APPENDIX

A COMPARISON OF SELF-SUPERVISED LEARNING FRAMEWORKS

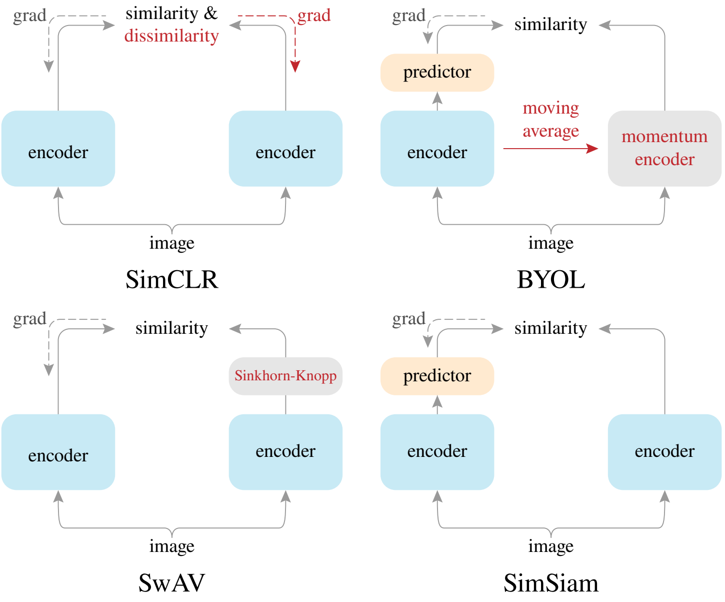
We compare state-of-the-art self-supervised learning frameworks (`SimCLR`, `SwAV`, `BYOL`) with `SimSiam` (Chen & He, 2020) in light of federated learning.

We choose `SimSiam` (Chen & He, 2020) because it requires a much smaller batch to perform normally. In the centralized setting, for each method to reach an accuracy level similar to that of `SimSiam`, a much larger batch size is necessary. Table 3 adopted from (Chen & He, 2020) provides a brief comparison between all listed self-supervised learning frameworks.

method	batch size	negative pairs	momentum encoder	100 ep	200 ep	400 ep	800 ep
SimCLR (repro.+)	4096	✓		66.5	68.3	69.8	70.4
BYOL (repro.)	4096		✓	66.5	**70.6**	**73.2**	**74.3**
SwAV (repro.+)	4096			66.5	69.1	70.7	71.8
SimSiam	**256**			**68.1**	70.0	70.8	71.3

Table 3: (Chen & He, 2020) **Comparisons on ImageNet linear classification**. All are based on **ResNet-50** pre-trained with **two 224×224 views** in a centralized setting. Evaluation is on a single crop. "repro." denotes reproduction conducted by authors of SimSiam (Chen & He, 2020), and "+" denotes *improved* reproduction v.s. original papers.

Another reason we prefer `SimSiam` (Chen & He, 2020) as the basic framework to build `SSFL` is that the design of `SimSiam` simplifies all other baselines and also obtains a relatively higher accuracy. Figure 7 abstracts these methods. The "encoder" contains all layers that can be shared between both branches (e.g., backbone, projection MLP (Chen et al., 2020), prototypes (Caron et al., 2021)). The components in red are those missing in `SimSiam`.



SimCLR

BYOL

SwAV

SimSiam

Figure 7: (Chen & He, 2020) **Comparison on Siamese architectures**. The encoder includes all layers that can be shared between both branches. The dashed lines indicate the gradient propagation flow. In `BYOL`, `SwAV`, and `SimSiam`, the lack of a dashed line implies stop-gradient, and their symmetrization is not illustrated for simplicity. The components in red are those missing in SimSiam.

SimCLR (Chen et al., 2020). `SimCLR` relies on negative samples ("dissimilarity") to prevent collapsing. `SimSiam` can be thought of as "`SimCLR` without negatives". In every mini-batch, for any image, one augmented view of the same image is considered to be its positive sample, and the remaining augmented views of different images are considered to be its negative samples. A contrastive loss term is calculated to push positive samples together and negative samples away.

SwAV(Caron et al., 2021). `SimSiam` is conceptually analogous to "`SwAV` without online clustering". SimSiam encourages the features of the two augmented views of the same image to be similar, while `SwAV` encourages features of the two augmented views of the same image to belong to the same cluster. An additional Sinkhorn-Knopp (SK) transform (Cuturi, 2013) is required for online clustering of `SwAV`. The authors of `SimSiam` (Chen & He, 2020) build up the connection between `SimSiam` and `SwAV` by recasting a few components in `SwAV`. (i) The shared prototype layer in `SwAV` can be absorbed into the Siamese encoder. (ii) The prototypes were weight-normalized outside of gradient propagation in (Caron et al., 2021); the authors of `SimSiam` instead implement by full gradient computation (Salimans & Kingma, 2016). (iii) The similarity function in `SwAV` is cross-entropy. With these abstractions, a highly simplified `SwAV` illustration is shown in Figure 7.

BYOL (Grill et al., 2020). `SimSiam` can be thought of as "`BYOL` without the momentum encoder", subject to many implementation differences. Briefly, in `BYOL`, one head of the Siamese architecture used in `SimSiam` is replaced by the exponential moving average of the encoder. As the momentum encoder has an identical architecture to that of the encoder, the introduction of an additional momentum encoder doubles the memory cost of the model.

SSL's recent success is the inductive bias that ensures a good representation encoder remains consistent under different perturbations of the input (i.e. consistency regularization). The perturbations can be either domain-specific data augmentation (e.g. random flipping in the image domain) (Berthelot et al., b; Laine & Aila; Sajjadi et al.; Berthelot et al., a; Hu et al.), drop out (Sajjadi et al.), random max pooling (Sajjadi et al.), or an adversarial transformation (Miyato et al.). With this idea, a consistency loss $\mathcal{L}$ is defined to measure the quality of the representations without any annotations.

B FORMULATION AND PSEUDO CODE FOR ALGORITHMS UNDER SSFL FRAMEWORK

Inspired by recent advances in personalized FL and self-supervised learning, we innovate several representative algorithms under `SSFL` framework. For each algorithm, we present its mathematical formulation and its pseudo code.

B.1 PER-SSFL

For `Per-SSFL`, as the formulation and algorithm have already been presented in Equation 4 and Algorithm 2, we provide a PyTorch style pseudo code in Algorithm 3 for additional clarity.

Algorithm 3: Per-SSFL PyTorch Style Pseudo Code

```
# F: global encoder
# H: global predictor
# f: local encoder
# h: local predictor

for x in loader:  # load a mini-batch x with n samples
    x1, x2 = aug(x), aug(x)  # random augmentation
    Z1, Z2 = F(x1), F(x2)  # global projections, n-by-d
    P1, P2 = H(Z1), H(Z2)  # global predictions, n-by-d

    L = D(P1, Z2) / 2 + D(P2, Z1) / 2  # global loss

    L.backward()  # back-propagate
    update(F, H)  # SGD update global model

    z1, z2 = f(x1), f(x2)  # local projections, n-by-d
    p1, p2 = h(z1), h(z2)  # local predictions, n-by-d

    l = D(p1, z2) / 2 + D(p2, z1) / 2  # local loss

    # distance between local and global representations
    l = l + λ * (D(p1, P1) + D(p1, P2) + D(p2, P1) + D(p2, P2)) / 4

    l.backward()  # back-propagate
    update(f, h)  # SGD update local model

def D(p, z):  # negative cosine similarity
    z = z.detach()  # stop gradient

    p = normalize(p, dim=1)  # l2-normalize
    z = normalize(z, dim=1)  # l2-normalize
    return -(p * z).sum(dim=1).mean()
```

B.2 Personalized SSFL with Local Adaptation (FedAvg-LA)

`FedAvg-LA` apply `FedAvg` (Brendan McMahan et al., 2016) on the `SimSiam` loss $\mathcal{L}_{\text{SS}}$ for each client to obtain a global model. We perform one step of SGD on the clients' local data for local adaption. The objective is defined in Equation 7, and the algorithm is provided in Algorithm 4.

$$\min_{\Theta,\mathcal{H}} \sum_{i=1}^{n} \frac{|D_k|}{|D|} \mathbb{E}_{\substack{\mathcal{T} \\ x\sim X_i}} \left[\| f_\Theta(\mathcal{T}(x)) - \mathcal{H}_x \|_2^2 \right] \tag{7}$$

Algorithm 4: FedAvg-LA

```
input :$K, T, \lambda, \Theta^{(0)}, \{\theta_i^{(0)}\}_{k\in[K]}$, $s$: number of local iteration, $\beta$: learning rate
for $t = 0, \dots, T-1$ do
    Server randomly selects a subset of devices $S^{(t)}$
    Server sends the current global model $\Theta^{(t)}$ to $S^{(t)}$
    for device $k \in S^{(t)}$ in parallel do
        Sample mini-batch $B_k$ from local dataset $D_k$, and do $s$ local iterations
        /* Optimize the global parameter Θ locally                  */
        $Z_1, Z_2 \leftarrow f_{\Theta^{(t)}}(\mathcal{T}(B_k)), f_{\Theta^{(t)}}(\mathcal{T}(B_k))$
        $P_1, P_2 \leftarrow h_{\Theta^{(t)}}(Z_1), h_{\Theta^{(t)}}(Z_2)$
        $\Theta_k^{(t)} \leftarrow \Theta^{(t)} - \beta \nabla_{\Theta^{(t)}} \frac{\mathcal{D}(P_1,\widehat{Z_2})+\mathcal{D}(P_2,\widehat{Z_1})}{2}$, where $\widehat{\cdot}$ stands for stop-gradient    CLIENTSSLOPT
        Send $\Delta_k^{(t)} := \Theta_k^{(t)} - \Theta^{(t)}$ back to server
    $\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \sum_{k\in S^{(t)}} \frac{|D_k|}{|D|} \Delta_k^{(t)}$    SERVEROPT
return :$\{\theta_i\}_{i\in[n]}, \Theta^{(T)}$
```

B.3 PERSONALIZED SSFL WITH MAML-SSFL

`MAML-SSFL` is inspired by perFedAvg (Fallah et al.) and views the personalization on each devices as the inner loop of MAML (Finn et al.). It aims to learn an encoder that can be easily adapted to the clients' local distribution. During inference, we perform one step of SGD on the global model for personalization. The objective is defined in Equation 8, and the algorithm is provided in Algorithm 5.

$$\begin{aligned} \min_{\Theta,\mathcal{H}} \quad & \sum_{i=1}^{n} \frac{|D_k|}{|D|} \mathbb{E}_{\substack{\mathcal{T} \\ x \sim X_i}} \left[\| f_{\Theta'}(\mathcal{T}(x)) - \mathcal{H}_x \|_2^2 \right] \\ \text{s.t.} \quad & \Theta' = \Theta - \nabla_\Theta \sum_{i=1}^{n} \frac{|D_k|}{|D|} \mathbb{E}_{\substack{\mathcal{T} \\ x \sim X_i}} \left[\| f_{\Theta}(\mathcal{T}(x)) - \mathcal{H}_x \|_2^2 \right] \end{aligned} \tag{8}$$

Algorithm 5: MAML-SSFL

```
input : $K, T, \lambda, \Theta^{(0)}, \{\theta_i^{(0)}\}_{k\in[K]}$, $s$: number of local iteration, $\beta$: learning rate, $M$
for $t = 0, \dots, T-1$ do
    Server randomly selects a subset of devices $S^{(t)}$
    Server sends the current global model $\Theta^{(t)}$ to $S^{(t)}$
    for device $k \in S^{(t)}$ in parallel do
        Sample mini-batch $B_k, B'_k$ from local dataset $D_k$, and do $s$ local iterations
        /* Inner loop update */
        $\Theta_k'^{(t)} \leftarrow \Theta^{(t)}$
        for $m = 0, \dots, M-1$ do
            $Z'_1, Z'_2 \leftarrow f_{\Theta'^{(t)}}(\mathcal{T}(B'_k)), f_{\Theta'^{(t)}}(\mathcal{T}(B'_k))$
            $P'_1, P'_2 \leftarrow h_{\Theta'^{(t)}}(Z'_1), h_{\Theta'^{(t)}}(Z'_2)$
            $\Theta_k'^{(t)} \leftarrow \Theta_k'^{(t)} - \beta \nabla_{\Theta_k'^{(t)}} \frac{\mathcal{D}(P'_1, \widehat{Z'_2}) + \mathcal{D}(P'_2, \widehat{Z'_1})}{2}$, where $\widehat{\cdot}$ stands for stop-gradient
        /* Outer loop update */
        $Z_1, Z_2 \leftarrow f_{\Theta'^{(t)}}(\mathcal{T}(B_k)), f_{\Theta'^{(t)}}(\mathcal{T}(B_k))$
        $P_1, P_2 \leftarrow h_{\Theta'^{(t)}}(Z_1), h_{\Theta'^{(t)}}(Z_2)$
        $\Theta_k^{(t)} \leftarrow \Theta^{(t)} - \beta \nabla_{\Theta^{(t)}} \frac{\mathcal{D}(P_1, \widehat{Z_2}) + \mathcal{D}(P_2, \widehat{Z_1})}{2}$        CLIENTSSLOPT
        Send $\Delta_k^{(t)} := \Theta_k^{(t)} - \Theta^{(t)}$ back to server
    $\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \sum_{k\in S^{(t)}} \frac{|D_k|}{|D|} \Delta_k^{(t)}$        SERVEROPT
return : $\{\theta_i\}_{i\in[n]}, \Theta^{(T)}$
```

B.4 PERSONALIZED SSFL WITH BILEVEL-SSFL

Inspired by Ditto (Li et al., 2021), `BiLevel-SSFL` learns personalized encoders on each client by restricting the parameters of all personalized encoders to be close to a global encoder independently learned by weighted aggregation. The objective is defined in Equation 9, and the algorithm is provided in Algorithm 6.

$$\begin{aligned} \min_{\theta_k, \eta_k} \quad & \mathbb{E}_{\substack{\mathcal{T} \\ x \sim X_k}} \left[\| f_{\theta_k}(\mathcal{T}(x)) - \eta_{k,x} \|_2^2 + \frac{\lambda}{2} \| \theta_k - \Theta_x^* \|_2^2 \right] \\ \text{s.t.} \quad & \Theta^*, \mathcal{H}^* \in \arg\min_{\Theta, \mathcal{H}} \sum_{i=1}^{n} \frac{|D_k|}{|D|} \mathbb{E}_{\substack{\mathcal{T} \\ x \sim X_i}} \left[\| f_{\Theta}(\mathcal{T}(x)) - \mathcal{H}_x \|_2^2 \right] \end{aligned} \tag{9}$$

Algorithm 6: BiLevel-SSFL

```
input  : $K, T, \lambda, \Theta^{(0)}, \{\theta_i^{(0)}\}_{k \in [K]}$, $s$: number of local iteration, $\beta$: learning rate
for $t = 0, \dots, T-1$ do
    Server randomly selects a subset of devices $S^{(t)}$
    Server sends the current global model $\Theta^{(t)}$ to $S^{(t)}$
    for device $k \in S^{(t)}$ in parallel do
        Sample mini-batch $B_k$ from local dataset $D_k$, and do $s$ local iterations
        /* Optimize the global parameter $\Theta$ locally                    */
        $Z_1, Z_2 \leftarrow f_{\Theta^{(t)}}(\mathcal{T}(B_k)), f_{\Theta^{(t)}}(\mathcal{T}(B_k))$
        $P_1, P_2 \leftarrow h_{\Theta^{(t)}}(Z_1), h_{\Theta^{(t)}}(Z_2)$
        $\Theta_k^{(t)} \leftarrow \Theta^{(t)} - \beta \nabla_{\Theta^{(t)}} \frac{\mathcal{D}(P_1, \widehat{Z_2}) + \mathcal{D}(P_2, \widehat{Z_1})}{2}$, where $\widehat{\cdot}$ stands for stop-gradient
        /* Optimize the local parameter $\theta_k$                           */
        $z_1, z_2 \leftarrow f_{\theta_k}(\mathcal{T}(B_k)), f_{\theta_k}(\mathcal{T}(B_k))$
        $p_1, p_2 \leftarrow h_{\theta_k}(z_1), h_{\theta_k}(z_2)$
        $\theta_k \leftarrow \theta_k - \beta \nabla_{\theta_k} \left( \frac{\mathcal{D}(p1, \widehat{z_2}) + \mathcal{D}(p2, \widehat{z_1})}{2} + \lambda \left\| \Theta^{(t)} - \theta_k \right\|_2^2 \right)$    CLIENTSSLOPT
        Send $\Delta_k^{(t)} := \Theta_k^{(t)} - \Theta^{(t)}$ back to server
    $\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \sum_{k \in S^{(t)}} \frac{|D_k|}{|D|} \Delta_k^{(t)}$    SERVEROPT
return : $\{\theta_i\}_{i \in [n]}, \Theta^{(T)}$
```

C DISTRIBUTED TRAINING SYSTEM FOR SSFL

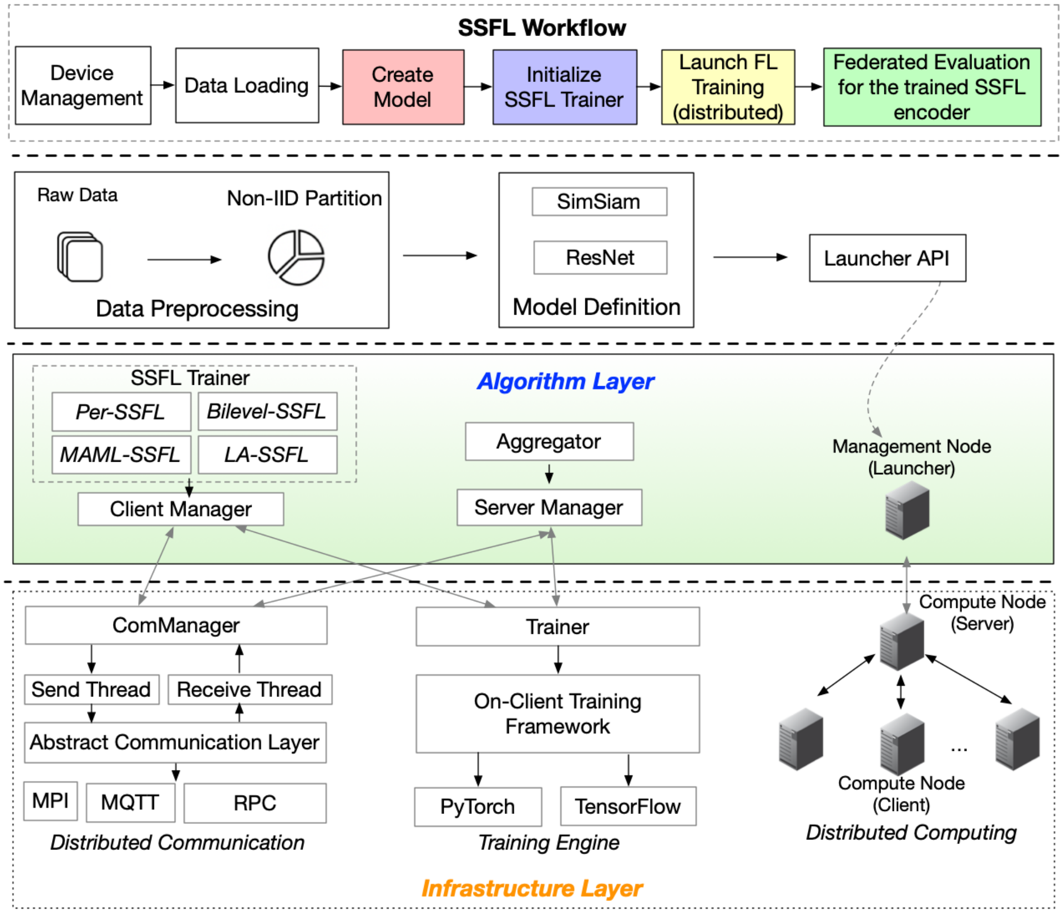


Figure 8: Distributed Training System for `SSFL` framework

We develop a distributed training system for our `SSFL` framework which contains three layers. In the infrastructure layer, communication backends such as MPI are supported to facilitate the distributed computing. We abstract the communication as `ComManager` to simplify the message passing between the client and the server. `Trainer` reuses APIs from PyTorch to handle the model optimizations such as forward propagation, loss function, and back propagation. In the algorithm layer, `Client Manager` and `Server Manager` are the entry points of the client and the server, respectively. The client managers incorporates various `SSFL` trainers, including `Per-SSFL`, `MAML-SSFL`, `BiLevel-SSFL`, and `LA-SSFL`. The server handles the model aggregation using `Aggregator`. We design simplified APIs for all of these modules. With the abstraction of the infrastructure and algorithm layers, developers can begin FL training by developing a workflow script that integrates all modules (as the "SSFL workflow" block shown in the figure). Overall, we found that this distributed training system accelerates our research by supporting parallel training, larger batch sizes, and easy-to-customize APIs, which cannot be achieved by a simple single-process simulation.

D EXPERIMENTAL RESULTS ON GLD-23K DATASET

We also evaluate the performance of SSFL on GLD-23K dataset. We use 30% of the original local training dataset as the local test dataset and filter out those clients that have a number of samples less than 100. Due to the natural non-I.I.D.ness of GLD-23K dataset, we only evaluate the Per-SSFL framework. The results are summarized in Table 4. *Note: we plan to further explore more datasets and run more experiments; thus we may report more results during the rebuttal phase.*

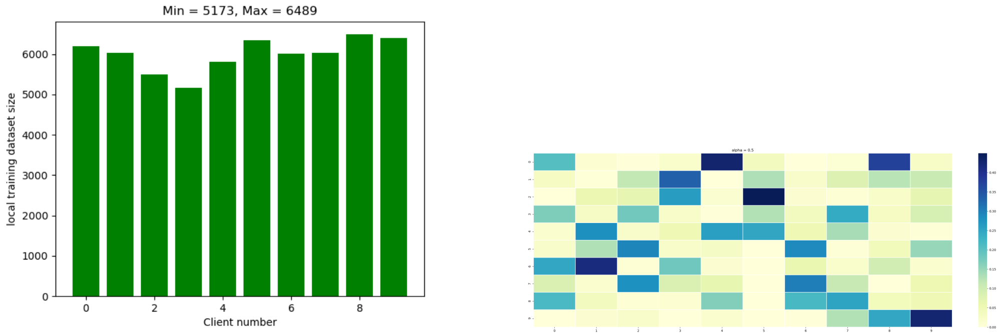
Table 4: Evaluation Accuracy for Various Per-SSFL Methods.

Method	KNN Indicator	Evaluation
`LA-SSFL`	0.6011	0.4112
`MAML-SSFL`	0.6237	0.4365
`BiLevel-SSFL`	0.6195	0.4233
`Per-SSFL`	0.6371	0.4467

*Note: the accuracy on supervised federated training using FedAvg is around 47%

E EXTRA EXPERIMENTAL RESULTS AND DETAILS

E.1 VISUALIZATION OF NON-I.I.D. DATASET



(a) Sample Number Distribution

(b) Label Distribution (deeper color stands for more samples

Figure 9: Visualization for non-I.I.D. synthesized using CIFAR-10

E.2 HYPER-PARAMETERS

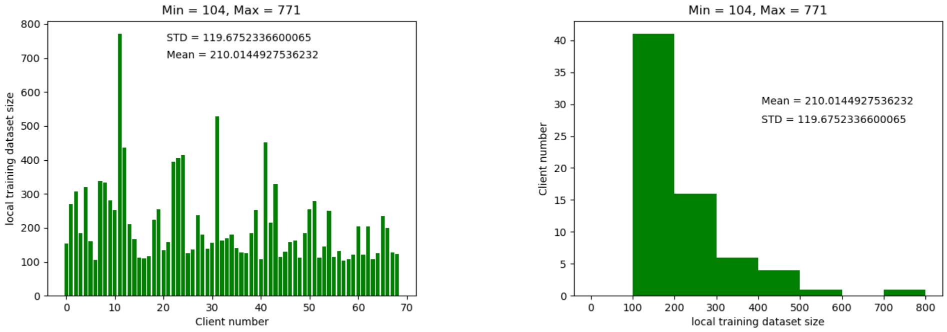
Table 5: Hyper-parameters for Section 5.2

Method	Learning Rate	Local Optimizer
SSFL (I.I.D)	0.1	SGD with Momemtum (0.9)
SSFL (non-I.I.D)	0.1	SGD with Momemtum (0.9)

Table 6: Hyper-parameters for Section 5.4.2

Method	Learning Rate	λ	Local Optimizer
Per-SSFL ($\alpha = 0.1$)	0.03	0.1	SGD with Momemtum (0.9)
Per-SSFL ($\alpha = 0.5$)	0.03	0.1	SGD with Momemtum (0.9)

All experiments set the local epoch number as 1, round number as 800, batch size as 256 (batch size 32 with 8 gradient accumulation steps).



(a) Sample Number Distribution (X-axis: Client Index; Y-axis: Number of Training Samples)

(b) Sample Number Distribution (X-axis: Number of Training Samples; Y-axis: Number of Clients)

Figure 10: Visualization for non-I.I.D. on GLD-23K

Table 7: Hyper-parameters for experimental results in Section 5.3

Method	Learning Rate	λ	Local Optimizer
LA-SSFL	0.1	1	SGD with Momemtum (0.9)
MAML-SSFL	0.03	1	SGD with Momemtum (0.9)
BiLevel-SSFL	0.1	1	SGD with Momemtum (0.9)
Per-SSFL	0.03	0.1	SGD with Momemtum (0.9)

F DISCUSSION

To overcome the large batch size requirement in `SSFL` and practical FL edge training, one direction is to use efficient DNN models such as EfficientNet (Tan & Le, 2019) and MobileNet (Howard et al., 2017) as the backbone of SimSiam. However, we tested its performance under our framework and found that the performance downgrades to a level of accuracy that is not useful (less than 60%). A recent work in centralized self-supervised learning mitigates these models' accuracy gap by knowledge distillation, which works in a centralized setting but is still not friendly to FL since KD requires additional resources for the teacher model. In practice, we can also explore batch size 1 training (Cai et al., 2020) at the edge, which dramatically reduces the memory cost with additional training time.