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 7 and Table 8). **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.3.

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: $\boldsymbol{A} = cat(\boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_{N_C})$; $\boldsymbol{y} = cat(\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_{N_C})$
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).

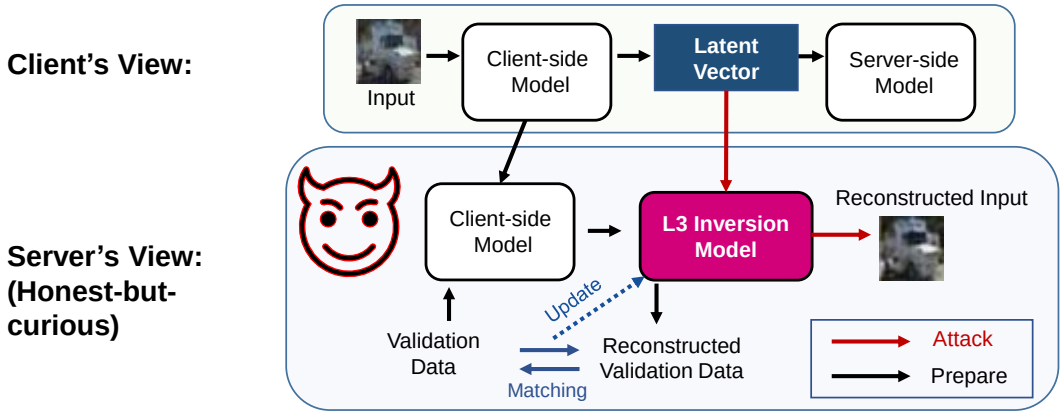

Figure 9: Details of model inversion attack using L3 inversion model done by an honest-but-curious server.

```
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.

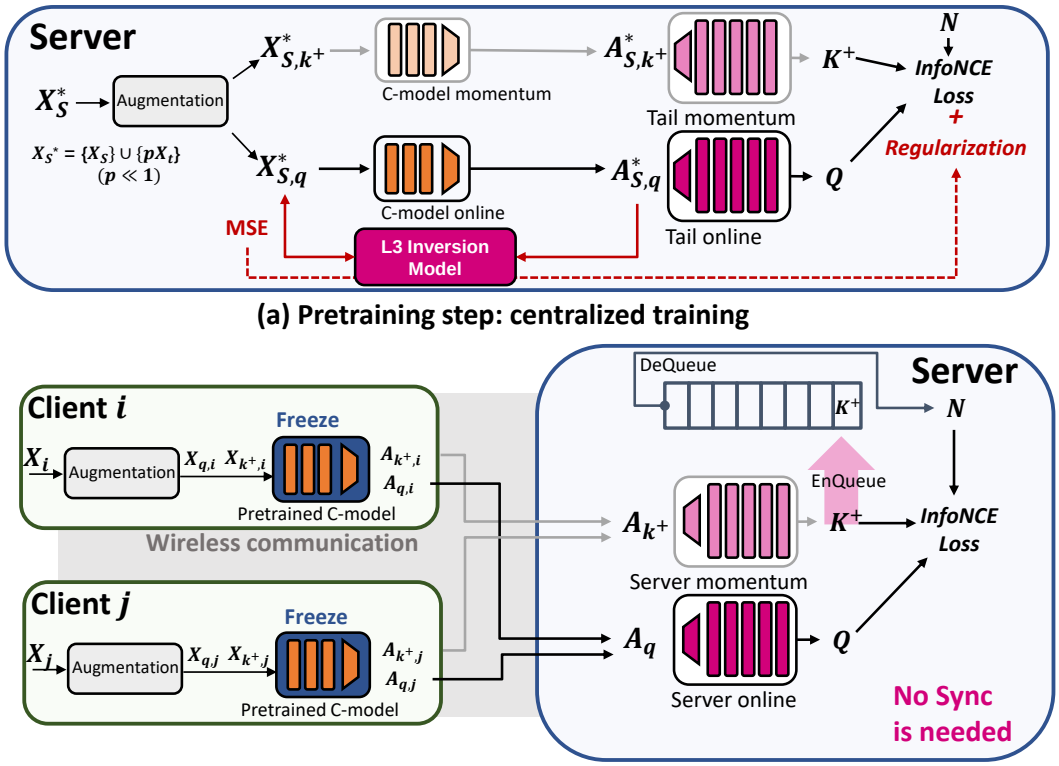

Figure 11: Overall MocoSFL + TAResSFL scheme.

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 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 λ 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 consider the weight communication. For MocoSFL without TAResSFL module, we need to consider the weight communication and also double the latent vector communication since clients receive the gradients sent back by the server. For MocoSFL with TAResSFL module, we do not double the latent vector communication because gradients are not needed, 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. And we also need to count the momentum model inference. Thus, the FLOPs number we show for training is **three times** of the model 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.3), performance evaluation of TAResSFL with different hyperparameters (Appendix B.4), performance evaluation of MocoSFL in a semi-supervised learning application (Appendix B.6), 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.

B.2 MobileNetV2 and VGG-13 cross-client results

We follow the same setup as in Table 2 and Table 3 on MobileNet-V2, VGG-13 architecture for cross-client settings (100 clients). Table 6 shows that MobileNet-V2 and VGG-13 suffers from a large accuracy drop compared with ResNet-18 model.

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

Method	ResNet-18		MobileNetV2		VGG-13	
	IID	Non-IID	IID	Non-IID	IID	Non-IID
MocoSFL-3 ($N_C = 100$)	87.29	87.10	81.64	74.48	81.41	76.02

Table 6: MocoSFL Performance (linear probe accuracy) on a MobileNetV2 and VGG-13 models

B.3 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 7 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 7: 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 8 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).

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.4 Different TAResSFL settings

We vary the hyperparameter settings of the proposed "MocoSFL + TAResSFL" scheme on ResNet-18 on CIFAR-10 dataset. In Table 9, 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

Table 8: 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

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 10, 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 11, we vary the regularization strength λ. The λ affects the gradient magnitude contributed by the regularization; using higher λ 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.

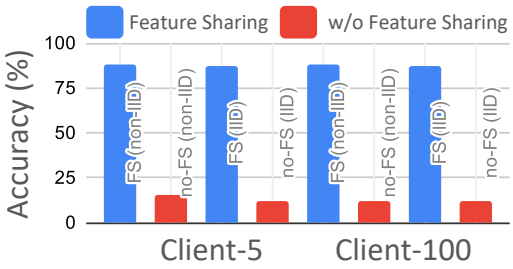

Figure 12: Effect of feature sharing in MocoSFL.

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 9: 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 10: Performance of "MocoSFL + TA-ResSFL" 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 λ**		
		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 11: Performance of "MocoSFL + TA-ResSFL" using CIFAR-100 as source dataset and 1.0% of CIFAR-10 target dataset as training data, with **different regularization strength** λ.

B.5 Extensive Image Quality Metrics for TAResSFL

In Appendix B.5, we report other image quality metrics Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR) metrics for Table 4.

B.6 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 12, 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.

Method	Metric	Target Data		
		0.0%	**0.5%**	**1.0%**
MocoSFL-3	MSE	0.039±0.005	0.033±0.014	0.039±0.002
	SSIM	0.487±0.029	0.545±0.207	0.424±0.012
	PSNR	15.35±0.436	15.93±1.924	14.79±0.090
MocoSFL-1	MSE	0.045±0.003	0.035±0.003	0.039±0.002
	SSIM	0.417±0.001	0.593±0.155	0.438±0.036
	PSNR	14.43±0.259	16.32±1.732	14.74±0.168

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

B.7 Extensive Comparison with Other Schemes

We provide a comprehensive comparison with prior SoTA works (Zhuang et al., 2021; 2022) in Table 13. 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 13: 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