# MOCOSFL: ENABLING CROSS-CLIENT COLLABORATIVE SELF-SUPERVISED LEARNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Existing collaborative self-supervised learning (SSL) schemes are not suitable for cross-client applications because of their expensive computation and large local data requirements. To address these issues, we propose MocoSFL, a collaborative SSL framework based on Split Federated Learning (SFL) and Momentum Contrast (MoCo). In MocoSFL, the large backbone model is split into a small client-side model and a large server-side model, and only the small client-side model is processed locally on the client's local devices. MocoSFL has three key components: (i) vector concatenation which enables the use of small batch size and reduces computation and memory requirements by orders of magnitude; (ii) feature sharing that helps achieve high accuracy regardless of the quality and volume of local data; (iii) frequent synchronization that helps achieve better non-IID performance because of smaller local model divergence. For a 1,000-client case with non-IID data (each client only has data from 2 random classes of CIFAR-10), MocoSFL can achieve over 84% accuracy with ResNet-18 model. Next we present TAResSFL module that significantly improves the resistance to privacy threats and communication overhead with small sacrifice in accuracy for a MocoSFL system. On a Raspberry Pi 4B device, the MocoSFL-based scheme requires less than 1MB of memory and less than 40MB of communication, and consumes less than 5W power. Thus, compared to the state-of-the-art FL-based approach, MocoSFL has significant advantages in both accuracy and practicality for cross-client applications.

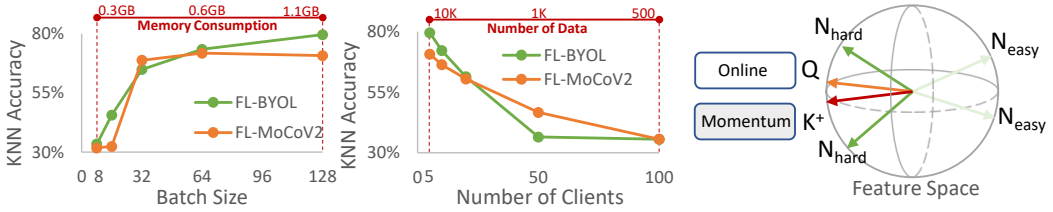
## 1 INTRODUCTION

Collaborative learning schemes have become increasingly popular, as clients can train their own local models without sharing their private local data. Current collaborative learning applications mostly focus on supervised learning applications where labels are available (Hard et al., 2018; Roth et al., 2020). However, availability of fully-labeled data may not be practical since labeling requires expertise and can be difficult to execute, especially for the average client.

Federated learning (FL) (McMahan et al., 2017) is the most popular collaborative learning framework. One representative algorithm is **"FedAvg"**, where clients send their local copies of the model to the server and the server performs a weighted average operation (weight depends on the amount of data) to get a new global model. FL has achieved great success in supervised learning, and has been used successfully in a wide range of applications, such as next word prediction McMahan et al. (2017), visual object detection for safety Liu et al. (2020), recommendation Wu et al. (2022a;b), graph-based analysis Chen et al. (2022); Wu et al. (2022c), etc.

For collaborative learning on unlabeled data, prior works (Zhang et al., 2020; Zhuang et al., 2021; 2022) combine FL scheme with classic self-supervised learning (SSL) methods such as BYOL (Grill et al., 2020) and Moco (He et al., 2020). These methods can all achieve good performance when clients' data is Independent and Identically Distributed (IID) but suffer from poor performance in non-IID cases. Recently, Zhuang et al. (2022) mitigated the non-IID performance drop with divergence-aware aggregation technique and provided state-of-the-art (SoTA) accuracy performance using a combination of FL and different SSL methods.

However, these SoTA FL-SSL schemes are not practical for cross-client applications. **First**, FL-SSL imposes a significant computation overhead and large memory requirement on clients. This is




(a) (b) (c)

Figure 1: Challenges in FL-SSL schemes. (a) A large batch size is necessary to achieve good performance (KNN validation accuracy (Wu et al., 2018)) (b) Accuracy reduces with an increasing number of clients since the amount of local data is now smaller. (c) Hard negative keys are essential for the success of contrastive learning.

because SSL requires a large backbone architecture (Chen et al., 2020a) together with a large batch size to ensure good performance. As shown in Fig. 1(a), accuracy drops dramatically when batch size is low for both FL-SSL methods. **Second**, FL-SSL schemes fail to maintain a good accuracy when the number of clients is large (cross-client cases), as shown in Fig. 1(b). For a dataset with fixed size, when the number of clients increases, data per client decreases, resulting in accuracy degradation. The drop in accuracy is mainly because of the failure to meet data size requirement in performing contrastive learning. Zhang et al. (2020); Wu et al. (2021) attempt to address this issue in FL-SSL with remote feature sharing. However, this introduces a high communication overhead due to client-to-client feature memory synchronization; for a 100-client system, it costs around 2.46GB per synchronization per client.

To solve these challenges, we propose MocoSFL, a scheme based on Split Federated Learning (SFL) Thapa et al. (2020) that incorporates the feature memory bank and momentum model designs of MoCo (He et al., 2020). We adopt the SFL scheme for three reasons: (i) SFL utilizes a smaller client-side model and so reduces the computation overhead and has lower memory consumption and model parameters; (ii) SFL's latent vector concatenation enables a large equivalent batch size for the centralized server-side model, making micro-batch training possible for clients and thus reducing client's local memory; (iii) When combined with MoCo's key-storing mechanism, SFL's shared server-side model enables effective feature sharing, which removes the requirement of large amount of local data and makes the scheme possible for cross-client applications. As a result, MocoSFL achieves good accuracy with ultra-low memory requirements and computation overhead, and can support a very large number of clients. MocoSFL shows better non-IID performance since local model divergence is smaller. However, the use of SFL brings extra communication overhead as well as data privacy concerns. Thus, we present target-aware ResSFL (TAResSFL) module as an effective solution to mitigate these issues with small accuracy drop.

Our main contributions are:

- We identify two major challenges in deriving high accuracy in FL-SSL schemes for cross-client applications. These are availability of a large amount of data that is required for contrastive learning and the ability to process them in clients who may not have sufficient hardware resources.

- We propose MocoSFL, an SFL-based scheme to address the two challenges. MocoSFL uses a small client-side model, latent vector concatenation, and feature sharing. For cross-client case, MocoSFL is the only viable and practical solution. For cross-silo case, MocoSFL can achieve even better performance than SoTA FL-SSL schemes under non-IID setting because of smaller model divergence.

- To address communication overhead and privacy issues that are inherent to SFL-based schemes, we propose target-domain-aware ResSFL (TAResSFL) that effectively reduces the communication cost and mitigates model inversion attacks.

# 2 BACKGROUND

## 2.1 SELF-SUPERVISED LEARNING

To learn from unlabeled data, SSL schemes based on contrastive learning such as SimCLR Chen et al. (2020a), BYOL Grill et al. (2020), Simsiam Chen and He (2021) and MoCo He et al. (2020) have achieved great performance on popular benchmarks. Unlike other schemes (BYOL, SimCLR, etc.) that use other samples in the current data-batch as negative keys, MoCo uses previously computed positive keys as negative keys, and stores current positive keys in the feature memory for future iterations. The key-storing mechanism results in a relatively smaller batch size requirement that is beneficial for reducing device memory and also makes it easy for feature sharing implementations for our proposed scheme. For the loss function, MoCo relies on InfoNCE loss Oord et al. (2018) as the contrastive mechanism to update its model parameters:

$$\mathcal{L}_{Q,K,N} = -log\frac{exp(Q \cdot K^{+}/\tau)}{exp(Q \cdot K^{+}/\tau) + \sum_{N \in M} exp(Q \cdot N/\tau)} \tag{1}$$

where query key $Q$ and positive key $K^{+}$ are the output vectors of server-side momentum model and the momentum model, respectively, obtained by processing two augmented views of the image. $N$ denotes all negative keys in the feature memory of size $M$. Importantly, the success of MoCo scheme highly depends on the "hardness" of its negative keys (Kalantidis et al., 2020; Robinson et al., 2020). The "hardness" of a negative key $N$ in the feature memory bank, can be determined by the similarity (inner-product) between $Q_t$ (at step $t$) and $N$; the smaller the similarity, the better the "hardness". We notice the "hardness" of negative key $N$ reduces quickly because model updates are in the direction of minimizing the InfoNCE loss in Eq. (1). As a result, MoCo adopts a slow-changing momentum model to produce consistent negative keys to add to the feature memory at the end of every training step and thereby maintains their hardness.

## 2.2 SPLIT FEDERATED LEARNING

Split Federated Learning (SFL) Thapa et al. (2020) is a recent collaborative learning scheme that focuses on high computation efficiency at the client side. It splits the original model architecture into two parts, the client-side model that contains all layers upto the "cut-layer" and the server-side model that contains the remaining layers. We distribute copies of client-side model $C_i$ to client-$i$'s local devices and instantiate the server-side model $S$ in a cloud server. To complete each training step, clients need to send the latent vectors (the output of client-side model) to the server, and the server processes latent vectors, computes the loss, performs backward propagation and returns the corresponding gradients to clients. Thapa et al. (2020) present two possible ways for server to process latent vectors sent by clients. In this paper, we use SFL-V1 where the server concatenates all clients' latent vectors and processes them altogether, which makes the equivalent batch size larger at the server and benefits contrastive learning. In contrast, in SFL-V2, client's latent vectors are processed sequentially in a first-come-first-serve manner and thus does not benefit from the "large batch". We provide details of SFL-V1 in Appendix A.2.

# 3 MOTIVATION

As mentioned in Section 1, the two challenges in extending FL-SSL to cross-client applications are high computing resource requirement and large data requirement.

## 3.1 HIGH COMPUTING RESOURCE REQUIREMENT

The first challenge is the computing resource requirement of training an SSL model locally. Using a compact backbone model may be accurate for supervised learning, but is not suitable for SSL as it has a much higher requirement on the model capacity. (Shi et al., 2021; Fang et al., 2021) show that compact architectures like Mobilenet-V3 (Howard et al., 2019), EfficientNet Tan and Le (2019) suffer from over 10% accuracy degradation compared to a larger ResNet-18 architecture, while an even larger ResNet-50 model has over 15% better accuracy compared to ResNet-18 on ImageNet dataset (Deng et al., 2009). This means memory requirement for training an SSL model with high

accuracy is very high. Using a smaller batch size reduces accuracy dramatically, as shown in Fig. 1(a), and is thus not an option. A FL-SSL scheme (ResNet-18 with a batch size of 128) costs 590.6 MFLOPs per image and over 1100MB of memory per client, which is not practical.

### 3.2 Large data requirement

The other major difficulty for FL-SSL to generalize to the cross-client case is the large data requirement. For cross-client applications, the amount of data available to each client can be very limited. For example, in a cross-silo medical application, a client can be a hospital with tons of data. In comparison, in a cross-client application, a client can be a patient who has limited amount of data.

The root of the problem lies in the difficulty to find hard negative samples when clients do not have enough local data. When the amount of data is larger, the chance for hard negative samples to be present becomes much higher. As a result, existing FL-SSL can only be successful for cross-silo applications where clients have large amount of data and can perform effective contrastive learning locally. As demonstrated in Fig. 1(b), we observe high accuracy when clients have 10K samples of data, while the accuracy drops quickly to around 30% when clients have only 500 samples.

Unfortunately, feature sharing Zhang et al. (2020); Wu et al. (2021) in FL-SSL schemes cannot solve the large data requirement. These schemes update the shared feature memory less frequently because of significant communication overhead with each update. Since clients must synchronize their local copies[1] of the shared feature memory each time a minor change happens, in Zhang et al. (2020), new negative keys are added to the feature memory only once per epoch. Unfortunately, they cannot maintain a good "hardness" through the entire epoch and thus feature sharing mechanism brings very limited ($\sim 1\%$) improvement in accuracy. Also, the fact that total synchronization overhead scales quadratically with the number of clients makes the method not practical for cross-client use case.

## 4 Method

### 4.1 Proposed MocoSFL

Our proposed MocoSFL is an innovative combination of SFL-V1 and MoCo-V2 (Chen et al., 2020b) as shown in Fig. 2. There are three key components. **First**, in each training step, the latent vectors sent by all clients are concatenated before being processed by the server-side model. This helps achieve a large equivalent batch size in order to support mini-batch training. **Second**, we use a shared feature memory which is updated by positive keys contributed by all clients in every training step. **Third**, we improve the non-IID performance by using a higher synchronization frequency. Next, we will elaborate on how these three components in the proposed MocoSFL address the two challenges in Section 4.2 and Section 4.3. Section 4.4 describes how MocoSFL achieves better non-IID performance and Section 4.5 addresses the privacy and communication issues of the proposed scheme.

### 4.2 Reduce Hardware Resource Requirement

Choice of SFL helps reduce computational overhead and memory consumption at the client-end because of the much smaller client-side model. For example, on a CIFAR-10 ResNet-18 model with a batch size of 128, a client-side model with 3 layers only costs 13.7% of the FLOPs compared to the entire model, and its memory cost is 227MB, merely one fourth of the entire model. Furthermore, we reduce the batch size to 1 (also known as "micro-batch"), to further reduce the memory consumption. The use of micro-batch in local model training is only possible thanks to the latent vector concatenation mechanism which basically aggregates latent vectors sent by all clients into a big batch before sending it to the server. In addition, in a micro-batch setting, we replace the batch normalization layer by group normalization Wu and He (2018) and weight standardization (Qiao et al., 2019) to gain better accuracy performance. In Fig. 3(b), we compare the computation and memory consumption of the proposed MocoSFL with the FL-SSL scheme. MocoSFL with cut-layer of 3 achieves $\sim 288\times$ reduction in memory consumption than FL-SSL and has 2%-10% higher accuracy. Details of accuracy evaluation are included in Section 5.1.

[1]Since clients must hold a local copy of the shared memory to calculate the loss locally.

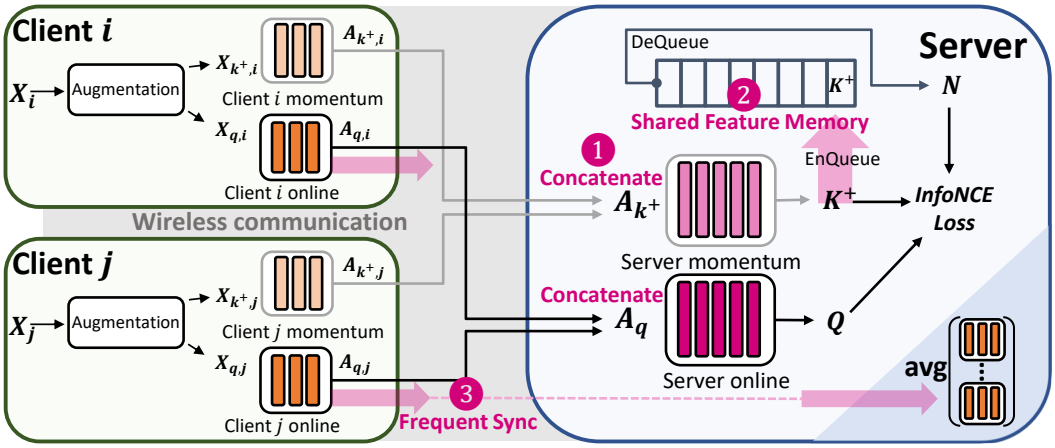



Figure 2: MocoSFL scheme. Three highlighted components are (1) latent vector concatenation, (2) shared feature memory, and (3) frequent synchronization.

### 4.3 Mitigate Large Data Requirement

As indicated by (Kalantidis et al., 2020; Robinson et al., 2020), the "hardness" of a negative key $N$ heavily depends on its similarity with the current query key $Q$, given that $N$ and $Q$ have different ground-truth labels. To evaluate the hardness of negative key $N_0$ residing in the feature memory, we use the similarity measure (inner-product) between $N_0$ and $Q_t$, a freshly calculated query key at time $t$. In FL-SSL with feature sharing, the negative key is only updated after a long period of time to reduce communication overhead. As a result, the hardness diminishes quickly. In contrast, MocoSFL frequently updates its feature memory to maintain a good hardness. At every training step, a freshly calculated positive key $K_+$ is added to the tail of the queue, and the oldest one is popped out.

However, frequent updates of feature memory is not enough to ensure a high level of hardness. We also find it is necessary to use a large batch size and a large feature memory. This finding agrees with the study in Bulat et al. (2021) and also explains the accuracy drop for a small batch size in FL-MocoV2 in Fig. 1(a). To illustrate this, we consider the total similarity measure at time $t$ and make the following assumptions: For the newest batch of negative keys $N_t = K_{t-1}$ of size $B$ in the feature memory at time $t$, we assume the similarity measure between $N_t$ and $Q_t$ is a constant $\eta$ for all $t$. We also assume similarity of every batch of negative keys in feature memory gets reduced by a constant factor $\gamma$ ($\gamma < 1$) after each model update to represent the degradation caused by model updates. Thus, for a freshly computed query $Q_t$, its total similarity measure with negative keys in the feature memory can be represented as:

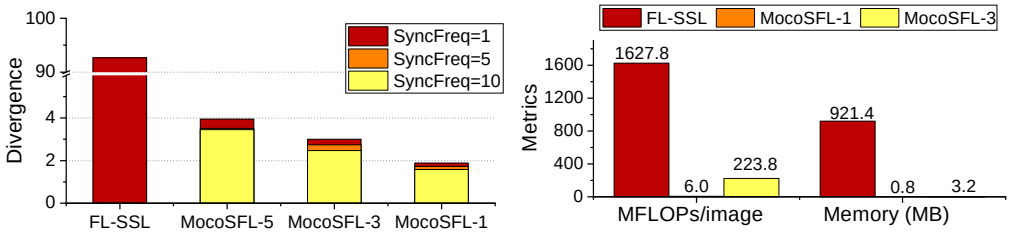
$$hardness = B\eta\gamma + B\eta\gamma^2 + ... + B\eta\gamma^{\lfloor M/B \rfloor} \tag{2}$$

$$= B\eta\gamma \times (\frac{1-\gamma^{\lfloor M/B \rfloor}}{1-\gamma}) \tag{3}$$

where $B$ is the batch size and $M$ is the feature memory size. We see that using a large batch size $B$ is beneficial as it helps bring more fresh negative keys and maintain better hardness. Also, using a larger feature memory (increasing $M$) can keep enough negative keys and contribute to a better total hardness. In the cross-client case, FL-SSL schemes can hardly meet these two requirements because of the small clients' memory. Nonetheless, MocoSFL can easily fulfill them because (1) latent vector concatenation enables a large equivalent batch size, and (2) feature memory hosted by the cloud server can be much larger.

### 4.4 Improving Non-IID Performance

We found that use of SFL results in fewer model parameters at the client side and hence smaller model divergence. Furthermore, introducing frequent synchronization in MocoSFL provides additional reduction in model divergence and greatly improves the non-IID performance. According to Zhang et al. (2020); Zhuang et al. (2021; 2022), the model divergence between two models is calculated




(a) (b)

Figure 3: (a) Proposed MocoSFL reduces model divergence. (b) Computation overhead (FLOP counts of inference per image of the online model at client-end) comparison between FL-SSL scheme and MocoSFL schemes. MocoSFL-$L$: client-side model has $L$ layers.

as the L2 norm of the weight difference. Following the same idea, the total divergence measure of a cross-client system can be measured as the average weight divergence of local models w.r.t. the global model during training:

$$divergence = \frac{1}{EN_C} \sum_{e=1}^{E} \sum_{i=1}^{N_C} \sum_{l=1}^{L} ||W_{e,l}^{i} - W_{e,l}^{*}||_2 \tag{4}$$

where $L$ denotes the number of layers in the client-side model, $E$ denotes the total number of synchronizations, $N_C$ denotes the number of clients, and $l, e, i$ are the respective indices for $L, E, N_C$. $W^*$ is the average of all client models $W^i$. MocoSFL reduces the model divergence with two orthogonal mechanisms. The first mechanism is the reduction of client-side model size, which directly results in a lower model divergence. As shown in Fig. 3(a), compared to FL-SSL scheme, MocoSFL has a much lower model divergence when the client-side model has less than 5 layers. The other mechanism is frequent model synchronization which helps reduce the model divergence. This is only possible in SFL because of communication overhead of sending weights in a smaller client-side model, is smaller. Fig. 3(a) also illustrates how model divergence further reduces as we increase the synchronization frequency.

### 4.5 Improving Privacy and Communication Overhead of MocoSFL

The proposed MocoSFL scheme is based on SFL and suffers from two issues – high overall communication overhead due to transmitting and receiving latent vectors and vulnerability to Model Inversion Attack (MIA) Fredrikson et al. (2015), in which the server can reconstruct clients' raw inputs from latent vectors, making clients' data privacy questionable. (We leave the details of its threat model and working mechanism in Appendix A.3) To address the privacy and communication issues in MocoSFL, we propose Target-Aware-ResSFL (TAResSFL). TAResSFL extends ResSFL Li et al. (2022) for self-supervised learning through: (1) target-data-aware self-supervised pre-training, and (2) freezing feature extractor during SFL training. TAResSFL also utilizes the bottleneck layer design from ResSFL to reduce the communication overhead.

In ResSFL Li et al. (2022), the server performs pretraining to build up the resistance to MIA using data from a different domain since it does not have access to clients' data. Next, the pretrained resistant client-side model is transferred to the clients and gets fine-tuned using SFL. TAResSFL improves the pretraining step by assuming that the server can get access to a small subset ($<1\%$) of training data, together with large amount of data from another domain, and perform pre-training using self-supervised learning. Such a pretrained client-side model has better transferability, and can thus stay frozen during SFL process, thereby avoiding the expensive fine-tuning. As shown in Fig. 4, we blend the source dataset $X_s$ with a tiny subset of target dataset $X_t$ during the feature extractor training. The attacker-aware training has the min-max form:

$$\min_{\boldsymbol{W}_C, \boldsymbol{W}_S} \max_{\boldsymbol{W}_G} \underbrace{\mathcal{L}(S(\boldsymbol{W}_S; C(\boldsymbol{W}_C; [\boldsymbol{x}_q, \boldsymbol{x}_{k+}]))}_{\text{Contrastive Loss}} + \lambda \underbrace{\mathbb{R}(G(\boldsymbol{W}_G; C(\boldsymbol{W}_C; \boldsymbol{x}_q)), \boldsymbol{x}_q)}_{\text{Inversion Score}} \tag{5}$$

where $\mathbb{R}$ in the inner maximization denotes a similarity measure, for which we use the structural similarity index (SSIM) score Zhao et al. (2016). The inner maximization is used to train the

simulated attack model $G$, whose function is reconstructing the activation back to a raw input state that is similar with ground-truth $X_q$. The outer minimization step goes in the direction of lowering contrastive loss, where the regularization term makes the model accurate as well as resistant to attack. These two steps are done alternatively to make the feature extractor resistant, and also be able to achieve good accuracy on the target dataset.

Fig. 4 presents the Target-aware ResSFL scheme. We use the resistant feature extractor to initialize client-side models during transfer step, as shown by the pink arrow in Fig. 4. Unlike ResSFL, here we freeze its parameter completely to maintain the resistance since any parameter change can cause resistance drop. However, freezing brings a noticeable accuracy drop of larger than 3%, even with the use of CIFAR-100 as source dataset if we perform pre-training by only using the source dataset (see Table 4). But if we blend the source data with a small portion of target data during pre-training, the accuracy can be greatly improved. Since the model accuracy validation is done by the server for monitoring purpose (Bhagoji et al., 2019), we believe that the server can separate out a small proportion of validation data to meet the target data availability assumption.

The freezing also greatly benefits hardware resource requirement since: (1) clients only need to transmit the latent vectors to the server and do not need to perform backward propagation using gradients from the server; (2) client-side model synchronization is not needed. As a result, MocoSFL with TAResSFL component achieves a $\sim 128\times$ (=5001.4/39.1) overall communication reduction compared to the original MocoSFL, as shown in Table 1. Here MocoSFL methods undergo 200 model synchronizations for 200 epochs while FL-SSL methods need 100 synchronizations (using the same setting in Zhuang et al. (2022)).

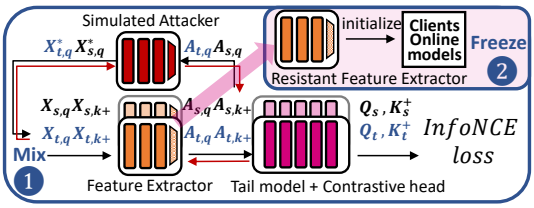



Figure 4: Target-aware ResSFL scheme: (1) Target domain data is used in pretraining; (2) Client-side model is frozen during training.

| Communication Overhead | Weights (MB) | Latent Vectors (MB) | Total (Relative (MB) Ratio) |
|---|---|---|---|
| FL-SSL methods Zhuang et al. (2022) | 8269.2 | 0.0 | 8269.2 (1.000x) |
| MocoSFL-1 | 1.4 | 5000 | 5001.4 (0.605x) |
| MocoSFL-3 | 57.9 | 5000 | 5057.9 (0.612x) |
| MocoSFL-1+TAResSFL | 0.0 | 39.1 | 39.1 (0.005x) |
| MocoSFL-3+TAResSFL | 0.0 | 39.1 | 39.1 (0.005x) |

Table 1: Communication overhead per client. FL-SSL: 100 times of synchronization; MocoSFL: 200 times of synchronization.

## 5 Experimental Result

**Experimental Setting.** We simulate the multi-client MocoSFL scheme on a Linux machine, where we use different CPU threads to simulate different clients and a single RTX-3090 GPU to simulate the cloud server. We use ResNet-18 (He et al., 2016) for the majority of the experiments to better compare with existing SoTA (Zhuang et al., 2022). We use CIFAR-10 as the main dataset and also present results on CIFAR-100 and ImageNet 12-class subset as in Li et al. (2021). For the IID case, we assume the entire dataset is divided randomly and equally among all clients. For non-IID experiments, we mainly consider the pathological (aka. class-wise) non-IID distribution as in McMahan et al. (2017); Zhuang et al. (2022) where we assign 2 classes of CIFAR-10/ImageNet-12 data or 20 classes of CIFAR-100 data randomly to each client. We perform MocoSFL training for a total of 200 epochs, using SGD as the optimizer with an initial learning rate of 0.06. For accuracy performance evaluation, we adopt similar linear probe methods as in Grill et al. (2020); Zhuang et al. (2022), where we train a new linear classifier on the outputs generated by the MocoSFL backbone model. We include details of hyper-parameter choices and evaluations in Appendix A.1.

### 5.1 Accuracy Performance

**Improved non-IID performance.** Fig. 5 shows how the increased synchronization frequency can significantly improve the non-IID accuracy. We present results for the cut-layer choices of 1 and 3 convolutional layers in the client-side model, represented by "MocoSFL-1" and "MocoSFL-3", respectively. We attribute the improved accuracy to the reduction in model divergence.

**Comparison with FL-SSL.** When synchronization frequency of the MocoSFL is set to 10 (per epoch) for the 5-client cases, MocoSFL achieves significantly better non-IID accuracy performance than Zhuang et al. (2022) on CIFAR-10 dataset due to lower model divergence (see Table 2). On CIFAR-100 dataset, with $N_C = 5$, we observe our method has lower accuracy than FL-SSL methods. We hypothesize that the performance of MocoSFL is more sensitive to the model complexity, and thus has limited performance for a more complex task like CIFAR-100. When model complexity is high enough, for instance, on a larger ResNet-50 model, accuracy of our method is ~4% higher than Zhuang et al. (2022) as shown in Appendix B.5. Furthermore, our methods outperform FL-SSL methods by a large margin in 20-client cases thanks to the feature sharing aspect.

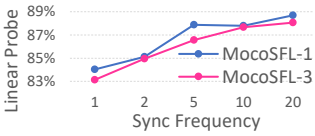



Figure 5: Effect of increasing synchronization

| Method | CIFAR-10 | | CIFAR-100 | |
|---|---|---|---|---|
| | $N_C = 5$ | $N_C = 20$ | $N_C = 5$ | $N_C = 20$ |
| FL-BYOL (Zhuang et al., 2022) | 83.34 | 75.77 | 61.78 | 52.78 |
| MocoSFL-1 (ours) | 87.81 | 85.84 | 58.78 | 57.80 |
| MocoSFL-3 (ours) | 87.29 | 85.32 | 57.70 | 57.52 |

Table 2: Non-IID performance comparison (linear probe accuracy)

**Cross-client Performance.** Our proposed MocoSFL can generalize from a cross-silo application (with upto 20 clients) to a cross-client application with 100, 200, and 1000 clients. Note that none of the previous FL-SSL methods can scale to such a large number of clients. For the hyper-parameter choices, we follow two principles introduced in Appendix A.1 – we let each client use a batch size of 1 and use the synchronization frequency of $f_S = (1000/N_C)$/epoch, and we set the client sampling ratio to $100/N_C$ to keep the same equivalent batch size at the server end. The results are shown in Table 3. Note that each client has only 50 data samples in the 1000-client case. MocoSFL's accuracy for IID case is high when $N_C$ increases from 100 to 1,000, though its accuracy drop by 1% for non-IID case. This small drop is because model divergence scales with number of clients as described in Section 4.4.

Table 3: MocoSFL cross-client accuracy performance (linear probe accuracy) of ResNet-18 model on CIFAR-10, CIFAR-100 and Imagenet-12 datasets with different number of clients $N_C$.

| Method | Dataset | IID | | | non-IID | | |
|---|---|---|---|---|---|---|---|
| | | $N_C = 100$ | $N_C = 200$ | $N_C = 1000$ | $N_C = 100$ | $N_C = 200$ | $N_C = 1000$ |
| MocoSFL-1 | CIFAR-10 | 87.29 | 87.38 | 87.51 | 87.71 | 87.39 | 86.46 |
| | CIFAR-100 | 58.91 | 59.15 | 58.85 | 59.22 | 58.90 | 56.75 |
| | ImageNet-12 | 92.02 | 91.73 | 91.76 | 92.24 | 91.44 | 91.28 |
| MocoSFL-3 | CIFAR-10 | 87.29 | 87.15 | 87.25 | 87.10 | 85.22 | 84.75 |
| | CIFAR-100 | 58.41 | 58.30 | 58.80 | 58.69 | 58.59 | 56.88 |
| | ImageNet-12 | 92.08 | 92.24 | 92.02 | 92.60 | 91.83 | 91.28 |

## 5.2 Privacy Evaluation

| Method | Metric | Target Data | | |
|---|---|---|---|---|
| | | **0.0%** | **0.5%** | **1.0%** |
| **MocoSFL-1** | Accuracy (%) | 81.14±0.47 | 80.78±1.34 | 79.96±2.96 |
| | Attack MSE | 0.039±0.005 | 0.033±0.014 | 0.039±0.002 |
| **MocoSFL-3** | Accuracy (%) | 81.19±2.32 | 80.51±1.49 | **83.13±2.40** |
| | Attack MSE | 0.045±0.003 | 0.035±0.003 | **0.039±0.002** |

Table 4: Accuracy and MIA resistance performance (Attack MSE) of MocoSFL+TAResSFL. (Averaging 3 random seeds)

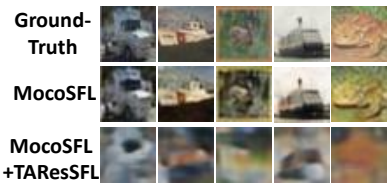



Figure 6: Visualization of MIA reconstructed images using TAResSFL.

We use $G$ to perform MIA attack; the architecture information of $G$ is given in Appendix A.3. We assume the real-time attacker also uses an attack model with the same architecture as $G$. For the pre-training step, we use a fixed hyper-parameter choice of $\lambda = 2.0$ and a target SSIM level of 0.6 to limit the strength of regularization. We assume 0.0%, 0.5% and 1.0% of the target dataset CIFAR-10 is accessible, and we use CIFAR-100 dataset as source dataset to assist the pre-training. For the

bottleneck layers, we use a pair of convolutional layers of 4 channels with stride of 2 (C4S2). We set the cut-layer choices to 1 and 3 and set the #clients to 100 for the training process.

**Successful mitigation of MIA.** As shown in Table 4, applying TAResSFL can achieve good accuracy performance (>81%) as well as high enough MIA resistance (>0.020) for most cases. Fig. 6 shows the visualization of MocoSFL-3 when 1.0% target data is available. We observe the reconstructed images are much more noisy and blurry such that the subject can be successfully hidden.

**Larger cut-layer allows a better resistance-accuracy tradeoff.** Using a smaller cut-layer of 1 seems a better choice in terms of accuracy and hardware requirement. However, as shown in Table 4, the accuracy and resistance tradeoff seems much better by using a cut-layer of 3. We believe that the extra client-side model complexity helps in the optimization of both accuracy and resistance objectives. So, applying TAResSFL with a larger cut-layer is a more favorable option.

### 5.3 Hardware Demonstration

Finally, we compare the total hardware resource cost of the proposed MocoSFL and "MocoSFL+TAResSFL" with synchronization frequency of 1/epoch for 200 epochs, and FL-SSL (Zhuang et al., 2022) with 500 local epochs per client and synchronization frequency set to 1 per 5 local epochs (original setting). For MocoSFL, we use 1,000 clients with batch size of 1, and cut-layer of 3. For FL-SSL, to achieve similar accuracy, we use 5 clients with batch size of 128. And we assume the data follows the default 2-class non-IID setting. For hardware overhead evaluation, we use commonly-used libraries such as 'fvcore' for FLOPs and 'torch.cuda.memory_allocated' for memory (please see Appendix A.5 for detail.) For power measurement, we use a Raspberry Pi 4B equipped with 1GB memory as one real client and simulate all other clients on the PC.

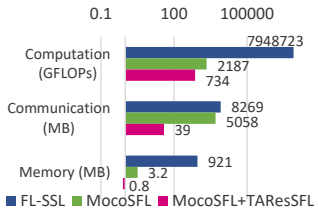



Figure 7: Hardware resource comparison between FL-SSL and MocoSFL schemes.

Figure 8: Raspberry Pi setup. The Raspberry connects to the output via a measurement tool, and communicates wirelessly with the server using websocket (TCP).

As shown in Fig. 7, FL-SSL requires 7,949 TFLOPs for the entire training session as each client needs to perform 500 local epochs on 10K data and a memory consumption of 921 MB[2]. The communication overhead due to synchronization has a high cost of around 8,269 MB. Thus, FL-SSL is only suitable for cross-silo situation. For "MocoSFL+TAResSFL", hardware requirements are reasonable since computation is only 734 GFLOPs and communication is around 39 MB in total. The memory consumption is a tiny 0.8 MB mostly due to the weight parameters since TAResSFL does not require local training. As shown in Fig. 8, our measurement using a USB multimeter shows the proposed MocoSFL running on the Raspberry Pi only draws power of 2.26W, in average, and consumes around 9,100 mAh on a 5V battery.

## 6 Conclusion

We propose MocoSFL, a collaborative SSL framework based on SFL. The proposed framework addresses hardware resource requirement at client-side by enabling small batch size training and computation offloading. It also relieves the large data requirement of local contrastive learning by enabling effective feature sharing. The proposed scheme is the only one that can support a large number of clients. In combination with a ResSFL-based module, it addresses privacy concerns of MocoSFL. Finally, it achieves even better IID/non-IID performance with much lower hardware requirement than the SoTA FL-based SSL methods.

[2]Memory for holding model parameters and intermediate data; actual CPU/GPU memory consumption can be higher considering memory for CUDA library/OS kernel.

## 7 ETHICS STATEMENT

In this work, we address two practical issues of hardware (compute and memory) resources and large data requirement in collaborative SSL by using a SFL-based scheme. Compared to the conventional approaches, the resulting MocoSFL scheme has significant advantages in affordability and is much easier to deploy. Apart from being environmental-friendly, MocoSFL makes it possible to reach more clients, especially for those in poverty or under-represented minorities, and thus eliminate potential human-bias and unfairness issues of the ML service.

We address the privacy issues in the proposed MocoSFL by proposing TAResSFL module to use in sensitive applications. Fig. 6 shows that the subject of the raw image can be successfully hidden.

**More discussion on the privacy of "MocoSFL+TAResSFL".** We notice a line of works known as instance encoding (Huang et al., 2020; Yala et al., 2021), which try to protect users' data by transforming the original dataset to a distorted version such that they cannot be recognized by humans while an arbitrary deep learning model can still learn useful information from it (i.e. achieve high accuracy on a classification task). MocoSFL, especially with the TAResSFL, where the client-side model is frozen, have some similarities since the frozen client-side model can be seen as a transformation, and the collection of latent vectors can be regarded as the transformed dataset. However, "MocoSFL+TAResSFL" scheme has two favorable properties that make it distinct from instance encoding methods. **Domain dependency.** First, SFL only finds a transformation method for a given task. Since the transformation method itself (the client-side model) heavily depends on the target domain information (as we need access to the target domain data to train the client-side model), it cannot work on data from another domain. As the output of the "client-side model transformation", latent vectors are only useful for current problem without any transferability guarantee. However, instance encoding methods intend to derive a general transformation method that can work across domains, targeting a harder problem. **Risk Control.** Instance encoding methods publish the transformed dataset to the wide public which is risky. As pointed out by Carlini et al. (2021): all the raw data will be leaked if a successful decryption method is invented in the future even if it does not exist now. While in our proposed scheme, the latent vectors from clients will only be accessible to the server party, thereby reducing the risk. Since we already provide TAResSFL to mitigate the MIA attack that can be possibly launched from the honest-but-curious server, the risk is minimized. To eliminate possibility of future advanced attack, we can introduce a protocol that requests the server party to regularly delete[3] these latent vectors immediately after the training is done.

## 8 REPRODUCIBILITY STATEMENT

To make it easier for readers to reproduce the results in this work, we have provided detailed explanation on the MocoSFL training and evaluation hyperparameters, collaborative learning system hyperparameters in Appendix A.1, and TAResSFL module hyperparameters in Appendix A.4. Moreover, we plan to release our code upon acceptance.

[3]This is a common practice according to https://oag.ca.gov/privacy/ccpa

## REFERENCES


Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.

Adrian Bulat, Enrique Sánchez-Lozano, and Georgios Tzimiropoulos. Improving memory banks for unsupervised learning with large mini-batch, consistency and hard negative mining. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1695–1699. IEEE, 2021.

Nicholas Carlini, Samuel Deng, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Abhradeep Thakurta, and Florian Tramèr. Is private learning possible with instance encoding? In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 410–427. IEEE, 2021.

Chaochao Chen, Jun Zhou, Longfei Zheng, Huiwen Wu, Lingjuan Lyu, Jia Wu, Bingzhe Wu, Ziqi Liu, Li Wang, and Xiaolin Zheng. Vertically federated graph neural network for privacy-preserving node classification. In *IJCAI*, 2022.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020a.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020b.

Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9640–9649, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Zhiyuan Fang, Jianfeng Wang, Lijuan Wang, Lei Zhang, Yezhou Yang, and Zicheng Liu. Seed: Self-supervised distillation for visual representation. *arXiv preprint arXiv:2101.04731*, 2021.

Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems*, 33: 21271–21284, 2020.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

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*, pages 9729–9738, 2020.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

Yangsibo Huang, Zhao Song, Kai Li, and Sanjeev Arora. Instahide: Instance-hiding schemes for private distributed learning. In *International conference on machine learning*, pages 4507–4518. PMLR, 2020.

Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard negative mixing for contrastive learning. *Advances in Neural Information Processing Systems*, 33:21798–21809, 2020.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pages 491–507. Springer, 2020.

Jingtao Li, Adnan Siraj Rakin, Xing Chen, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Ressfl: A resistance transfer framework for defending model inversion attack in split federated learning, 2022.

Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Anti-backdoor learning: Training clean models on poisoned data. *Advances in Neural Information Processing Systems*, 34:14900–14912, 2021.

Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. Fedvision: An online visual object detection platform powered by federated learning. In *IAAI*, 2020.

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

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

Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

Joshua David Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In *International Conference on Learning Representations*, 2020.

Holger R Roth, Ken Chang, Praveer Singh, Nir Neumark, Wenqi Li, Vikash Gupta, Sharut Gupta, Liangqiong Qu, Alvin Ihsani, Bernardo C Bizzo, et al. Federated learning for breast density classification: A real-world implementation. In *Domain adaptation and representation transfer, and distributed and collaborative learning*, pages 181–191. Springer, 2020.

Haizhou Shi, Youcai Zhang, Siliang Tang, Wenjie Zhu, Yaqian Li, Yandong Guo, and Yueting Zhuang. On the efficacy of small self-supervised contrastive models without distillation signals. *arXiv preprint arXiv:2107.14762*, 2021.

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

Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. *arXiv preprint arXiv:2004.12088*, 2020.

Praneeth Vepakomma, Abhishek Singh, Otkrist Gupta, and Ramesh Raskar. Nopeek: Information leakage reduction to share activations in distributed deep learning. In *2020 International Conference on Data Mining Workshops (ICDMW)*, pages 933–942. IEEE, 2020.

Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. Communication-efficient federated learning via knowledge distillation. *Nature communications*, 13(1):1–8, 2022a.

Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. Fedctr: Federated native ad ctr prediction with cross platform user behavior data. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2022b.

Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Tao Qi, Yongfeng Huang, and Xing Xie. A federated graph neural network framework for privacy-preserving personalization. *Nature Communications*, 13(1):1–10, 2022c.

Yawen Wu, Dewen Zeng, Zhepeng Wang, Yi Sheng, Lei Yang, Alaina J James, Yiyu Shi, and Jingtong Hu. Federated contrastive learning for dermatological disease diagnosis via on-device learning. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–7. IEEE, 2021.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3733–3742, 2018.

Adam Yala, Homa Esfahanizadeh, Rafael GL D' Oliveira, Ken R Duffy, Manya Ghobadi, Tommi S Jaakkola, Vinod Vaikuntanathan, Regina Barzilay, and Muriel Medard. Neuracrypt: Hiding private health data via random neural networks for public training. *arXiv preprint arXiv:2106.02484*, 2021.

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*, 2020.

Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57, 2016.

Weiming Zhuang, Xin Gan, Yonggang Wen, Shuai Zhang, and Shuai Yi. Collaborative unsupervised visual representation learning from decentralized data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4912–4921, 2021.

Weiming Zhuang, Yonggang Wen, and Shuai Zhang. Divergence-aware federated self-supervised learning. *arXiv preprint arXiv:2204.04385*, 2022.

# A Technical Details

## A.1 Hyper-parameter Choices

**Training Hyper-parameters.** For data preprocessing, we split each dataset into 50K training data and 10K validation data. We use the training data directly as query data $X_q$, and its augmented version as positive data $X_{k^+}$. We use the same augmentation pipeline as Chen et al. (2020b). For the backbone model, we remove the last fully-connected layer and replace it with a MLP head consisting of two larger fully connected layers of 1024 neurons. To fit well with small batch size training, we replace the batch normalization layers by *group normalization* layers and set the number of channels per group to 4 while keeping the minimum number of groups to 1 if the number of channels is less than 4. We also use the *weight standardization* technique (Qiao et al., 2019) in combination with group normalization for better performance (Kolesnikov et al., 2020).

At the server-side, we use a feature memory of 6,000 negative keys that is implemented as a First-In-First-Out (FIFO) queue, and we update it by pushing newly generated positive keys at every training step. For contrastive loss calculation, we use the symmetric contrastive loss calculation introduced in Moco-V3 (Chen et al., 2021), where the position of query and positive keys are swapped in Eq. (1), and the new loss takes the form of $(\mathcal{L}_Q, K, N + \mathcal{L}_K, Q, N)/2$. For model updating, we update the online model of both client-side and server-side model with the SGD optimizer with 0.06 initial learning rate, 0.9 momentum and 0.0005 weight decay. We schedule the learning rate using the *cosine annealing* trick for both client-side and server-side models. The momentum model update is done through the moving average mechanism in He et al. (2020). According to previous study Chen et al. (2020b), the training accuracy converges with extensive long training (i.e. $>$800 epochs), however, such long training session is impractical for collaborative learning setting. Thus, we set the number of epochs to 200 to maximize time and performance tradeoff. At the end of each epoch, we save the model and perform a k-nearest-neighbor (knn) validation using the 20% validation dataset. Finally, we use the saved model that has the best knn accuracy to perform the final linear probe evaluation.

**Evaluation Hyper-parameters.** For the knn validation, we feed the validation dataset and use the normalized feature output of the backbone (together with the MLP head) and follow the implementation in Wu et al. (2018) to get the accuracy. For the linear probe evaluation, we follow exactly the same procedure as in Zhuang et al. (2022), where we replace the MLP head with a randomly initialized fully-connected layer, fine-tune the new model on the labeled dataset while keeping the backbone parameters frozen. We train the single fully-connected layer for a total of 100 epochs on the labeled training dataset and use the accuracy on the labeled validation dataset as final accuracy. To optimize the single FC layer, we use a batch size of 128 and Adam optimizer with default learning rate 0.001 and use cosine annealing to schedule its learning rate.

**Collaborative Learning Hyper-parameters: Synchronization Frequency and Client Sampling Ratio.** In our proposed MocoSFL scheme, we use an auto-adjust mechanism for adjusting the synchronization frequency, the client-sampling ratio, and the local batch size according to number of clients. The auto-adjust mechanism aims to strike a balance between model divergence and sample hardness and is based on two principles (also supported by empirical results shown in Table 6 and Table 7). **The first principle** is to always keep the server-side batch size around 100, so that we can keep a good hardness (indicated by Eq. (3)). For example, for 5-client MocoSFL, we force each client to have a batch size of 20 to keep the equivalent batch size to 100. For 1,000-client MocoSFL, we use a client sampling ratio of 0.1 to let 100 clients join in each round. **The second**

**principle** is that the synchronization frequency must be adjusted according to the number of local training steps. The reason is that more local updates leads to higher model divergence, which needs to be mitigated using more frequent synchronization. For example, for a 100-client MocoSFL where each client performs 500 local training steps, accuracy drops for a synchronization frequency of "1/epoch", and accuracy increases until "10/epoch". However, for a 1,000-client MocoSFL where each client has 50 data and performs 50 local training steps in each epoch (with batch size of 1), setting synchronization frequency to "2/epoch" is sufficient. We provide further evidence of the above two principles in Appendix B.2.

**TAResSFL Hyper-parameters.** For the Hyperparameter choices in TAResSFL, please refer to Appendix A.4.

### A.2 SFL-V1 Process

**Algorithm 1** Split Federated Learning V1 (Thapa et al., 2020)

**Require:** For $N_C$ clients, instantiate private training data $(\mathbf{X}_i, \mathbf{Y}_i)$ for $1, 2, ..., N_C$. Each client-side model $C_i$ has $L$ layers and server-side model $S$ contains the remaining layers.

1: **for** epoch $t \leftarrow 1$ to num_epochs **do**
2: $C^* = \frac{1}{N_C}\sum_{i=1}^{N_C} C_i;\ C_i \leftarrow C^*$ for all $i$ {Model Synchronization}
3: **for** step $s \leftarrow 1$ to num_batches **do**
4: **for** client $i \leftarrow 1$ to $N_C$ **in Parallel do**
5: data batch $(\boldsymbol{x}_i, \boldsymbol{y}_i) \leftarrow (\mathbf{X}_i, \mathbf{Y}_i)$
6: $\boldsymbol{A}_i = C_i(\boldsymbol{W}_{C_i}; \boldsymbol{x}_i)$ {Send $\boldsymbol{A}_i$ to Server}
7: **end for**
8: $\boldsymbol{A} = cat(\boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_{N_C});\quad \boldsymbol{y} = cat(\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_{N_C})$
9: $\mathcal{L} = \mathcal{L}_{CE}(S(\boldsymbol{W}_S; \boldsymbol{A}), \boldsymbol{y})$
10: $\nabla_{\boldsymbol{A}}\mathcal{L} \leftarrow$ back-propagation {Partition $\nabla_{\boldsymbol{A}}\mathcal{L}$, Send $\nabla_{\boldsymbol{A}_i}\mathcal{L}$ Correspondingly to Client $i$}
11: Update $\boldsymbol{W}_S$;
12: **for** client $i \leftarrow 1$ to $N_C$ **in Parallel do**
13: $\nabla_{\boldsymbol{x}_i}\mathcal{L} \leftarrow$ back-propagation
14: Update $\boldsymbol{W}_{C_i}$;
15: **end for**
16: **end for**
17: **end for**

The training process of SFL-V1 in supervised learning is shown in Algorithm 1. Since a large equivalent batch size at server side is essential for the success of contrastive learning, we adopt SFL-V1. Here a large equivalent batch size (equal to the number of clients times clients' batch size) can be achieved due to latent vector concatenation as shown in line 8 of Algorithm 1. We choose SFL-V1 over SFL-V2 because: V1 performs concatenation of clients' latent vectors and feeds it to S and updates the model only one time (call "optimizer.step() for one time" in pytorch), while V2 sequentially feeds them and updates S for a total of $N_C$ times, where $N_C$ is the number of clients. In PyTorch, this means calling "optimizer.step() for $N_C$ time".

### A.3 Data Privacy: Model inversion attack

The MIA workflow is shown in Fig. 9. It follows the same procedure as previous works on the data privacy of SFL (Vepakomma et al., 2020; Li et al., 2022).

**Threat Model.** The basic assumption is that the server follows the honest-but-curious assumption, that is, it follows the protocol strictly, but can gather information during this process to breach the privacy. We also assume the server party can access the 10K validation dataset since validation needs to be done by the server for monitoring purpose (Bhagoji et al., 2019).

**Attack Preparation.** To perform MIA, the server needs to instantiate an inversion model. We assume that the server use the L3 inversion model (top-tier inversion model in Li et al. (2022)) as shown

in Fig. 10. Then, the server queries the client-side model using the 10K validation dataset. to gather enough input-output pairs. These pairs are used to train the initialized inversion model, where we use mean-squre-error (MSE) loss with Adam optimizer with 0.001 learning rate for 50 epochs and set the batch size to 32.

**Attack Evaluation.** Once the inversion model is ready, server can now use it on the latent vectors sent by the target client to generate the reconstructed image. We use a fixed set of 128 images that are randomly sampled from the training dataset to represent the private data of the target client, and use the MSE metric between the ground-truth image and reconstructed image to evaluate the data privacy (higher the MSE, higher the data privacy).

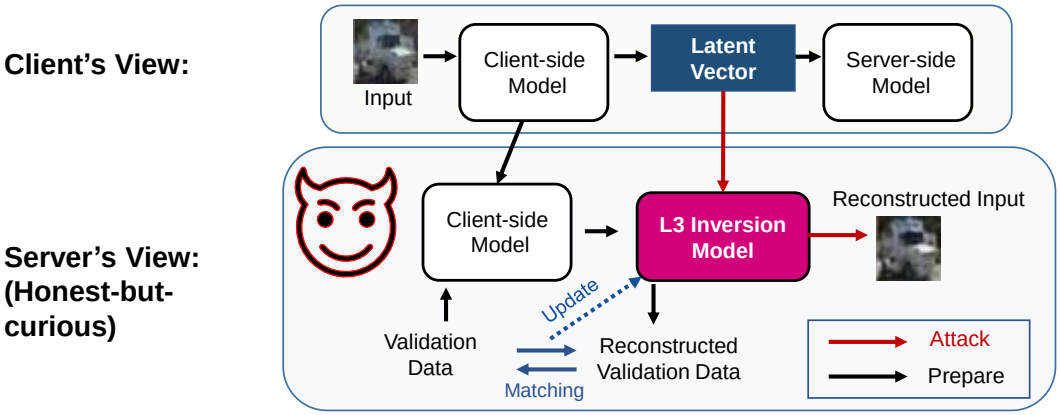



Figure 9: Details of model inversion attack using L3 inversion model done by an honest-but-curious server.

```
Inversion_Model_L3(
  (m): Sequential(
    (0): ResBlock(128, 64 , BN = True)
    (1): ReLU()
    (2): ResBlock(64, 64 , BN = True)
    (3): ReLU()
    (4): ResBlock(64, 64, BN = True)
    (5): ReLU()
    (6): ResBlock(64, 64 , BN = True)
    (7): ReLU()
    (8): ResBlock(64, 64 , BN = True)
    (9): ReLU()
    (10): ConvTranspose2d(64, 64, kernel_size=(3, 3),
stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (12): ReLU()
    (13): ConvTranspose2d(64, 64, kernel_size=(3, 3),
stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (14): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (15): ReLU()
    (16): ResBlock(64, 3 , BN = True)
    (17): Sigmoid()
  )
)
```

Figure 10: Details of L3 inversion model architecture.

### A.4 TAResSFL module

The MocoSFL+TAResSFL scheme is depicted in Fig. 11. It consists of a pretraining step and a transfer step. The default architecture is modified by introducing a pair of bottleneck layers (the last layer of "C-model" and the first layer of the tail model). The pretraining step is done offline at

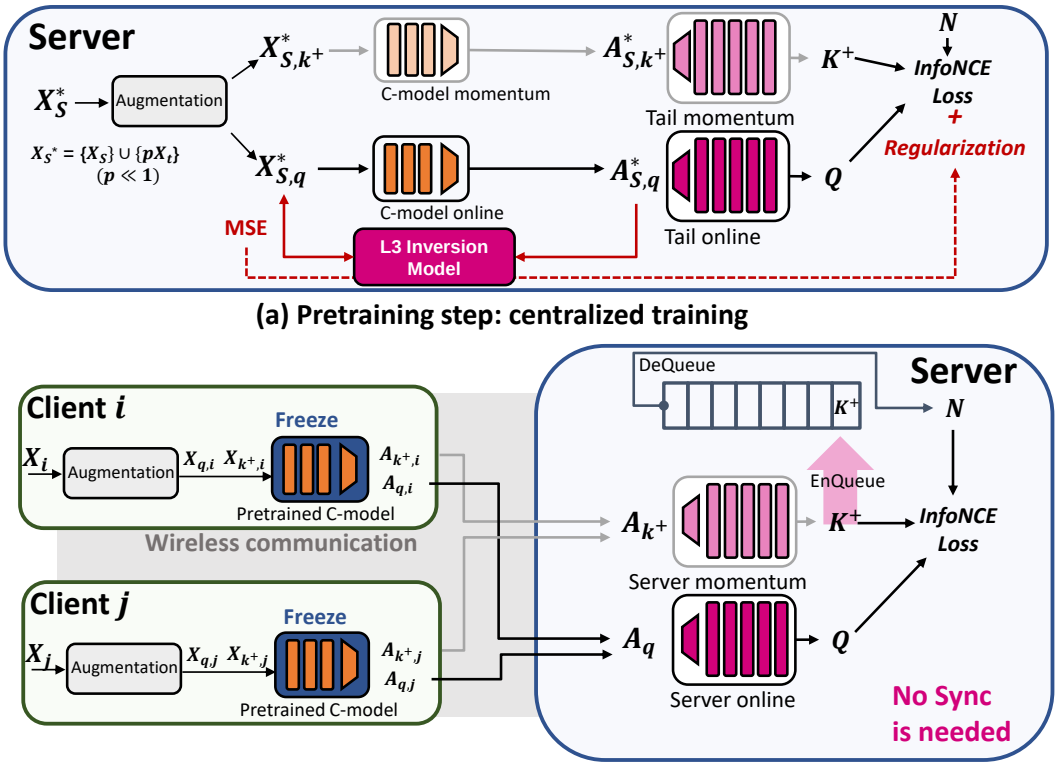



Figure 11: Overall MocoSFL + TAResSFL scheme.

the server and the transfer step is the actual MocoSFL collaborative learning step, where clients use the pretrained "C-model" as the initial client-side model and participate in the SFL-based training to protect data privacy from MIA attacks. In the pretraining step in Fig. 11 (a), the input $X_S^*$ is the mixture of source dataset $X_S$ (an out-of-distribution dataset/auxiliary dataset) and a small proportion $p$ of target dataset $X_S$. The optimization process follows Eq. (5), whose objective is to make "C-model" have enough resistance to MIA attack while keeping a high accuracy. In the transfer step in Fig. 11 (b), all client-side models are initialized using the pretrained "C-model" and kept frozen. All clients feed augmented images to "C-model" locally and send latent vectors to the server to perform the MocoSFL training process. This design reduces communication overhead dramatically thanks to bottleneck layers. It also reduces computation and memory since client-side model's backward computation and synchronization costs are saved.

**TAResSFL hyperparameters.** For the pretraining step, we use a single-client MocoSFL scheme to simulate the offline server pretraining process. We use batch size of 128 with learning rate of 0.06 that is scheduled by cosine annealing. We add a pair of bottleneck layers, both with 3x3 kernel size. The first bottleneck layer is a Conv2D layer that has output channel size of 4 and stride of 2 and is placed at the end of client-side model as part of "C-model", the second one is a Transpose Conv2d layer ("torch.nn.ConvTranspose2d" in pytorch), having a stride of 2 that is placed at the front end of the tail model. The training data is a combination of source data and a small proportion of target data. For example, we use CIFAR-100 as the source data and combine it with 0.0%, 0.5%, and 1.0% randomly sampled target data to derive the training dataset of the pretraining step. Then, we use MSE loss to optimize the inversion model and use Structural Similarity Index (SSIM) loss of the inversion model as the regularization term of the contrastive loss. We set $\lambda$ to 2.0 and SSIM threshold to 0.6. If SSIM is lower than 0.6, then we disable the regularization term temporarily, until it is higher than 0.6 again. to stabilize the training. As described earlier, we perform one inversion model training step and one classification model (C-model + tail model) training step for every source data batch. The number of epochs for the pretraining step is set to 200.

For the transfer step, we initialize the client-side model using the "C-model" derived in the pretraining step and initialize the server-side model using the "tail model". During training, the client-side model

update is disabled (keep client-side model frozen), and the server-side model is optimized using SGD with a small learning rate of 0.001, to be able to achieve high accuracy on the target data. We also remove client-side momentum models, the client-side model backward operation and the client-side models' synchronization step since they are not required for frozen client-side models. The number of epochs for the transfer step is set to 200.

### A.5 Communication, Computation and Memory Overhead Calculation

For communication overhead calculation, we divide it into two parts, which are latent vector size and weight size. The latent vector size is determined by the output size of client-side model, and the weight size is determined by the parameter size of client-side model. In each epoch, weight communication is weight size multiplies by the number of synchronization times per epoch, while latent vector communication is latent vector size multiplied by the total number of data sent in this epoch. For FL, we only need to count the weight communication. For MocoSFL without TAResSFL module, we need to count the weight communication and double the latent vector communication since clients need to receive the gradient sent back by the server. For MocoSFL with TAResSFL module, the communication overhead does not need to double, also, the weight communication is zero since synchronization is not needed. Finally, we multiply communication overhead per epoch by the total number of epochs.

For computation overhead, we use FLOPs to represent since floating-point operation dominates the computation. The FLOPs of the models were counted by the 'fvcore' library developed by Facebook AI Research. To count the inference FLOPs, we multiply the output of 'fvcore.nn.FlopCountAnalysis' on a single image by the number of data per client and then multiply it with the total number of epochs. Due to symmetry, the FLOPs of model backpropagation are approximately equal to that of the model inference. Thus, the number we show for training FLOPs is by doubling the inference FLOPs.

For memory overhead, we use 'torch.cuda.memory_allocated' API to measure the allocated memory for training and inference.

## B Extensive Empirical Results

In this section, we perform extensive empirical evidence of the proposed MocoSFL scheme on different architecture (Appendix B.1), evidence of two hyperparameter choice principles (Appendix B.2), performance evaluation of TAResSFL with different hyperparameters (Appendix B.3), performance evaluation of MocoSFL in a semi-supervised learning application (Appendix B.4), and extensive comparison with other schemes.

### B.1 ResNet-50 results

We follow the same setup as in Table 2 and Table 3 on ResNet-50 architecture for cross-silo (5 and 20 clients) and cross-client settings (100 clients). Table 5 shows that compared with ResNet-18 model, ResNet-50 has $\sim$3% better accuracy performance.

| **Method** | $N_C = 5$ | | $N_C = 20$ | | $N_C = 100$ | |
|---|---|---|---|---|---|---|
| | IID | Non-IID | IID | Non-IID | IID | Non-IID |
| MocoSFL-1 (ResNet-18) | 87.38 | 87.81 | 87.21 | 85.84 | 87.29 | 87.71 |
| MocoSFL-3 (ResNet-18) | 87.09 | 87.29 | 87.09 | 85.32 | 87.29 | 87.10 |
| MocoSFL-1 (ResNet-50) | 90.68 | 90.78 | 91.07 | 89.35 | 90.59 | 90.96 |
| MocoSFL-3 (ResNet-50) | 90.86 | 90.94 | 90.58 | 89.83 | 90.67 | 90.66 |

Table 5: MocoSFL Performance (linear probe accuracy) on a ResNet-18 and ResNet-50 models

### B.2 EFFECT OF BATCH SIZE, SYNCHRONIZATION FREQUENCY AND FEATURE SHARING

We provide some empirical evidence to support the **two principles** presented in Appendix A.1. The first is keeping the batch size to above 100. Table 6 provides the accuracy performance when the number of clients is set to 5 for different server-side batch size $B$ (after vector concatenation). So $B = 50$ corresponds to local batch size of 10 since there are 5 clients. A small batch size has much worse performance (around 5% lower), while a large batch size (i.e., 100, 200) helps keep a good hardness and maintains good accuracy. However, when batch size increases further to 400, we notice small degradation in accuracy. Thus, as stated in the **first principle**, when the number of clients is larger than 200 and each client locally uses a batch size of 1 (resulting in a batch size of 200), we have to apply client sampling to force a smaller total batch size.

Table 6: MocoSFL-3's accuracy performance (ResNet-18 on CIFAR-10) varies with batch size. This observation leads to the **first principle** of hyperparameter choice.

| **Distribution** | **Batch Size** | | | | |
|---|---|---|---|---|---|
| | $B = 25$ | $B = 50$ | $B = 100$ | $B = 200$ | $B = 400$ |
| IID | 79.66 | 85.00 | 86.77 | 87.49 | 86.61 |
| Non-IID | 74.51 | 80.31 | 83.14 | 84.90 | 84.64 |

We also investigate the necessity of synchronization frequency when the number of clients increases. While previous experiments (Fig. 5) demonstrate significant benefit of increasing synchronization frequency, we find that for a cross-client application where each client has very small amount of data, increasing synchronization frequency does not have much impact on the accuracy. Table 7 illustrates this finding. The accuracy performance quickly saturates at "2/epoch" and shows no further improvement when frequency is increased further. This is because the number of local training steps becomes less as stated in **second principle** (see Appendix A.1).

Table 7: MocoSFL-3's accuracy performance (ResNet-18 on CIFAR-10) quickly saturates with synchronization frequency when the number of clients is large (so the amount of data in each client is small).

| | **Synchronization Frequency** | | | |
|---|---|---|---|---|
| | 1/epoch | 2/epoch | 3/epoch | 5/epoch |
| Non-IID Accuracy | 84.75 | 87.47 | 87.41 | 87.38 |

Next, we investigate the effectiveness of feature sharing in MocoSFL. In MocoSFL, the negative keys from different clients are shared. If such sharing is disabled and server stores keys contributed from different clients separately in different feature memories, the accuracy is significantly lower. As shown in Fig. 12, without feature sharing, both IID and non-IID accuracy drop significantly.

### B.3 DIFFERENT TARESSFL SETTINGS

We vary the hyperparameter settings of the proposed "MocoSFL + TAResSFL" scheme on ResNet-18 on CIFAR-10 dataset. In Table 8, we change the source dataset to SVHN. Here, we observe that using SVHN as source data gets worse accuracy performance than using CIFAR-100. This is because the domain difference between SVHN and CIFAR-10 (target dataset) is much larger than CIFAR-100 and CIFAR-10. However, even with such domain difference, the proposed method still works. For MocoSFL with cut-layer of 3, we can still achieve over 80% accuracy with 0.034 MSE.

In Table 9, we vary the bottleneck layer settings, where we set the bottleneck layer size to $C2S2$, $C4S2$ and $C8S2$. A smaller channel size leads to better "bottlenecking" effect that is, lower communication overhead, better resistance to attack, but lower accuracy performance. For MocoSFL-3, we observe that $C2S2$ achieves the highest resistance with lowest accuracy, and using $C8S2$ cannot meet the required resistance. We find that using $C4S2$ is the best choice in our specific case.

In Table 10, we vary the regularization strength $\lambda$. The $\lambda$ affects the gradient magnitude contributed by the regularization; using higher $\lambda$ can boost the resistance but may lead to accuracy drop. However, we observe resistance almost saturates when $\lambda = 2.0$ and hence use this setting across our experiments.

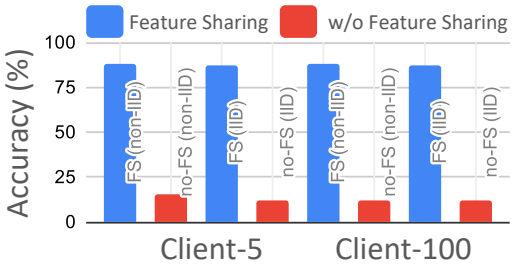



Figure 12: Effect of feature sharing in MocoSFL.

| Method | Metric | Target Data | | |
|---|---|---|---|---|
| | | **0.0%** | **0.5%** | **1.0%** |
| **MocoSFL-1** | Accuracy (%) | 70.54 | 79.26 | 74.25 |
| | Attack MSE | 0.039 | 0.035 | 0.037 |
| **MocoSFL-3** | Accuracy (%) | 79.41 | 76.02 | 80.19 |
| | Attack MSE | 0.041 | 0.035 | 0.034 |

Table 8: Performance of "MocoSFL + TAResSFL" **using SVHN as source dataset**, with 0.0%, 0.5% and 1.0% of CIFAR-10 as target dataset.

| Method | Metric | Bottleneck Layer | | |
|---|---|---|---|---|
| | | **C2S2** | **C4S2** | **C8S2** |
| **MocoSFL-1** | Accuracy (%) | 78.21 | 77.86 | 76.12 |
| | Attack MSE | 0.037 | 0.040 | 0.020 |
| **MocoSFL-3** | Accuracy (%) | 80.94 | 84.81 | 82.39 |
| | Attack MSE | 0.050 | 0.038 | 0.009 |

Table 9: Performance of "MocoSFL + TAResSFL" using CIFAR-100 as source dataset and 1.0% of CIFAR-10 target dataset as training data, with **different bottleneck layer settings**.

| Method | Metric | Regularization $\lambda$ | | |
|---|---|---|---|---|
| | | 1.0 | 2.0 | 4.0 |
| **MocoSFL-1** | Accuracy (%) | 79.83 | 77.86 | 80.43 |
| | Attack MSE | 0.019 | 0.040 | 0.040 |
| **MocoSFL-3** | Accuracy (%) | 82.58 | 84.81 | 80.82 |
| | Attack MSE | 0.040 | 0.038 | 0.041 |

Table 10: Performance of "MocoSFL + TAResSFL" using CIFAR-100 as source dataset and 1.0% of CIFAR-10 target dataset as training data, with **different regularization strength** $\lambda$.

### B.4 Semi-supervised Learning Evaluation

We report the performance of using the proposed MocoSFL for semi-supervised learning applications for both cross-silo (5 and 20 clients) and cross-client setting (100 clients). As shown in Table 11, the proposed MocoSFL achieves competitive accuracy performance. For ResNet-18 model, we achieve 10% higher accuracy than FL-BYOL (Zhuang et al., 2022) for the semi-supervised learning setting where only 1% label is given on a CIFAR-10 dataset.

| Arch | Method | $N_C = 5$ | | $N_C = 20$ | | $N_C = 100$ | |
|---|---|---|---|---|---|---|---|
| | | 1% | 10% | 1% | 10% | 1% | 10% |
| **ResNet-18** | FL-BYOL (Zhuang et al., 2022) | 73.44 | 79.49 | N/A | N/A | N/A | N/A |
| | MocoSFL-1 | 84.69 | 87.05 | 76.80 | 84.46 | 82.27 | 86.84 |
| | MocoSFL-3 | 84.46 | 86.50 | 76.70 | 84.20 | 81.72 | 85.91 |
| **ResNet-50** | FL-BYOL (Zhuang et al., 2022) | 72.52 | 80.68 | N/A | N/A | N/A | N/A |
| | MocoSFL-1 | 86.28 | 89.55 | 80.34 | 86.09 | 84.83 | 88.78 |
| | MocoSFL-3 | 87.09 | 90.33 | 81.59 | 87.21 | 84.65 | 88.81 |

Table 11: MocoSFL Semi-supervised learning Performance (Non-IID).

### B.5 Extensive Comparison with Other Schemes

We provide a comprehensive comparison with prior SoTA works (Zhuang et al., 2021; 2022) in Table 12. Our proposed MocoSFL scheme shows significantly better accuracy performance, especially for ResNet-50 model, where we achieve 5% higher accuracy than Zhuang et al. (2022) on all metrics (linear probe and semi-supervised with 1% and 10% labels).

Table 12: Top-1 accuracy comparison under linear evaluation protocol, on 1% and 10% of labeled data for semi-supervised learning on the non-IID setting of CIFAR datasets. MocoSFL outperforms all other methods (number of clients is set to 5).

| **Method** | **Arch** | **CIFAR-10 (%)** | | | **CIFAR-100 (%)** | | |
|---|---|---|---|---|---|---|---|
| | | Linear | 1% label | 10% label | Linear | 1% label | 10% label |
| FedU (Zhuang et al., 2021) | ResNet-18 | 80.52 | 69.52 | 77.06 | 57.21 | 29.00 | 46.67 |
| FedEMA (Zhuang et al., 2022) | ResNet-18 | 83.34 | 73.44 | 79.49 | 61.78 | 33.04 | 50.48 |
| MocoSFL-1 (ours) | ResNet-18 | 87.81 | 84.69 | 87.05 | 58.78 | 33.09 | 51.59 |
| MocoSFL-3 (ours) | ResNet-18 | 87.29 | 73.44 | 79.49 | 57.70 | 32.44 | 51.21 |
| FedCA (Zhang et al., 2020) | ResNet-50 | 68.01 | 28.50 | 36.28 | 42.34 | 16.48 | 22.46 |
| FedU (Zhuang et al., 2021) | ResNet-50 | 83.25 | 69.76 | 80.25 | 61.94 | 28.42 | 48.42 |
| FedEMA (Zhuang et al., 2022) | ResNet-50 | 84.31 | 72.52 | 80.68 | 62.77 | 29.68 | 50.75 |
| MocoSFL-1 (ours) | ResNet-50 | 90.78 | 85.59 | 89.81 | 67.09 | 35.31 | 56.97 |
| MocoSFL-3 (ours) | ResNet-50 | 90.94 | 87.09 | 90.33 | 66.45 | 35.30 | 56.98 |