive loss at the server. It is well-known that contrastive learning benefits more from larger model, larger batch size, and longer training time (Chen et al., 2020). Considering the memory and computing resource limitations of the clients, however, it might be difficult or, at times, even impossible to perform such contrastive learning on (some) clients (Li et al., 2014). Besides, directly optimizing the contrastive loss across clients as in the traditional FL (i.e., averaging all parameters of clients' local models trained with contrastive losses) may filter out the subtle differences between the two positive examples (i.e., two augmented versions of the same data sample), which can be important in determining the similarities of examples.

In our proposed MFCL scheme, the contrastive loss is minimized using the mini-batches of data representation (not raw data) constructed at the server, which will be referred to as the *contrastive server representation mini-batches*. In contrastive learning, each contrastive mini-batch must be composed of two augmented versions of data samples. In FL, however, the augmented versions of the data samples cannot be constructed directly at the server, because the raw data samples should be kept private to the clients. To address this issue, in our proposed scheme, we introduce and train sensor modules, which are autoencoders, as shown in Figure 1. For training of the sensor modules, the mini-batches of the augmented versions of raw input data samples are first constructed at each individual client, which are referred to as *the contrastive client input mini-batches*. The sensor modules are federally trained across the clients by minimizing a loss function such as the cross entropy or the mean squared error (MSE) between the contrastive inputs and their reconstructions produced by the decoder part of the sensor module. Once the sensor modules are trained, each client produces the representations of its own samples, called the *contrastive client representation mini-batches*, which are the outputs of the *trained* (fixed) encoder of the client's sensor module. Only the representations of input data are transmitted (disclosed) to the server, not the raw data. It is acknowledged that disclosing the representations is still not secure enough in terms of data privacy. In practice, therefore, additional security measures should be used, such as homomorphic encryption Zhang et al. (2020a), differential privacy Wei et al. (2020), secure multi-party FL Mugunthan et al. (2019), etc. We note that additional security measures are also required anyway for most of the existing FL methods (not only for our proposed method) because disclosing the model parameters or the gradients is not secure either (Li et al., 2020a; Mothukuri et al., 2021; Bagdasaryan et al., 2020).

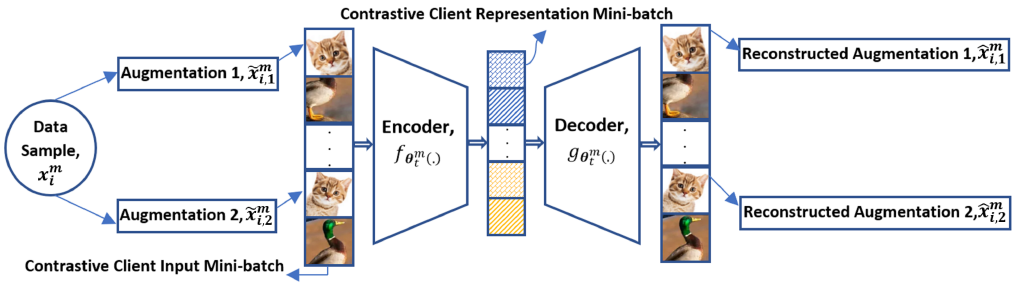


Figure 1: The sensor module at each client.

In training of the sensor modules (the autoencoders) across the clients, it is essential to mitigate the adverse effects of heterogeneous and CIB data of the clients. To address this issue, we propose a new and effective normalization scheme, namely PN, for the autoencoders. With PN, the contrastive client mini-batches do not have to be very large; besides, they are less biased toward the classes

with more samples. Unlike each client's contrastive input/representation mini-batches, the server's contrastive representation mini-batches can be constructed to be much larger, as needed, because a larger amount of data (i.e., the representations of augmented data) can be collected from multiple clients, and much more memory and computing resources are typically available at the server. At the server, therefore, we can benefit from BN (Ioffe & Szegedy, 2015).

2.1 TRAINING THE PROPOSED MFCL

The proposed MFCL is trained in two phases as follows.

Training Phase 1: At the first stage of training, the sensor modules are collaboratively trained in the FL system composed of a single server and M clients, where client m has a local data set, $\mathcal{D}^m$. As the sensor module, although any feature extractors can be potentially used, in this paper, we use convolutional autoencoders, each of which is composed of encoder $f_{\boldsymbol{\theta}_t^m}(\cdot)$ and decoder $g_{\boldsymbol{\theta}_t^m}(\cdot)$, where $\boldsymbol{\theta}_t^m$ denotes the set of parameters of the autoencoder at client m at FL round t. For normalizing the layers' outputs of the autoencoder, one could simply use the widely adopted BN if the clients' data *were* homogeneous and CB; unfortunately, however, this is not the case in many practical FL scenarios. To mitigate the effects of heterogeneous and CIB data, in the next subsection, we will design a new normalization technique, PN, considering the special structure of the contrastive mini-batch at the clients. The PN layer is inserted right after each convolutional layer (Conv2D) and each transposed convolution layer (Conv2DTranspose), except for the last layer of the decoder. The architecture of the autoencoder is described in Table 5 in Appendix C.3.

To train the autoencoder, client m first creates two different augmented versions $\tilde{\boldsymbol{x}}_{i,j}^m, j = 1, 2$ of each data sample $\boldsymbol{x}_i^m \in \mathcal{D}^m$, where $\boldsymbol{x}_i^m$ denotes the ith data sample for client m. Combining the augmented versions of K data samples, client m constructs contrastive client input mini-batches of size $2K$ denoted by $\mathcal{B}^m = \left(\tilde{\boldsymbol{x}}_{1,1}^m, \ldots, \tilde{\boldsymbol{x}}_{K,1}^m, \tilde{\boldsymbol{x}}_{1,2}^m, \ldots, \tilde{\boldsymbol{x}}_{K,2}^m\right)$. At the beginning of the first FL round, the server sends the model and its initial parameters to a set of selected clients. The clients then update the model parameters on their own data sets. Specifically, at the end of FL round $t-1$, the selected clients send the updated model parameters, $\boldsymbol{\theta}_{t-1}^m, m \in \pi_{t-1}$, to the server, and the server updates the global parameters by aggregating parameters as follows:

$$\boldsymbol{\theta}_{t-1}^{\text{fed}} = \frac{1}{\sum_{m' \in \pi_{t-1}} |\mathcal{D}^{m'}|} \sum_{m \in \pi_{t-1}} |\mathcal{D}^m| \boldsymbol{\theta}_{t-1}^m. \tag{1}$$

In FL round t, the server broadcasts the global parameters, $\boldsymbol{\theta}_{t-1}^{\text{fed}}$, to a set π_t of the clients. Then, each client $m \in \pi_t$ locally updates the received parameters $\boldsymbol{\theta}_{t-1}^{\text{fed}}$ to $\boldsymbol{\theta}_t^m$ with the following objective:

$$\min_{\boldsymbol{\theta}_t^m} \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}^m} \left[\mathcal{L}_{\boldsymbol{\theta}_{t-1}^{\text{fed}}} \left(\mathcal{A}(\boldsymbol{x}), g_{\boldsymbol{\theta}_t^m} \left(f_{\boldsymbol{\theta}_t^m} \left(\mathcal{A}(\boldsymbol{x}) \right) \right) \right) \right], \tag{2}$$

where $\mathcal{A}(\cdot)$ denotes data augmentation and $\mathcal{L}_{\boldsymbol{\theta}_{t-1}^{\text{fed}}}(\cdot, \cdot)$ is the loss function to optimize $\boldsymbol{\theta}_t^m$ when the initial parameters are given by $\boldsymbol{\theta}_{t-1}^{\text{fed}}$. For implementation, FedAvg, proposed by McMahan et al. (2016), can be used to compute $\boldsymbol{\theta}_t^m$ by running stochastic gradient descent (SGD) on $\mathcal{D}^m$ for multiple iterations using the binary cross entropy or MSE loss, with $\boldsymbol{\theta}_{t-1}^{\text{fed}}$ as the initial parameters.

Training Phase 2: The second stage begins when federated training of the sensor modules is finished, say at FL round T. Supplying two different augmented versions $\tilde{\boldsymbol{x}}_{i,j}^m, j = 1, 2$, of each data sample $\boldsymbol{x}_i^m \in \mathcal{D}^m$ to the encoder of the trained autoencoder, client m produces representation vectors, $\boldsymbol{z}_{i,j}^m = f_{\boldsymbol{\theta}_T^{\text{fed}}}(\tilde{\boldsymbol{x}}_{i,j}^m), j = 1, 2$, called the contrastive client representation mini-batches. Then the contrastive client representation mini-batches are transmitted to the server. Combining the representation mini-batches received from at least $Q \leq M$ clients, the server constructs the contrastive server representation mini-batches, $\mathcal{B}^{\text{s}}$, such that each mini-batch $\mathcal{B}^{\text{s}}$ is large enough for contrastive learning. For $\mathcal{B}^{\text{s}}$ at the server, we can define the concept of CB or CIB representations (see Figure 5 in Appendix C.2). In general, the server representations can be (much) more CB, than the raw data at each client. At the server, the discriminator module, $\mathcal{M}_{\boldsymbol{\chi}^{\text{s}}}$, parameterized by χ^{s}, is trained by minimizing the contrastive loss of the contrastive server mini-batches, $\mathcal{B}^{\text{s}}$. Figure 2 shows a schematic view of these two training phases in our proposed method, MFCL.

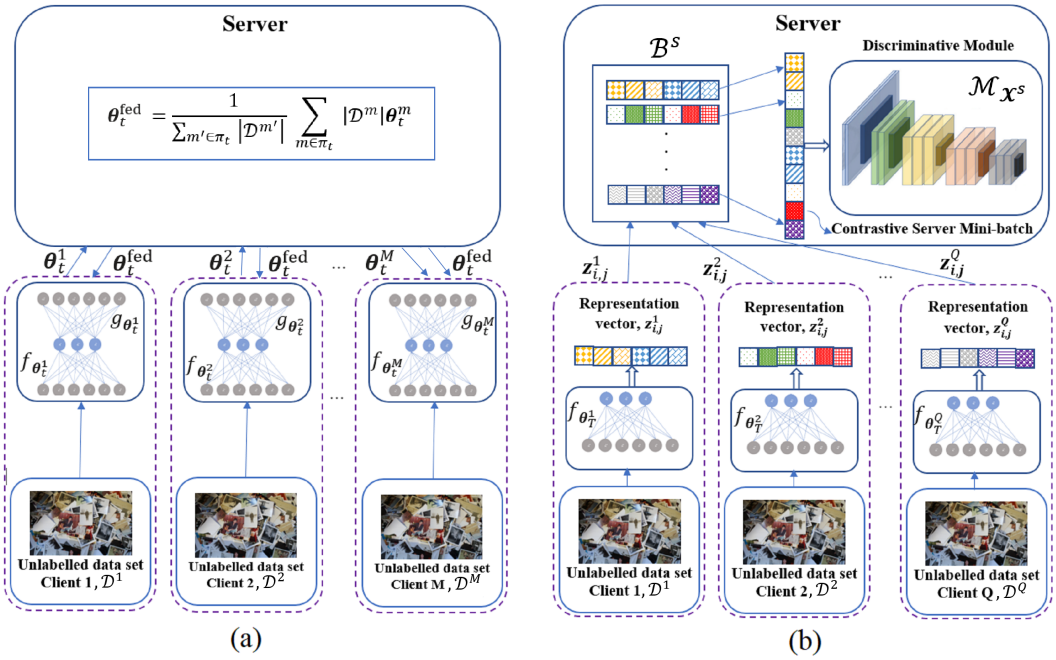


Figure 2: The proposed MFCL framework. (a) Phase 1: training the sensor modules federally across the clients. (b) Phase 2: training the discriminator module at the server.

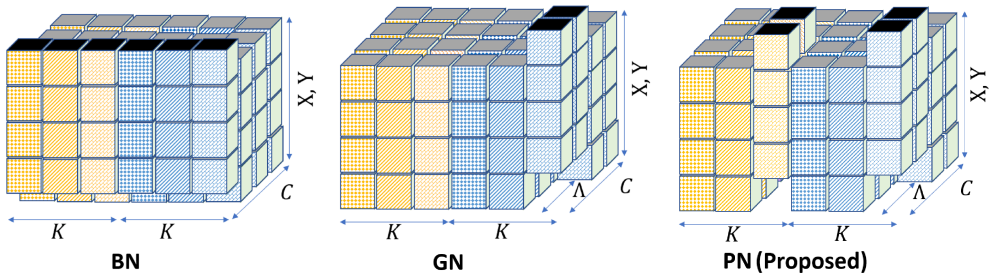


Figure 3: Different normalization techniques for the sensor module (the autoencoder) of each client. The x-axis, denoted by $2K$, represents the elements of each contrastive client mini-batch, composed of two augmented views of the same image. The y-axis, denoted by C, represents the channels and the z-axis, denoted by (X, Y), represents the spatial axes of images' pixels. The misaligned blocks with black tops are simultaneously normalized together with the same mean and variance.

2.2 Normalization of the Sensor Modules

In training neural networks, the distribution of each layer's output changes whenever the parameters of the layer change. Thus, properly normalizing the input to each layer of the network is essential in training. In the following, we first briefly discuss no-normalization (NoN), BN, and GN. We then explain the proposed PN in detail. Figure 3 shows the differences among BN, GN, and PN.

No-Normalization (NoN): Without any normalization, CIB data in the mini-batch can increase the variance of the mini-batch gradients. Therefore, the output distribution of the layers may diverge, which leads to lower accuracy (Wu & He, 2018).

Batch Normalization (BN) (Ioffe & Szegedy, 2015): The strong point of BN is that each channel is normalized over the entire mini-batch. BN works well in contrastive learning when each contrastive client mini-batch is large and CB (this is the case for the contrastive mini-batch at the server). However, when the mini-batch size becomes smaller and/or the data of each mini-batch is CIB, the

batch statistics estimated by BN are biased and inaccurate. For this reason, using BN for training the sensor modules on the heterogeneous and CIB data in FL leads to biased models, which are unable to accurately learn the features of those classes having fewer data samples.

Group Normalization (GN) (Wu & He, 2018): The dependence of the batch-normalized outputs of each layer on the entire mini-batch makes BN powerful; however, it is also the source of BN drawbacks. GN, proposed for distributed training, normalizes a group of channels of one batch element together without relying on the mini-batch size. Specifically, GN *randomly* chooses a group of $\Lambda \leq C$ channels that do not necessarily have the same scale and then those Λ channels are normalized together. Recently, GN has been applied to FL to address the issue of heterogeneous data across the clients (Zhang et al., 2021; Yu et al., 2021; 2020b). In FL, when the mini-batch size is small or CIB, GN generally works better than BN. The reason is that in FL, the model parameters (e.g., the filters' parameters in convolutional neural networks) are averaged over multiple clients. Averaging can somehow make the scales of filters smoother. Applying smoother filters on channels leads to smoother features and normalizing a sample over a group of its channels becomes meaningful. One issue with GN is the amount of information GN uses to normalize one sample over a group of Λ channels. For contrastive client mini-batches, our experiments show that a group of Λ channels do not provide accurate estimations of the mean and variance of each sample. In other words, GN compromises the precision in estimating the mean and variance of each sample over a group of its features in order to make the estimation independent of mini-batch size.

Peer Normalization (PN): For training phase 1 using the contrastive client mini-batches, we propose PN to address the lack of enough information in GN for calculating the mean and variance of a sample over a group of its features. Benefiting from the advantages of GN in addressing the issue of the clients having heterogeneous and CIB data, PN provides more accurate mean and variance for each sample than GN. PN normalizes the positive example *pairs* together, making the positive examples more similar and differentiating them from negative examples.

For mathematical formulation, we let $\boldsymbol{w}_{i,j}^{c,l}, i = 1, \ldots, K; j = 1, 2; c = 1, \ldots, C_l$ denote the attributes of channel c at layer l for the jth augmented version, $\tilde{\boldsymbol{x}}_{i,j}^{m}, j = 1, 2$, of the ith input data sample, $\tilde{\boldsymbol{x}}_i^m$, in the client's mini-batch. We first determine the mean $\boldsymbol{\mu}_i^{\Phi_l,l}$ and variance $(\sigma_i^{\Phi_l,l})^2$ of Λ channels corresponding to augmented versions of each input in the lth layer:

$$\boldsymbol{\mu}_i^{\Phi_l,l} = \frac{1}{2\Lambda} \sum_{j=1}^{2} \sum_{c \in \Phi_l} \boldsymbol{w}_{i,j}^{c,l} \quad \text{and} \quad (\sigma_i^{\Phi_l,l})^2 = \frac{1}{2\Lambda} \sum_{j=1}^{2} \sum_{c \in \Phi_l} \|\boldsymbol{w}_{i,j}^{c,l} - \boldsymbol{\mu}_i^{\Phi_l,l}\|^2 \tag{3}$$

where $\|\cdot\|$ denotes the l_2 norm and $\Phi_l \subseteq \{1, \ldots, C_l\}$ denotes the set composed of Λ channels at layer l. Then, we perform the normalization as follows:

$$\bar{\boldsymbol{w}}_{i,j}^{c,l} = \epsilon_1 \frac{\boldsymbol{w}_{i,j}^{c,l} - \boldsymbol{\mu}_i^{\Phi_l,l}}{\sqrt{(\sigma_i^{\Phi_l,l})^2}} + \epsilon_2, \quad \text{for } c \in \Phi_l, \tag{4}$$

where $\bar{\boldsymbol{w}}_{i,j}^{c,l}$ is the normalized version of $\boldsymbol{w}_{i,j}^{c,l}$ for $c \in \Phi_l$. Also, ϵ_1 and ϵ_2 are trainable scaling factor and shift, respectively. Note that PN ensures that the attributes of two augmented versions of each data sample are normalized independently of the other samples in the contrastive mini-batches at the clients. This way, the statistics used for normalizing the layers' outputs of the autoencoders (i) are independent of mini-batch size (this means that each mini-batch is not necessarily large, which is very advantageous for clients) and (ii) are less biased toward the classes having more samples (this mitigates the adverse effects of heterogeneous and CIB data).

In training phase 2, at the server, constructing the contrastive server representation mini-batches, we train the discriminative module using BN with a contrastive loss. We use BN in the server because the contrastive server mini-batches are better class-balanced and large enough for BN to work well.

3 Experiments

Setup: We define Ω as the universal data set, which is the union of local data sets of all clients, $\Omega = \bigcup_{m=1,\ldots,M} \mathcal{D}^m$. Assuming that the universal data set Ω consists of n classes, we use $\mathcal{V} = \{1, \ldots, n\}$ to denote the set of indices of those n classes. Let $\mathcal{V}^m = \{v_1^m, \ldots, v_{n_m}^m\} \subseteq \mathcal{V}$ denote

Table 1: The universal data set Ω is CIFAR-10. The number M of clients is 10. To cover various degrees of data heterogeneity and class imbalance, ten different scenarios of clients' data sets are considered. For each scenario of each client, the set of indices of classes that have non-zero data samples is shown, where $\{a:b\} = \{a, a+1, a+2, \ldots, b\}$ if $a < b$ and $\{a:b\} = \{a, a+1, \ldots, 10, 1, 2, \ldots, b\}$ if $a > b$. For each scenario for each client, the number of samples of all classes having non-zero samples are the same. Scenario 1 is the extreme case of heterogeneous and CIB data distribution. Scenario 10 is the case of homogeneous and CB data distribution.

Scenario (Sc)	$\mathcal{V}^1$, Client 1	$\mathcal{V}^2$, Client 2	$\cdots$	$\mathcal{V}^9$, Client 9	$\mathcal{V}^{10}$, Client 10	β^{CIB}	β^{hetero}
1	{1}	{2}	$\cdots$	{9}	{10}	3.32	17.94
2	{1:2}	{3:4}	$\cdots$	{8:9}	{10:1}	2.32	15.14
3	{1:3}	{4:6}	$\cdots$	{5:7}	{8:10}	1.73	12.84
4	{1:4}	{5:8}	$\cdots$	{4:7}	{8:1}	1.32	10.82
5	{1:5}	{6:10}	$\cdots$	{1:5}	{6:10}	1.00	8.81
6	{1:6}	{7:2}	$\cdots$	{9:4}	{5:10}	0.74	6.97
7	{1:7}	{8:4}	$\cdots$	{7:3}	{4:10}	0.51	5.14
8	{1:8}	{9:6}	$\cdots$	{5:2}	{3:10}	0.32	3.38
9	{1:9}	{10:8}	$\cdots$	{3:1}	{2:10}	0.15	1.67
10	{1:10}	{1:10}	$\cdots$	{1:10}	{1:10}	0.00	0.00

the set of indices of the classes for client m's data set $\mathcal{D}^m$, where n_m is the number of classes with non-zero samples for client m. We perform our experiments for the two widely adopted data sets, CIFAR-10 and CIFAR-100, as Ω (Krizhevsky, 2009). The sensor module is a convolutional autoencoder that is trained federally across clients, of which structure is presented in Table 5 in Appendix C.3. The discriminator module at the server is ResNet-18, adopted to be trained on the received representation vectors with a contrastive loss. We implemented MFCL with TensorFlow 2.9 following the standard structure of FL in McMahan et al. (2017) for training the sensor modules federally, and following the method in Chen et al. (2020) for training the discriminator module at the server. We considered a wide range of heterogeneous and CIB data scenarios for our experiments, which are listed in Table 1 for CIFAR-10 (and in Table 4 in Appendix C.1 for CIFAR-100).

We use Kullback-Leibler (KL) divergence to quantify how much heterogeneity exists in the data sets over the clients and how much CIB the data distribution of each client is. Let $N(v) > 0$ denote the number of samples in class $v \in \mathcal{V}$. Also, $P^m = (p^m(1), \ldots, p^m(n))$ denotes the vector presenting the distribution of the mth client's data set $\mathcal{D}^m$, where $p^m(v) = \frac{N(v)}{|\mathcal{D}^m|}$ if $v \in \mathcal{V}^m$ and $p^m(v) = 0$ if $v \notin \mathcal{V}^m$. We propose to use $\beta^{\text{CIB}} = \frac{1}{M}\sum_{m=1}^{M} \beta^m$ to denote the average degree of class imbalance of each client's data set, where $\beta^m = \text{KL}(P^m, \mathcal{U}) = \log_2(n) - H(P^m)$. In this equation, $\mathcal{U}$ denotes a uniform distribution of length n, given by $\mathcal{U} = (\frac{1}{n}, \ldots, \frac{1}{n})$. We also use $\beta^{\text{hetero}} = \frac{1}{M(M-1)}\sum_{m=1}^{M}\sum_{z\in[1:M], z\neq m} \beta^{m,z}$ to denote the average degree of data heterogeneity across the clients. In this equation, $\beta^{m,z} = \text{KL}\left(P^m, P^z\right) = H\left(P^m, P^z\right) - H\left(P^m\right)$ denotes how much the data distribution of client m differs from that of client z. More details of β^{CIB} and β^{hetero} are given in Appendix B.

Performance metric: We evaluated the classification accuracy of the proposed MFCL based on the evaluation method widely adopted for self-supervised learning (Chen et al., 2020). Specifically, we used non-linear projection layers on top of the discriminator module. The structure of the non-linear projection layers are presented in Table 6 in Appendix C.3. Also, we used one linear Dense layer on top of the projection layers for linear evaluation. In training phase 2, the discriminator module, the projection layers, and the linear (classifier) layer can be jointly trained as long as the gradients are not backpropagated from the linear (classifier) layers to the discriminator module and the projection layers. For downstream tasks, the linear (classifier) layer is discarded. Alternatively, in training phase 2, it is possible to first train the discriminator module and the projection layers with representation vectors in the unsupervised manner, and then, the Dense layer connected on top of the frozen (trained) discriminator module/projection layers is trained with labeled data.

Baselines: For performance comparison, we consider the state-of-the-art works in self-supervised FL and supervised FL that were proposed to address data heterogeneity. In the field of self-supervised FL, we combined SwAV, in Caron et al. (2020), and SimCLR, in Chen et al. (2020),

Table 2: Accuracy comparison of the proposed MFCL (with BN, GN, and PN in the sensor module) and the baseline methods (in self-supervised FL and supervised FL) on CIFAR-10 for the 10 scenarios defined in Table 1 with $M = 10$ clients. The best performance is highlighted in boldface. For self-supervised methods, the top-1 accuracy with linear evaluation is reported. In all simulation scenarios for MFCL, BN is used at the server.

	supervised FL			**self-supervised FL**					
Sc	FedAvg McMahan et al. (2016)	FedProx Li et al. (2020b)	MOON Li et al. (2021)	FedSwAV Luo et al. (2021)	FedSimCLR Luo et al. (2021)	Orchestra Lubana et al. (2022)	MFCL(BN) proposed	MFCL(GN) proposed	MFCL(PN) proposed
1	0.57(±0.05)	0.57(±0.03)	0.58(±0.03)	0.75(±0.05)	0.71(±0.02)	0.73(±0.02)	0.10(±0.03)	0.79(±0.02)	**0.83(±0.02)**
2	0.64(±0.04)	0.64(±0.02)	0.64(±0.03)	0.75(±0.03)	0.73(±0.03)	0.78(±0.03)	0.17(±0.05)	0.79(±0.03)	**0.84(±0.02)**
3	0.67(±0.04)	0.68(±0.02)	0.69(±0.04)	0.77(±0.04)	0.74(±0.01)	0.78(±0.01)	0.29(±0.11)	0.79(±0.01)	**0.84(±0.02)**
4	0.73(±0.02)	0.73(±0.03)	0.74(±0.01)	0.78(±0.02)	0.75(±0.04)	0.79(±0.04)	0.39(±0.02)	0.80(±0.01)	**0.84(±0.01)**
5	0.77(±0.03)	0.78(±0.04)	0.78(±0.01)	0.79(±0.04)	0.75(±0.03)	0.79(±0.02)	0.47(±0.06)	0.80(±0.01)	**0.84(±0.01)**
6	0.82(±0.04)	0.82(±0.03)	0.84(±0.04)	0.79(±0.01)	0.76(±0.04)	0.80(±0.04)	0.58(±0.05)	0.81(±0.03)	**0.84(±0.02)**
7	0.86(±0.03)	0.88(±0.04)	**0.88(±0.01)**	0.80(±0.03)	0.77(±0.02)	0.80(±0.01)	0.67(±0.08)	0.81(±0.03)	0.84(±0.01)
8	0.90(±0.04)	0.91(±0.03)	**0.91(±0.01)**	0.81(±0.01)	0.79(±0.04)	0.81(±0.03)	0.77(±0.02)	0.81(±0.01)	0.85(±0.03)
9	0.92(±0.04)	0.92(±0.04)	**0.93(±0.04)**	0.82(±0.04)	0.81(±0.03)	0.82(±0.04)	0.85(±0.03)	0.82(±0.03)	0.85(±0.03)
10	0.92(±0.02)	0.93(±0.02)	**0.94(±0.02)**	0.83(±0.02)	0.83(±0.03)	0.82(±0.01)	0.88(±0.02)	0.82(±0.01)	0.85(±0.01)

Table 3: Accuracy comparison on CIFAR-100 for one hundred clients ($M = 100$) in 8 different scenarios given in Table 4 in Appendix C.1. Other settings are the same as in Table 2.

	supervised FL			**self-supervised FL**					
Sc	FedAvg McMahan et al. (2016)	FedProx Li et al. (2020b)	MOON Li et al. (2021)	FedSwAV Luo et al. (2021)	FedSimCLR Luo et al. (2021)	Orchestra Lubana et al. (2022)	MFCL(BN) proposed	MFCL(GN) proposed	MFCL(PN) proposed
11	0.18(±0.02)	0.18(±0.02)	0.20(±0.01)	0.31(±0.03)	0.31(±0.03)	0.37(±0.02)	0.02(±0.02)	0.40(±0.02)	**0.43(±0.02)**
12	0.20(±0.03)	0.21(±0.01)	0.22(±0.03)	0.34(±0.02)	0.34(±0.03)	0.40(±0.03)	0.04(±0.04)	0.43(±0.03)	**0.45(±0.02)**
13	0.21(±0.04)	0.22(±0.02)	0.23(±0.01)	0.36(±0.05)	0.35(±0.02)	0.40(±0.02)	0.07(±0.05)	0.44(±0.01)	**0.45(±0.01)**
20	0.27(±0.03)	0.27(±0.01)	0.28(±0.02)	0.37(±0.02)	0.36(±0.02)	0.40(±0.01)	0.14(±0.01)	0.44(±0.01)	**0.45(±0.01)**
30	0.31(±0.02)	0.31(±0.01)	0.33(±0.03)	0.37(±0.03)	0.38(±0.01)	0.41(±0.01)	0.18(±0.02)	0.45(±0.02)	**0.46(±0.01)**
70	0.47(±0.04)	0.47(±0.01)	**0.48(±0.02)**	0.40(±0.02)	0.40(±0.02)	0.42(±0.01)	0.45(±0.01)	0.45(±0.01)	0.47(±0.01)
109	0.57(±0.01)	0.57(±0.01)	**0.57(±0.03)**	0.44(±0.01)	0.46(±0.01)	0.49(±0.01)	0.50(±0.02)	0.48(±0.02)	0.49(±0.02)
110	0.57(±0.02)	0.57(±0.01)	**0.58(±0.04)**	0.45(±0.02)	0.46(±0.01)	0.49(±0.01)	0.51(±0.02)	0.49(±0.01)	0.49(±0.02)

with FedAvg, which are respectively denoted as FedSwAV and FedSimCLR. We then applied the regularization techniques of Luo et al. (2021) to FedSwAV and FedSimCLR. FedAvg in McMahan et al. (2016), FedProx in Li et al. (2020b), and MOON in Li et al. (2021) are fully supervised FL.

Results: Table 2 shows the experimental results of our method and baselines on CIFAR-10 with ten clients ($M = 10$) under different heterogeneous and CIB settings (Sc1 to Sc9) and homogeneous and CB setting (Sc10). In each scenario, the top-1 accuracy on a uniform test set is reported. Each scenario is run 3 times, and the mean and standard deviations are reported. Table 3 shows the experimental results of our method and baselines on CIFAR-100 with 100 clients ($M = 100$). The simulation scenarios for CIFAR-100 are presented in Table 4 in Appendix C.1.

From the numerical results, we see that, in the extreme case of heterogeneous and CIB data sets (e.g., Sc1 and Sc2), the proposed MFCL scheme significantly outperforms the other methods, whether based on supervised FL or unsupervised FL. Furthermore, in those scenarios, the proposed MFCL scheme with PN outperforms that with GN or BN. The reasons why the proposed MFCL works well in heterogeneous and CIB scenarios in the FL setting are as follows. First, MFCL benefits from self-supervised learning technique, which has been mathematically and experimentally proved to be more robust to CIB data (Liu et al., 2021). In supervised learning, the expected values of the squares of the lengths of the gradient vectors corresponding to different classes are approximately related to the square of the number of samples in those classes (Anand et al., 1993). This means that when the data set is CIB, the gradients of the classes with more samples are larger than those of the classes with fewer samples, which leads to the gradients biased toward the class with more samples. In contrast to this, contrastive loss does not rely on the labels to calculate the loss values, and thus, the gradients are less biased and the model can better learn about those classes having fewer samples. Therefore, with self-supervised learning techniques, we can mitigate the *bias of labels*.

Second, MFCL benefits from modular training. Although we can suppress the bias of labels, we still have the *bias of clusters*. In training with a contrastive loss, the model learns to put the positive examples in the same clusters and gradually increase the distance between the positive and negative examples. However, when the mini-batch is CIB, there is not enough diversity of samples for constructing clusters correctly. Therefore, we end up having larger clusters for classes with more samples and having smaller clusters for classes with fewer samples. Again, biased clusters lead to biased gradients. With modular training, we do not have to backpropagate the gradients from the last layers of discriminator at the server (i.e., ResNet-18), which are responsible for determining the clusters, to the layers extracting the representations from the input data in the sensor module at the clients (i.e., the autoencoders). This helps the sensor modules learn less biased representations from the data. Therefore, with modular training we can mitigate the *bias of clusters*.

Third, MFCL benefits from PN. In severe heterogeneous and CIB scenarios, in addition to the bias of labels and bias of clusters, we face the *bias of features*. The bias of labels and clusters can be mitigated by new designs of the network (e.g., our proposed modular training instead of the end-to-end (E2E) training, as discussed in Appendix A). However, the bias of features is the natural result of having clients with heterogeneous and CIB data. One effective way to moderate the bias of features is to use properly designed normalization techniques. PN adopted in the sensor modules moderates the effect of bias of features when data is severely heterogeneous and CIB. The advantage of PN can also be demonstrated by t-SNE visualizing the learned clusters. For more details, see Appendix D.1 and Figure 6. The benefit of PN also extends to other semi-supervised FL setting. For example, when PN is applied to FedSimCLR Luo et al. (2021), the performance is consistently improved. For more details, see Appendix D.3 and Table 7.

The proposed MFCL scheme works (very) well when data sets are severely heterogeneous and CIB. In (closely) homogeneous and CB scenarios (e.g., Sc7 to Sc10), however, supervised FL methods outperform MFCL. The reasons can be as follows. First, when the clients have homogeneous and CB data, the gradients are not biased. In this case, the labels are homogenous, and there is nothing like the bias of labels. In these scenarios, supervised learning works better than self-supervised learning, including the centralized setting (Wu & He, 2018). Second, when the clients have homogeneous and CB data, the clusters and the gradients are not biased. In this case, backpropagating the unbiased gradients from the last layers of the discriminator to the first layers of the sensor module as in the E2E training in the FL setting helps the network to learn the data more accurately. Third, sample-based normalization techniques (e.g., GN, which normalizes Λ channels of one example together, or PN, which normalizes Λ channels of two positive examples together) work better than BN when the mini-batch is CIB or small. However, when the mini-batch is CB and large enough, normalizing the data over the entire mini-batch (i.e., BN) works better. The reason is that the batch statistics estimated from a large and CB mini-batch is more accurate, yielding smaller variations of statistics from one mini-batch to another, and thus, it leads to better generalization performance.

Another issue that needs to be taken into account for practical FL implementation is that not all the clients who participated in training phase 1 necessarily participate in training phase 2. This issue and the performance results are presented in Appendix D.2 and Figure 7, which show that our proposed MFCL scheme is robustly working even in that situation. Last but not least, it is worth checking the amount of communication burden between the clients and the server, which is the major bottleneck for the actual implementation of FL. Thanks to the modular structure, our proposed MFCL involves a much lower communication burden, which is discussed in detail in Appendix D.4 and Table 8.

4 Conclusions

We have provided a new perspective for FL by changing the training framework from E2E to modular in order to address the statistical challenges of FL when the clients have unlabeled, heterogeneous, and CIB data. In particular, we have proposed a new normalization scheme, PN, to facilitate contrastive learning in the FL setting. Our extensive simulations show that the proposed MFCL with PN has better and stable accuracy in various scenarios of heterogeneous and CIB data. The proposed MFCL also works well with GN, which means that modular training does not have to be limited to contrastive learning and can also be used with the non-contrastive methods in the server. We release our codes to encourage developments in designing new modular architectures for FL.

REFERENCES

Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *Proceedings of International Conference on Learning Representations*, 2021.

Rangachari Anand, Kishan G Mehrotra, Chilukuri K Mohan, and Sanjay Ranka. An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969, 1993.

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, pp. 2938–2948, 2020.

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems*, pp. 9912–9924, 2020.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of International Conference on Machine Learning*, pp. 1597–1607, 2020.

Aymeric Dieuleveut, Gersende Fort, Eric Moulines, and Geneviève Robin. Federated expectation maximization with heterogeneity mitigation and variance reduction. *arXiv preprint arXiv:2111.02083*, 2021.

Moming Duan, Duo Liu, Xianzhang Chen, Renping Liu, Yujuan Tan, and Liang Liang. Self-balancing federated learning with global imbalanced data in mobile systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):59–71, 2020.

Alp Emre Durmus, Zhao Yue, Matas Ramon, Mattina Matthew, Whatmough Paul, and Saligrama Venkatesh. Federated learning based on dynamic regularization. In *Proceedings of International Conference on Learning Representations*, 2021.

Weituo Hao, Mostafa El-Khamy, Jungwon Lee, Jianyi Zhang, Kevin J Liang, Changyou Chen, and Lawrence Carin Duke. Towards fair federated learning with zero-shot data augmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3310–3319, 2021.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

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.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of International Conference on Machine Learning*, pp. 448–456, 2015.

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

Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016a.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016b.

A Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pp. 661–670, 2014.

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, 2021.

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. *Machine Learning and Systems*, 2:429–450, 2020b.

Hong Liu, Jeff Z. HaoChen, Adrien Gaidon, and Tengyu Ma. Self-supervised learning is more robust to dataset imbalance. In *Proceedings of NeurIPS Workshop on Distribution Shifts*, 2021.

Nan Lu, Zhao Wang, Xiaoxiao Li, Gang Niu, Qi Dou, and Masashi Sugiyama. Federated learning from only unlabeled data with class-conditional-sharing clients. In *Proceedings of International Conference on Learning Representations*, 2022.

Ekdeep Singh Lubana, Chi Ian Tang, Fahim Kawsar, Robert P Dick, and Akhil Mathur. Orchestra: Unsupervised federated learning via globally consistent clustering. *arXiv preprint arXiv:2205.11506*, 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*, 2021.

David JC MacKay, David JC Mac Kay, et al. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Disha Makhija, Nhat Ho, and Joydeep Ghosh. Federated self-supervised learning for heterogeneous clients. *arXiv preprint arXiv:2205.12493*, 2022.

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

H Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2016.

Runxuan Miao and Erdem Koyuncu. Federated momentum contrastive clustering. *arXiv preprint arXiv:2206.05093*, 2022.

Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

Xutong Mu, Yulong Shen, Ke Cheng, Xueli Geng, Jiaxuan Fu, Tao Zhang, and Zhiwei Zhang. Fedproc: Prototypical contrastive federated learning on non-iid data. *arXiv preprint arXiv:2109.12273*, 2021.

Vaikkunth Mugunthan, Antigoni Polychroniadou, David Byrd, and Tucker Hybinette Balch. Smpai: Secure multi-party computation for federated learning. In *Proceedings of the NeurIPS Workshop on Robust AI in Financial Services*, 2019.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *Proceedings of International Conference on Machine Learning*, pp. 3067–3075, 2017.

Haizhou Shi, Youcai Zhang, Zijin Shen, Siliang Tang, Yaqian Li, Yandong Guo, and Yueting Zhuang. Federated self-supervised contrastive learning via ensemble similarity distillation. *arXiv preprint arXiv:2109.14611*, 2021.

MyungJae Shin, Chihoon Hwang, Joongheon Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Xor mixup: Privacy-preserving data augmentation for one-shot federated learning. *arXiv preprint arXiv:2006.05148*, 2020.

Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *Proceedings of International Conference on Learning Representations*, 2020.

Lixu Wang, Shichao Xu, Xiao Wang, and Qi Zhu. Addressing class imbalance in federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10165–10173, 2021.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Miao Yang, Ximin Wang, Hongbin Zhu, Haifeng Wang, and Hua Qian. Federated learning with class imbalance reduction. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, pp. 2174–2178, 2021.

Felix Yu, Ankit Singh Rawat, Aditya Menon, and Sanjiv Kumar. Federated learning with only positive labels. In *Proceedings of International Conference on Machine Learning*, pp. 10946–10956, 2020a.

Fuxun Yu, Weishan Zhang, Zhuwei Qin, Zirui Xu, Di Wang, Chenchen Liu, Zhi Tian, and Xiang Chen. Heterogeneous federated learning. *arXiv preprint arXiv:2008.06767*, 2020b.

Fuxun Yu, Weishan Zhang, Zhuwei Qin, Zirui Xu, Di Wang, Chenchen Liu, Zhi Tian, and Xiang Chen. Fed2: Feature-aligned federated learning. In *Proceedings of the Conference on Knowledge Discovery and Data Mining*, pp. 2066–2074, 2021.

Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *Proceedings of USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 493–506, 2020a.

Fengda Zhang, Kun Kuang, Zhaoyang You, Tao Shen, Jun Xiao, Yin Zhang, Chao Wu, Yueting Zhuang, and Xiaolin Li. Federated unsupervised representation learning. *arXiv preprint arXiv:2010.08982*, 2020b.

Zhengming Zhang, Yaoqing Yang, Zhewei Yao, Yujun Yan, Joseph E Gonzalez, Kannan Ramchandran, and Michael W Mahoney. Improving semi-supervised federated learning by reducing the gradient diversity of models. In *Proceedings of IEEE International Conference on Big Data (Big Data)*, pp. 1214–1225, 2021.

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.

Appendix

A Related works

E2E Training vs Modular Training: End-to-end (E2E) training means training a differentiable learning using a single overall loss (objective) function. Although the E2E training performs well on many tasks, in some cases (e.g., ill-conditioned problems), it may converge slowly, converge into possibly local optima, and face vanishing gradients (Shalev-Shwartz et al., 2017). By contrast, in modular training, different tasks are learned by separate network modules with different loss functions, and the gradients of each network module are not back propagated to the other modules. In our proposed scheme, we adopt the approach of modular training: the sensor module is used to federally extract the representations of clients' data, and the discriminator module at the server performs the classification task.

Supervised FL with clients' heterogeneous data sets: There are plenty of works aiming to address the issue of heterogeneous data of the clients in the supervised learning-based FL, which can be classified as follows. The first approach is leveraging a shared public or synthesized data set in the clients or server. These methods try to find a solution for the local models that under-represent the patterns of minor classes having fewer samples at the clients (Zhao et al., 2018); (Hao et al., 2021). The second approach is based on regularization. FedProx, studied in Li et al. (2020b), regularizes the Euclidean distance between the local and global models. MOON, proposed by Li et al. (2021), uses contrastive loss to maximize the agreement of the representations learned by the local and global models. SCAFFOLD, introduced by Karimireddy et al. (2020), reduces the bias in the gradients by introducing some control variates. FedDyn, proposed by Acar et al. (2021), dynamically changes the local objectives at each FL round to ensure that the local optimum is consistent with the global optimum. The third approach is based on aggregation on the server. The authors in Hsu et al. (2019), leveraging the momentum update on the server, tried to alleviate the oscillations resulting from averaging the biased gradients. Wang et al. (2020) proposed to match the local updates while aggregating to reduce the effect of heterogeneous data.

Supervised FL with clients' CIB data: In the FL setting, the issue of CIB data for each client has been studied only in very limited works. Astraea, proposed by Duan et al. (2020), tries to alleviate the CIB issue by data augmentation and multi-client rescheduling based on the KL divergence of the clients' data distribution. Recently, Wang et al. (2021) proposed a regularization-based method to monitor the composition of data according to the current global model in each FL round to mitigate the impact of CIB data.

Contrastive learning and self-supervised FL: The contrastive loss, used in Oord et al. (2018), is defined between two augmented views (i, j) of the same data sample in a mini-batch of the size of $2K$, as follows:

$$\mathcal{L} = -\frac{1}{2K} \sum_{i,j \in \mathcal{B}} \log \frac{\exp\left(\frac{\text{sim}(\boldsymbol{h}_i, \boldsymbol{h}_j)}{\tau}\right)}{\sum_{k=1}^{2K} \mathbb{1}_{[k \neq i]} \exp\left(\frac{\text{sim}(\boldsymbol{h}_i, \boldsymbol{h}_k)}{\tau}\right)}, \tag{5}$$

where $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are hidden representations of two positive examples[1], $\mathbb{1}_{[k \neq i]}$ is an indicator function that is equal to 1 if $k \neq i$, $\text{sim}(\boldsymbol{h}_i, \boldsymbol{h}_j)$ is the cosine similarity between two vectors of hidden representations, τ is a temperature scalar, and $\mathcal{B}$ is a randomly sampled mini-batch consisting of augmented pairs of images. Only few works have studied contrastive learning for unsupervised FL (Zhang et al., 2020b); (Miao & Koyuncu, 2022); (Shi et al., 2021); (Yu et al., 2020a). For representation learning without a contrastive loss, Lu et al. (2022) proposed to use prior information about the clients' data set, instead of labels, to apply supervised learning methods to unlabeled data. Recently, Lubana et al. (2022) proposed Orchestra as an unsupervised FL scheme to partition the client's data into discriminable clusters. The topic of unsupervised FL or self-supervised FL has been very underexplored in the literature.

[1]By means of examples, we mean augmented data samples. Positive examples are two differently augmented versions of the same image, and negative examples are the augmented versions of other images.

B METRICS QUANTIFYING DATA HETEROGENEITY AND CLASS-IMBALANCE

For two probability distributions, A and B, defined on the probability space, $\mathcal{X}$, the KL divergence from B to A is defined as $\mathrm{KL}(A, B) = -\sum_{x \in \mathcal{X}} A(x) \log_2 \left(\frac{B(x)}{A(x)} \right)$ (MacKay et al., 2003). The KL divergence is defined only if for all x, $B(x) = 0$ implies $A(x) = 0$. When $A(x) = 0$, since $\lim_{x \to 0^+} x \log(x) = 0$, the contribution of the term corresponding to $A(x)$ would be zero. The KL divergence, β^m, between the distribution, P^m, of the mth client's data and the uniform distribution, $\mathcal{U}$, is given by:

$$\begin{aligned} \beta^m = \mathrm{KL}(P^m, \mathcal{U}) &= \sum_{v \in \{1,\dots,n\}} p^m(v) \log_2 \left(\frac{p^m(v)}{\mathcal{U}(v)} \right) \\ &= \log_2(n) + \sum_{v \in \{1,\dots,n\}} p^m(v) \log_2 \left(p^m(v) \right) \\ &= \log_2(n) + \sum_{e=1}^{n_m} p^m(v_e^m) \log_2 \left(p^m(v_e^m) \right) = \log_2(n) - H\left(P^m\right) \end{aligned} \tag{6}$$

where $H(P^m)$ is the entropy of P^m. In order to numerically compute the KL divergence in our setting, when some classes do not have any samples, we set $p^m(v) = \varsigma$ for $v \notin \mathcal{V}^m$, where ς is a small value, such that $N \times \varsigma \ll 1$, where N is the total number of samples in the universal data set, $\mathcal{V}$. This means that there is not even one sample in the particular class v. In our simulation, we set, $\varsigma = 10^{-6}$. The average of β^m is β^{CIB}. The smaller β^{CIB} means the data set in each client is closer to the CB data.

The KL divergence between the mth client's distribution, P^m, and the zth client's distribution, P^z, $z \neq m$, is defined as follows:

$$\begin{aligned} \beta^{m,z} = \mathrm{KL}(P^m, P^z) &= \sum_{v \in \{1,\dots,n\}} p^m(v) \log_2 \left(\frac{p^m(v)}{p^z(v)} \right) \\ &= \sum_{v \in \{1,\dots,n\}} p^m(v) \log_2 \left(p^m(v) \right) - \sum_{v \in \{1,\dots,n\}} p^m(v) \log_2 \left(p^z(v) \right) \\ &= H\left(P^m, P^z\right) - H\left(P^m\right) \end{aligned} \tag{7}$$

where $H(P^m, P^z)$ is the cross entropy between P^m and P^z. The average of $\beta^{m,z}$ over m and z is β^{hetero}. The smaller β^{hetero} means the clients are more close to be homogeneous.

C EXPERIMENTAL DETAILS

In this section, we illustrate the details of the model architectures and some additional simulation scenarios.

C.1 SIMULATION SCENARIOS FOR CIFAR-100

The simulation scenarios for CIFAR-100 with 100 clients are presented in Table 4.

C.2 HETEROGENEOUS AND CIB DATA

In this paper, we use the terminologies of homogeneous/heterogeneous and CB/CIB data in the following sense. The concept of CB or CIB data is defined for a single (or each individual) data set at each client (and possibly at the server as well): we say a data set is CB when the data set contains the same number of samples over all possible classes of the universal data set Ω that is defined as the union of the data sets of all clients. Meanwhile, the concept of homogeneous or heterogeneous data is defined over multiple data sets in multiple clients: we say the clients' data is heterogeneous when

Table 4: The universal data set Ω is CIFAR-100. The number M of clients is 100. In principle, 100 different scenarios can be defined based on our notation; however, for simplicity, only some limited number of scenarios are shown in this table. For each scenario of each client, the set of indices of classes that have non-zero data samples is shown, where $\{a:b\} = \{a, a+1, a+2, \ldots, b\}$ if $a < b$ and $\{a:b\} = \{a, a+1, \ldots, 100, 1, 2, \ldots, b\}$ if $a > b$. For each scenario for each client, the number of samples of all classes having non-zero samples are the same. Scenario 11 is the extreme case for heterogeneous clients with CIB data set for each client. Scenario 110 is the case for homogeneous clients with CB data set for each client.

Scenario (Sc)	$\mathcal{V}^1$, Client 1	$\mathcal{V}^2$, Client 2	$\cdots$	$\mathcal{V}^{99}$, Client 99	$\mathcal{V}^{100}$, Client 100	β^{CIB}	β^{hetero}
11	{1}	{2}	$\cdots$	{9}	{100}	6.64	19.73
12	{1:2}	{3:4}	$\cdots$	{8:9}	{100:1}	5.63	18.54
13	{1:3}	{4:6}	$\cdots$	{5:7}	{8:100}	5.06	17.79
$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
20	{1:10}	{11:20}	$\cdots$	{81:90}	{91:100}	3.32	14.99
$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
30	{1:20}	{21:40}	$\cdots$	{61:80}	{81:100}	2.32	13.81
$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
70	{1:60}	{61:20}	$\cdots$	{81:40}	{41:100}	0.74	5.60
$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
109	{1:99}	{100:98}	$\cdots$	{3:1}	{2:100}	0.00	0.00
110	{1:100}	{1:100}	$\cdots$	{1:100}	{1:100}	0.00	0.00

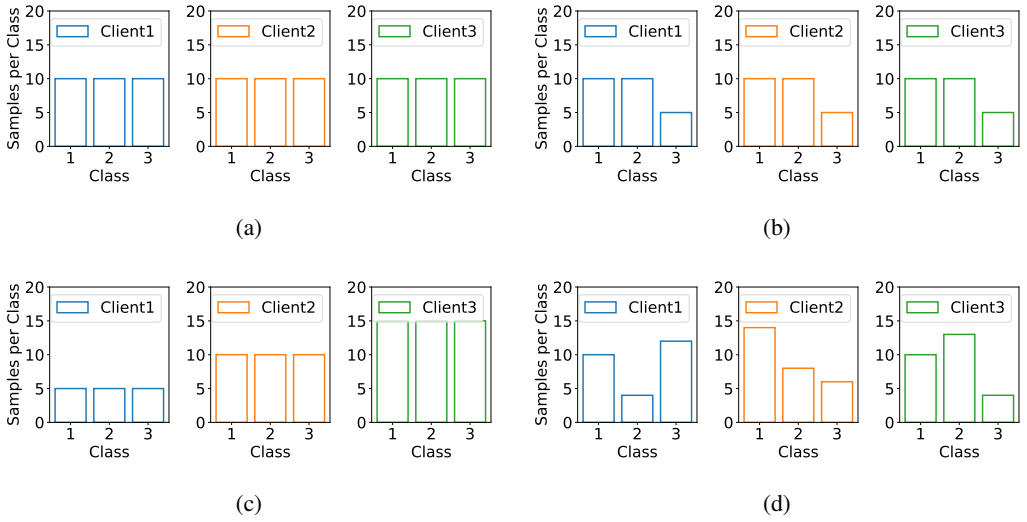


Figure 4: Different scenarios for the raw data in the clients. (a) The clients have homogeneous and CB data, (b) the clients have homogeneous and CIB data, (c) the clients have heterogeneous and CB data, and (d) the clients have heterogeneous and CIB data.

the data is not identically distributed across all clients. Figure 4 shows the four different scenarios of data distribution of clients in FL having homogeneous/heterogeneous and CB/CIB data.

In our proposed MFCL scheme, the data representations sent from the clients are collected at the server, and the contrastive server mini-batches are constructed with the received representation vectors at the server. Therefore, we need to define the concept of CB or CIB representations at the server. For clarification, we provide Figure 5, which consists of data distribution in the clients and the distribution of the representations at the server. In Figure 5 (a), the clients have homogeneous and CB data in each client; In Figure 5 (b), the clients have homogeneous and CIB data in each client; In Figures 5 (c) and 5 (d), the clients have heterogeneous and CIB data in each client; In Figure 5 (e),

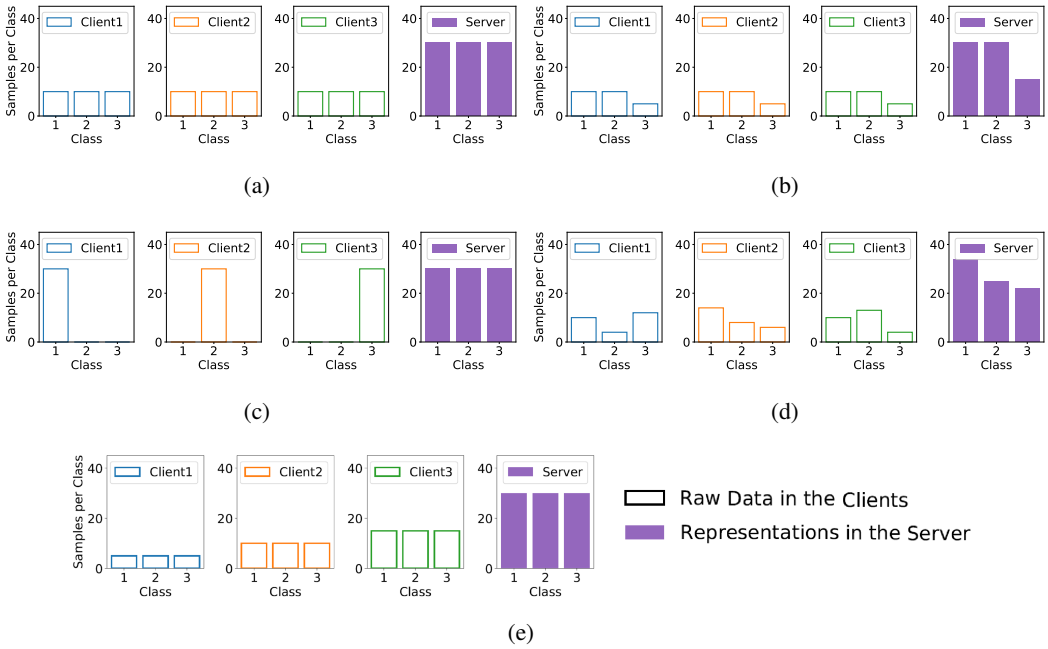


Figure 5: Different scenarios for the (raw) data in the clients and the representations in the server. (a) The clients have homogeneous and CB data, and the representations are CB in the server, (b) the clients have homogeneous and CIB data, and the representations are CIB in the server, (c) the clients have heterogeneous and CIB data, and the representations are CB in the server, (d) the clients have heterogeneous and CIB data, and the representations are CIB in the server, and (e) the clients have heterogeneous and CB data, and the representations are CB in the server.

the clients have heterogeneous and CB data in each client. When the clients send their representation vectors of their own local raw data to the server, the server constructs mini-batches collecting the received representation vectors from the clients. The server representation mini-batches (more precisely, the contrastive server representation mini-batches) can be CB or CIB. If the data set is CB in all clients, the server mini-batch is always CB, even if some clients are not successful in sending their representation vectors (Figures 5 (a) and 5 (e)). Otherwise, the server mini-batch can be CB or CIB: the server mini-batch is CIB in Figures 5 (b) and 5 (d), whereas it is CB in Figure 5 (c).

C.3 Model Architecture

For our experiments, we use a convolutional autoencoder as the sensor module, which is trained federally, with binary cross entropy loss, across $M = 10$ clients for CIFAR-10, where the training mini-batch size is 64, the learning rate is 0.001, and the optimizer is ADAM. The sensor module details are listed in Table 5.

The classification accuracy of the proposed MFCL is evaluated based on the evaluation method widely adopted for self-supervised learning (Chen et al., 2020). Specifically, we use non-linear projection layers on top of the discriminator module and the structure of the nonlinear projection layers are presented in Table 6. Also, one linear Dense layer is used on top of the projection layers for linear evaluation.

D Additional Experimental Results

D.1 Visualization of Learned Clusters by MFCL

In this appendix, we visualize the features of the learned clusters to better understand the advantage of the proposed normalization method, PN. We consider the case of data heterogeneous clients with

Table 5: The architecture of the sensor module, which is a convolutional autoencoder. For convolutional layer (Conv2D) and transposed convolution layer (Conv2DTranspose), we show the list of the parameters in the sequence of input and output dimensions, kernel size, stride, and padding. For max pooling layer (MaxPool2D), we show the list the kernel and stride.

Layer	Details
1	Conv2D(3, 64, 3, 1, 1)
2	PN, ReLU, MaxPool2D(2, 2)
3	Conv2D(64, 64, 3, 1, 1)
4	PN, ReLU
5	Conv2DTranspose(64, 64, 3, 1, 1)
6	PN, ReLU
7	Conv2DTranspose(64, 3, 3, 1, 1)
8	PN, ReLU
9	Conv2D(3, 3, 3, 1, 1)

Table 6: The projection layer architecture.

Layer	Details
1	Dense
2	BN, ReLU
3	Dense
4	BN, ReLU
5	Dense
6	BN

CIB data, Sc2, in which each client has only 2 classes when $M = 10$ on CIFAR-10. We aim to check the quality of the learned clusters with MFCL, after the projection layers at the server, in the sense of number of distinguishable clusters, when BN, GN, or PN are used at the sensor modules of the clients.

In Sc2, each client only has two classes as defined in Table 1. Figure 6 (a) shows the t-SNE visualization of raw input data at a single client, while Figures 6 (b), 6 (c), and 6 (d) show the t-SNE visualizations of the clusters learned from the data representations (i.e., the combined representation vectors from all clients) at the server of our proposed MFCL method. Figure 6 (b) shows the learned clusters by MFCL when BN is used in the sensor module at the clients. The proposed MFCL with BN only learns 3 distinguishable clusters for truck, ship, and frog. There are also wrong predictions for some of the samples belonging to the ship's cluster. In other words, MFCL with BN cannot accurately separate the samples of airplanes from those of ships. Also, there is a small cluster of automobiles, and the rest of the samples of automobiles are very close to the cluster of trucks. Figure 6 (c) shows the learned clusters by MFCL when GN is used in the sensor module at the clients. The proposed MFCL with GN is able to learn 5 distinguishable clusters for truck, ship, frog, automobile, and airplane. Also, the clusters for airplanes and ships are more separated. Figure 6 (d) shows the learned clusters by MFCL when PN is used in the sensor module at the clients. The proposed MFCL with PN is able to learn 6 distinguishable clusters for truck, ship, frog, automobile, airplane, and horse.

D.2 Effect of the client drop in training phase 2

When training of phase 1 is finished, the clients send their data representations to the server. In real world scenarios, however, not all the clients participated in training phase 1 are able to take part in training phase 2. Specifically, in training phase 2, the server may not successfully receive the representation vectors from some clients who participated in training phase 1, because the representation vectors might be corrupted or lost during transmissions or some clients might not be able to transmit their representations. In this case, our assumption that the contrastive server mini-batches are (closely) class-balanced might not be valid anymore. In this appendix, we investigate its impact on the performance of our proposed MFCL scheme with BN, GN, and PN. To model that the representations of some clients are not successfully received by the server in training phase 2, we

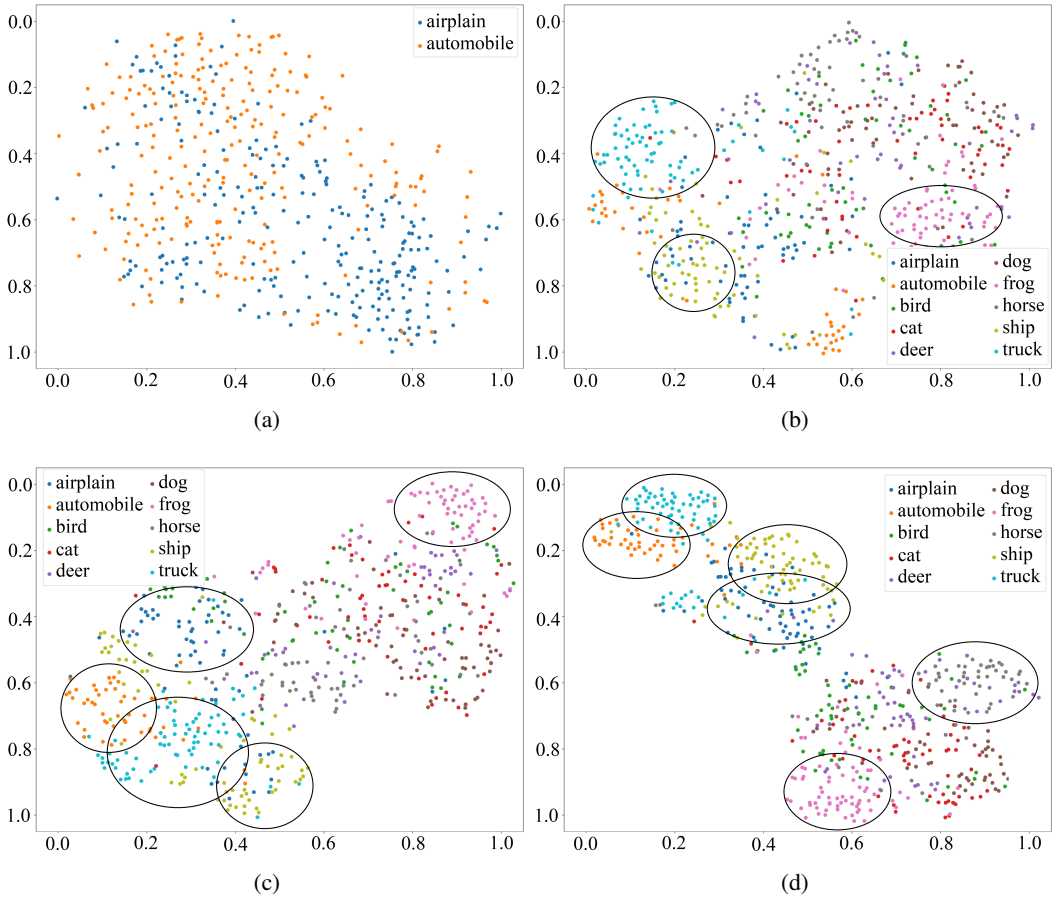


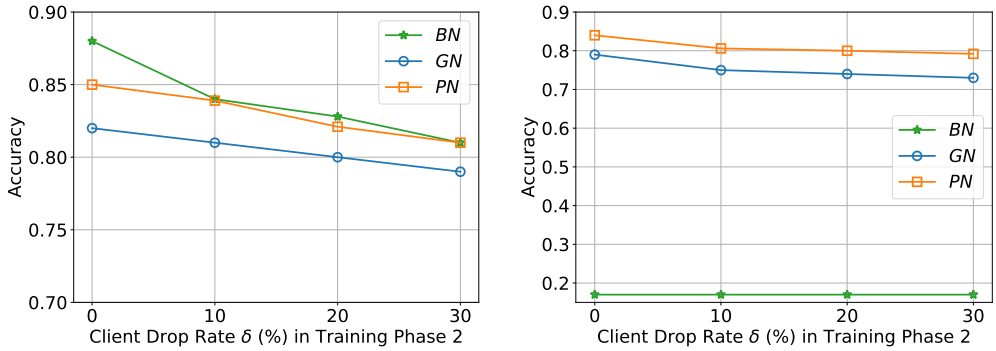
Figure 6: t-SNE visualization of the raw input data at a single client and t-SNE visualization of the clusters learned from the representation vectors at the server in the proposed MFCL scheme. CIFAR-10 is trained in simulation scenario Sc2, where each client has only 2 classes with $M = 10$. (a) t-SNE visualization of raw data in a single client. In the other three figures, t-SNE visualizations of the clusters learned at the server by MFCL after the projection layer are shown when the normalization technique used in the sensor module is (b) BN, (c) GN, and (d) PN.

introduce the concept of the percentage δ (%) of the clients who are dropped in training phase 2 (i.e., the clients whose representation vectors are not successfully received by the server in training phase 2).

In Figure 7 (a), for the scenario of homogeneous and CB data (Sc10), we train our MCFL with BN, GN, and PN on the CIFAR-10 data set when $M = 10$ clients have participated in training phase 1. Then, in training phase 2, the representation vectors from δ (%) clients are not received by the server. For a severe heterogeneous and CIB case (Sc2), we repeated the same experiments in Figure 7 (b). From Figures 7 (a) and 7 (b), we can see that both PN and GN have similar behavior when some clients are dropped in training phase 2. For the homogeneous and CB scenario (Sc10), the gap between BN and our proposed PN decreases as more clients are dropped in training phase 2. Meanwhile, for the severe heterogeneous and CIB case (Sc2), our proposed PN continues to be the best choice even when more and more clients are dropped in training phase 2.

D.3 PN IN OTHER SELF-SUPERVISED FL SCHEMES

In this appendix, we check if our proposed PN is still effective in other semi-supervised FL setting (other than our proposed MFCL). To this end, we apply our proposed PN to FedSimCLR Luo et al. (2021), which is a self-supervised training algorithm based on a contrastive loss in the FL setting.



(a) (b)

Figure 7: The classification accuracy of the proposed MFCL scheme with BN, GN, and PN, when the client drop rate is δ % in training phase 2. The number of clients participating in training phase 1 is $M = 10$. (a) Sc10 is considered, which is the scenario of homogeneous and CB data. (b) Sc2 is considered, which is a severe case of data heterogeneous clients with CIB data.

Table 7: Simulation results of applying PN in FedSimCLR in comparison with MFCL. In all simulation scenarios, the number of clients is $M = 10$ and the training data set is CIFAR-10. In all simulation scenarios for MFCL, BN is used at the server.

	BN		**GN**		**PN**	
Sc	FedSimCLR	MFCL	FedSimCLR	MFCL	FedSimCLR	MFCL
2	0.73(±0.02)	0.17(±0.05)	0.74(±0.03)	0.79(±0.03)	0.75(±0.05)	0.84(±0.02)
10	0.82(±0.01)	0.88(±0.02)	0.82(±0.04)	0.83(±0.01)	0.84(±0.02)	0.85(±0.01)

The model is trained on CIFAR-10 with $M = 10$. After the first two Conv2D layers in FedSimCLR, the PN is used in lieu of BN. The simulation results for Sc2 and Sc10 are presented in Table 7. We can see that PN improves the performance even in the FedSimCLR framework as well as in our proposed MFCL framework. Another observation is that our proposed MFCL with PN always outperforms FedSimCLR whether the clients have homogeneous and CB data or heterogeneous and CIB data.

D.4 Communication burden with MFCL

The proposed MFCL is the first modular self-supervised learning framework in FL. A major novelty is that the clients only train a shallow sensor module federally to learn the low-level features without the bias of labeling, clustering, or CIB data thanks to our proposed PN. This is in contrast to the traditional E2E training of FL, in which a deep and wide model is federally trained across the clients, which requires sending a huge amount of model parameters (or gradients) the server in each FL round, In our MFCL, the number of FL rounds required to train such low-level features across the clients is much less than the traditional E2E training in the FL setting. Therefore, the communication burden in training phase 1 is far lighter than in the E2E training in the FL setting. Although our proposed MFCL involves two stage training (rather than one as in E2E training), the overall communication overhead is much lower. Specifically, even including the burden of transmitting the representation vectors from the clients to the server in training phase 2, our proposed MFCL still has a ten times lighter total communication burden compared to the traditional E2E training with the same number of clients (e.g., $M = 10$) and the same amount of training data (e.g., 50,000 training samples in total).

In Table 8, as an example of traditional E2E self-supervised training in the FL setting, we compared the communication burden of MFCL with FedSimCLR, which trains ResNet-18 federally across the clients. FedSimCLR needs 800 FL rounds to achieve the reported accuracy in Table 2, while MFCL with PN needs only 15 FL rounds for training phase 1. In Table 8, we use the single precision format

Table 8: Simulation results of applying PN to FedSimCLR in comparison with MFCL.

Communication burden		**Method**	
		FedSimCLR	**MFCL**
In training phase 1	Parameters per FL round per client	11×10^6	10×10^4
	Transmitted bits per FL round per client	20×10^8	20×10^6
	Total transmitted bits per client	16×10^{11}	30×10^7
In training phase 2	Size of the representation vector per client	0	40×10^5
	Transmitted bits of representation vectors per client	0	30×10^{10}
In total	Phase 1 and 2 for $M = 10$ clients	16×10^{12}	30×10^{11}

(32 bits) with channel code (i.e., error correction code) rate $\frac{1}{3}$ to transmit the model parameters in training phase 1 and representation vectors in training phase 2. The size of one representation vector in the sixth row of Table 8 is equal to (number of data samples per client) $\times$ (size of each element in the representation vector), where the number of data samples per client is 5000 and the size of each element in the representation vector is $32 \times 32 \times 64$. Clearly, with more data samples in the clients, the communication burden of MFCL would increase linearly. However, with more data, we also need larger models to train FedSimCLR, which again leads to higher communication burden for FedSimCLR. Overall, the communication burden of the proposed MFCL is much lower than that of the traditional E2E training of FL.