

Appendix

A. Implementation

We implemented in eight ten SL/FL algorithms, including PSL, SFLV1, SFV2, SGLR, FedAvg (excluded from paper due to page limit), CyclePSL, CycleSFL, and CycleSGLR. All methods were implemented with PyTorch. An overview of our experiment environment is given in Table 7. Additionally we provided the code for sequential SL (SSL) and its cycle-version (CycleSSL). Since they are not scalable methods, we excluded them from benchmark. Our implementation can be found on <https://gitlab.lrz.de/hctl/CycleSL>. By following the instructions we provided, our experiment results should be completely reproducible up to numerical error.

B. Randomness

To ensure the robustness and reliability of results, we replicated the experiments for five times, with each time being initialized with a different and unique random seed from $\{0, 1, 2, 3, 4\}$. The random seed was fed to all libraries which can potentially be influenced by randomness in the beginning, including *numpy.random.seed*, *torch.manual.seed*, and *random.seed*. We reported all our metrics in form of mean \pm std over the five seeds.

C. Environment

The benchmarks were conducted on a computer with AMD EPYC 7763 and NVIDIA A100 80GB PCIe. All SL algorithms were implemented in PyTorch. An overview of the hardware and software of our environment is given in Table 7. It should be noticed that although we conducted the experiments on a powerful machine, we have tried to optimize our code so that it can be run on a normal PC as well, even without dedicated GPU.

Table 7. Experiment environment by July 12th, 2025.

Server	Specification	Library	Version
CPU	AMD EPYC 7763 64-Core	Python	3.13.5 by Anaconda
GPU	NVIDIA A100 80GB PCIe \times 4	PyTorch	2.7.1 for CUDA 12.8
RAM	1TB	Scikit-Learn	1.7.0
OS	Ubuntu 22.04.5 LTS	WandB	0.21.0

D. Server-side latency

As CycleSL performs feature resampling and feeds the smashed data into the server model twice, we were interested in quantifying the time cost incurred. On the CIFAR-100 dataset across different data heterogeneity levels, we measured the elapsed time on the server side between the arrival of smashed data and the transmission of gradients while ignoring the cost caused by model transfer between CPU and GPU. The recorded time cost was accumulated over SL rounds. We mainly compared three algorithms, namely SFLV1, SFLV2, and CycleSFL, as they are representative of aggregation-based, aggregation-free, and CycleSL respectively, and equivalent in regard of client behavior. The results are provided in Table 8. As the results show, the server processing time is rather consistent across various degrees of data heterogeneity. Among three methods, SFLV2 is the most time-efficient. SFV1 requires more computation time due to model aggregation. As CycleSL conducts feature resampling and a second time model feedforward, it incurs the highest server processing time. However, this time cost can be optimized with sophisticated choices for hyperparameters on the server side, especially batch size.

Table 8. Server-side processing time (in seconds).

Method	<i>iid</i>	$\alpha = 1.0$	$\alpha = 0.5$	$\alpha = 0.1$
SFLV1	19.3	19.7	19.1	19.9
SFLV2	10.4	10.4	10.7	10.3
CycleSFL	39.1	39.5	39.7	39.9

E. Toy example for gradient stability

Along side the empirical validation about gradient stability in subsection 4.3, we also provided a toy example in this regard in 1D. Consider a simplified regression task in SL where both the client and server part models are a single linear layer with one neuron. We further ignore activation functions and bias terms. Then, for a sample point (x, y) , its predicted value can be given as $\hat{y} = w_s w_c x$, where w_c and w_s are the parameters of client and server part models respectively. Consider mean squared error as a loss function, i.e., $\ell = (y - \hat{y})^2$. Traditionally, SL follows an end-to-end [15] pattern, which means both w_c and w_s are updated in the same gradient backward flow: $w'_s \leftarrow w_s - 2\eta w_c x (w_s w_c x - y)$, $w'_c \leftarrow w_c - 2\eta w_s x (w_s w_c x - y)$ where w'_c and w'_s are the updated parameters and $\eta > 0$ is learning rate. In contrast, CycleSL updates w_s and w_c one after another: $w'_s \leftarrow w_s - 2\eta w_c x (w_s w_c x - y)$, $w'_c \leftarrow w_c - 2\eta w'_s x (w'_s w_c x - y)$.

Comparing the two update strategies, it is clear that both are identical with respect to server side training, while they respectively use the old and the new server models to update the client part model. We now compare the two gradient steps for the client model, namely $2\eta w_s x (w_s w_c x - y)$ and $2\eta w'_s x (w'_s w_c x - y)$, when approaching convergence. For simplicity, we limit our discussion to the case where all $w_c, w_s, x, y > 0$ and $w_s w_c x > y$. All other cases can be analyzed similarly. Since $w_s w_c x > y$, we expect that with a proper choice of η , w'_s is reduced for a decently small step during server training, i.e., $\frac{y}{w_c x} < w'_s < w_s$ such that $w'_s w_c x$ shrinks towards y . Observe the function $f(w_s) = w_s x (w_s w_c x - y)$ and its derivative $f'(w_s) = 2w_s w_c x^2 - xy = x(w_s w_c x - y + w_s w_c x)$. When approaching convergence, i.e. $w_s w_c x - y \rightarrow 0^+$, we have $f'(w_s) > 0$, which means $f(w_s)$ is increasing in its neighborhood. Thus for $\frac{y}{w_c x} < w'_s < w_s$ the following applies: $f(w'_s) < f(w_s) \Leftrightarrow 2\eta w'_s x (w'_s w_c x - y) < 2\eta w_s x (w_s w_c x - y)$. That said, the alternating update strategy of CycleSL could possibly result in a smaller gradient step on the client side compared to the conventional end-to-end paradigm when approaching convergence, which may lead to stabler gradient steps.

F. Communication cost

As FL is more communication-costly in contrast to SL, some variants of FL have been proposed to reduce the model transmission overhead, such as knowledge distillation-based FL (KDFL, [31]) and partial training FL (PTFL [7, 13], also known as sub-model extraction FL). We gave a conceptual comparison of communication cost among standard FL, KDFL, PTFL, and SL (not necessarily CycleSL). Let N be the number of clients, M be the number of parameters in a full model, T be training rounds, B be batch size, D be the size of public/global dataset, L be the activation dimension at cut layer for SL, $0 < k < 1$ be the proportion of parameters shared in PTFL. Generally $L \ll M$ and $kM \ll M$. The communication cost is given in Table 9. It should be noted that these training paradigms commonly have many advanced variants [2, 56, 57, 62] and mechanisms can substantially change, and hence this comparison does not always apply.

Table 9. Communication cost comparison.

	FL	KDFL	PTFL	SL
Require public/server data	no	yes	no	no
Communication cost	$O(MT)$	$O(DT)$	$O(kMT)$	$O(BLT)$

G. Hyperparameters

For FEMNIST, CelebA, and Shakespeare, we followed the hyperparameters suggested by LEAF, especially the learning rates, unless they performed to be too small or too large in the experiment. For CIFAR-100 and OpenEDS2020, we kept batch size of 64 and conducted grid search for learning rate in range of $\{1e-5, 1e-4, 1e-3, 1e-2, 1e-1\}$ for each algorithm. The overall best performing learning rate was $1e-4$ and $1e-3.5$. An overview of hyperparameters is given in Table 10.

Table 10. Details of hyperparameters (consistent for clients and server across SL methods unless specifically mentioned).

Dataset	FEMNIST	CelebA	Shakespeare	CIFAR-100	OpenEDS2020
Batch Size	32	16	32	64	64
Optimizer	Adam	Adam	Adam	Adam	Adam
Learning Rate	$3e-4$	$1e-2$	$3e-2$	$1e-4$	$1e-3.5$

H. Model architecture

For FEMNIST, CelebA, and Shakespeare, we followed the model structures as suggested by LEAF (<https://github.com/TalwalkarLab/leaf/tree/master/models>), which are CNN, CNN, and LSTM, respectively. For the two CNN models, we cut them in the middle such that client and server parts have similar numbers of layers. For the LSTM model, we kept the embeddings and recurrent cells on the client while the projection head on the server. The model architectures, sources, and cut points are summarized in Tables 11–12 respectively.

Table 11. CNN architecture for the FEMNIST task.

Layer	Specification
Input	shape $1 \times 28 \times 28$
Conv2d	kernel size 5, in/out channel 1/32, same padding
ReLU	-
MaxPooling	kernel size 2, stride 2
Conv2d	kernel size 5, in/out channel 32/64, same padding
ReLU	-
MaxPooling	kernel size 2, stride 2
Cut Layer	client/server cut point
Flatten	-
Linear	in/out dimension 3136/2048
ReLU	-
Linear	in/out dimension 2048/62

Table 12. LSTM architecture for the Shakespeare task.

Layer	Specification
Embedding	number of embeddings 80, dimension 8
LSTM	3n/hidden dimension 8/256, hidden layers 2
Cut Layer	client/server cut point
Linear	in/out dimension 256/80

For the CIFAR-100 task, we adopted an ResNet9 network [19]. Our implementation followed <https://www.kaggle.com/code/kmlDas/cifar10-resnet-90-accuracy-less-than-5-min?scriptVersionId=38462746&cellId=28>). ResNet9 contains four convolutional blocks, two residual blocks, and a projection head. To balance the number of layers, we kept two convolutional blocks and one residual block on the client side, while the rest and the projection head on the server side. Implementation details can be found in <https://github.com/AnonymWriter/CycleSL/blob/main/models.py>. In the ablation study, we further investigated the influence of cut point on CycleSL. The corresponding results were discussed in subsection 4.3.

For the OpenEDS2020 task, we employed an appearance-based estimation model as suggested in [43]. The model consists of a pre-trained ResNet50-based feature extractor and a MLP head. We kept the feature extractor on the client side and the projection head on the server side.

I. Data distribution

The histograms of number of samples per client of the FEMNIST, CelebA, and Shakespeare datasets are given in Figure 2. Particularly, the CIFAR-100 dataset was partitioned with Dirichlet distribution using different α values to emulate different levels of data heterogeneity across clients (smaller α implies stronger data heterogeneity). The partition was done via FL-bench [52]. The impact of α on label distribution can be observed in Figure 3. OpenEDS2020 is considered as self-partitioned, as the data comes user-wise and each user was treated as a client in our experiments.

Table 13. CNN architecture for the CelebA task.

Layer	Specification
Input	shape $3 \times 84 \times 84$
Conv2d	kernel size 3, in/out channel 3/32, same padding
BatchNorm2d	-
MaxPooling	kernel size 2, stride 2
ReLU	-
Conv2d	kernel size 3, in/out channel 32/32, same padding
BatchNorm2d	-
MaxPolling	kernel size 2, stride 2
ReLU	-
Cut Layer	client/server cut point
Conv2d	kernel size 3, in/out channel 32/32, same padding
BatchNorm2d	-
MaxPolling	kernel size 2, stride 2
ReLU	-
Conv2d	kernel size 3, in/out channel 32/32, same padding
BatchNorm2d	-
MaxPolling	kernel size 2, stride 2
ReLU	-
Flatten	-
Linear	in/out dimension 800/2

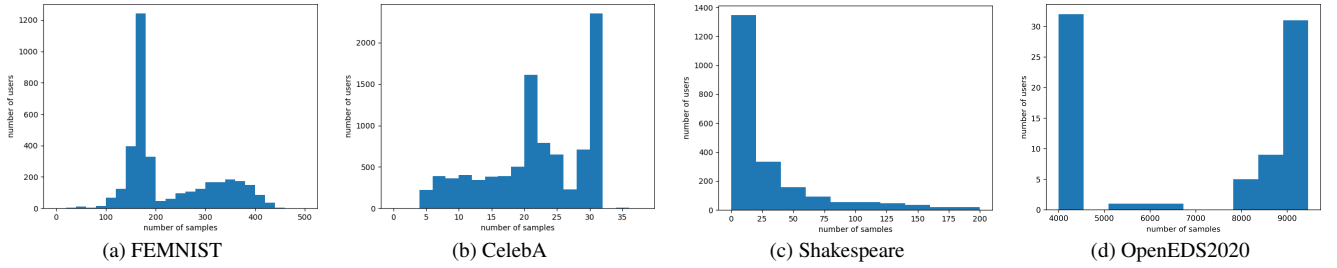


Figure 2. Histograms of samples per user for the FEMNIST, CelebA, Shakespeare, and OpenEDS2020 datasets.

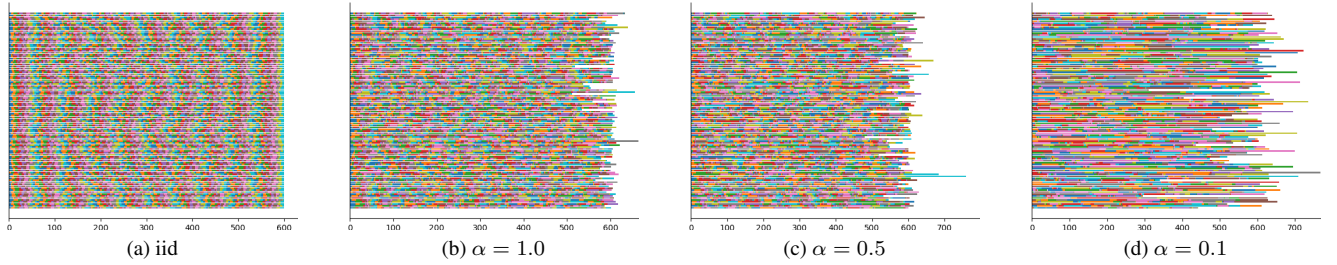


Figure 3. Label distributions among clients in CIFAR-100 (smaller α implies stronger data heterogeneity).

J. Additional results

J.1. Convergence rate

We measured the convergence speed of each algorithm by recording the first time that their test accuracies surpassed certain thresholds (45% for FEMNIST, 75% for CelebA, 35% for Shakespeare, 30% for CIFAR-100) in Table 14.

Table 14. Minimal epochs required to reach certain test accuracy for each task (45% for FEMNIST, 75% for CelebA, 35% for Shakespeare, 30% for CIFAR-100). Smaller is better.

Method	FEMNIST	CelebA	Shakespr	CIFAR _(iid)	CIFAR _($\alpha=1.0$)	CIFAR _($\alpha=0.5$)	CIFAR _($\alpha=0.1$)
PSL	459	255	> 600	> 1000	> 1000	> 1000	476
SGLR	517	244	> 600	> 1000	> 1000	> 1000	476
SFLV1	184	39	154	293	253	289	342
SFLV2	8	31	105	131	118	149	162
FedAvg	287	65	386	335	337	488	> 1000
CyclePSL	8	94	> 600	584	552	526	422
CycleSGLR	10	94	> 600	581	534	532	395
CycleSFL	6	29	98	116	107	132	162

J.2. Metric plots

The test metrics, including loss (cross entropy), accuracy, F1 score, and MCC (Matthews correlation coefficient), are plotted in Figures 4–10, respectively. It should be noticed that although some methods like PSL and SGLR overfitted for CelebA (increase in test loss), metrics like accuracy and F1 score were not negatively impacted. Hence we still reported metrics around 600th epoch.

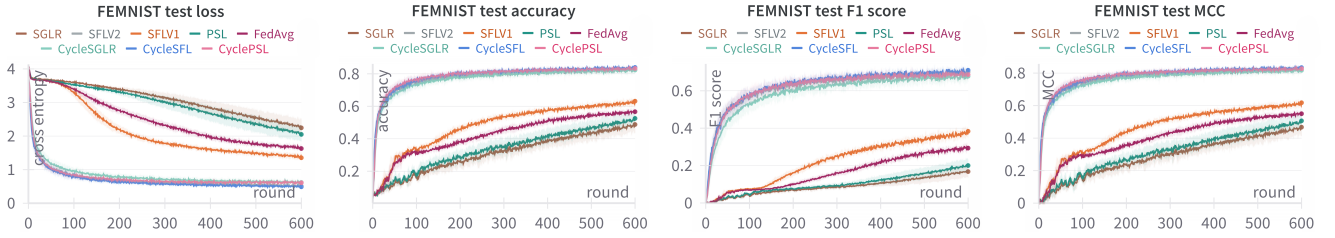


Figure 4. Test metrics for the FEMNIST task.

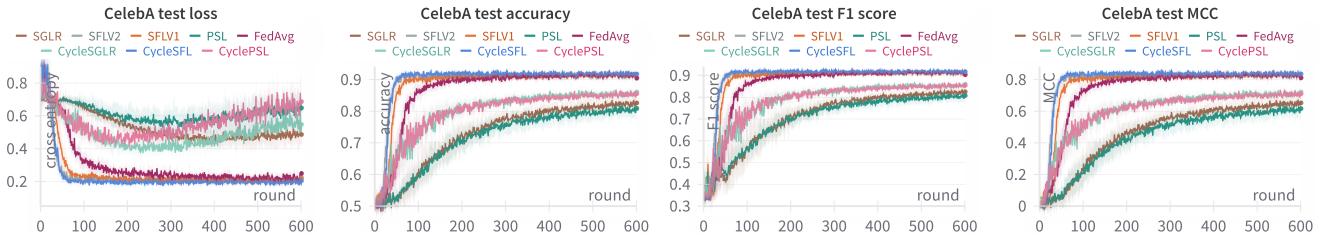


Figure 5. Test metrics for the CelebA task.

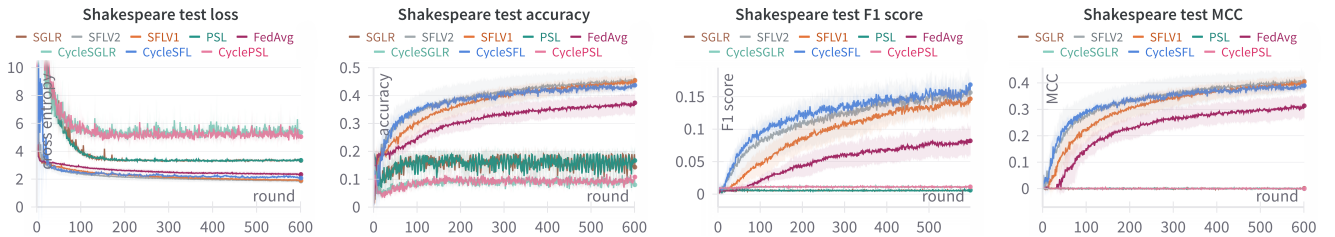


Figure 6. Test metrics for the Shakespeare task.

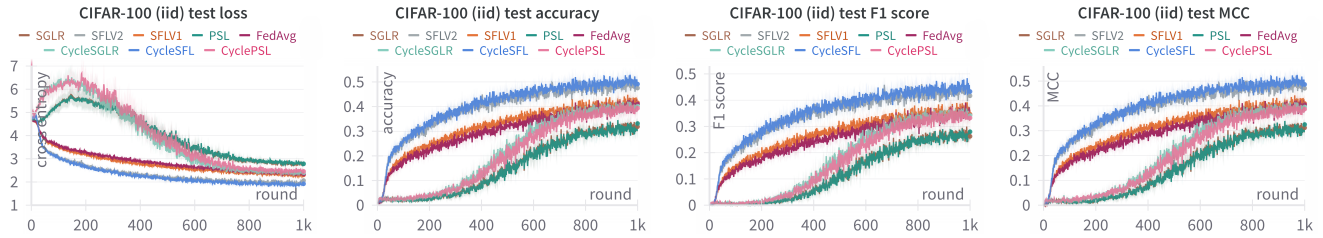


Figure 7. Test metrics for the CIFAR-100 task (iid).

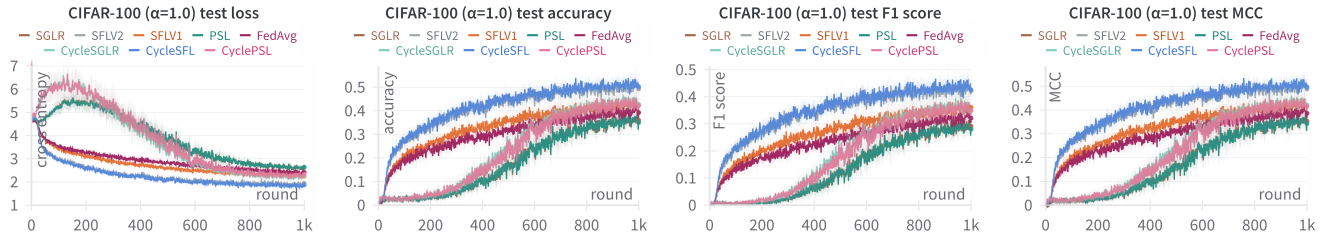


Figure 8. Test metrics for the CIFAR-100 task ($\alpha = 1.0$).

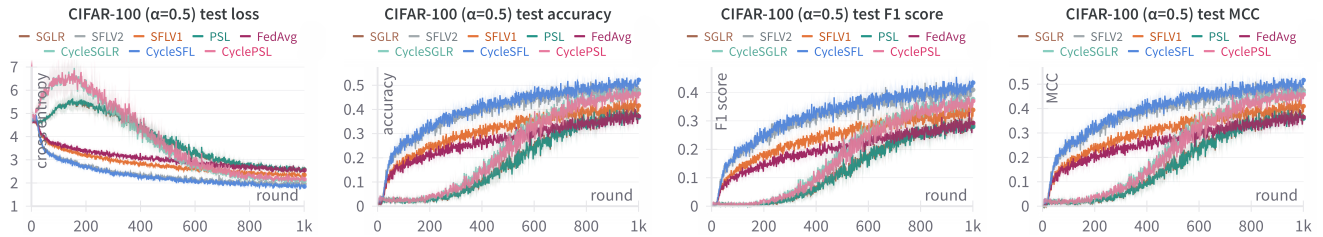


Figure 9. Test metrics for the CIFAR-100 task ($\alpha = 0.5$).

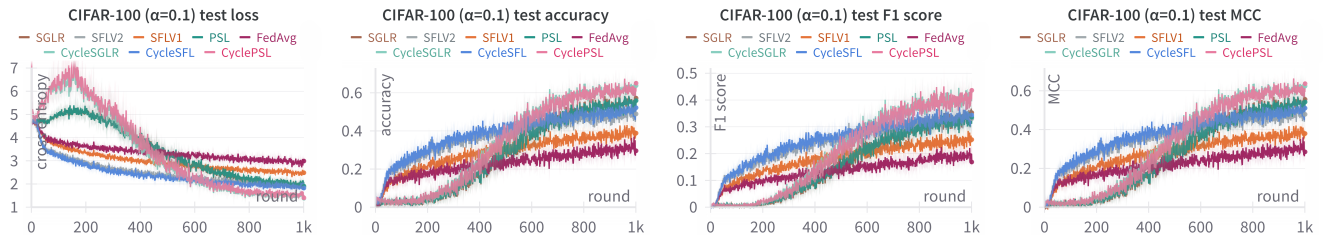


Figure 10. Test metrics for the CIFAR-100 task ($\alpha = 0.1$).

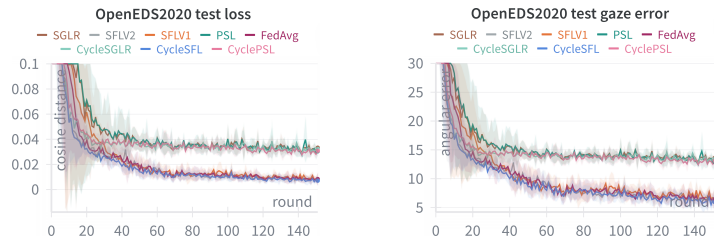


Figure 11. Test metrics for the OpenEDS2020 task.

J.3. Ablation study - impact of cut layer and server round

The impact of cut layer (block-wise) on CycleSFL test loss on the CIFAR-100 dataset is plotted in Figure 12. And the influence of server epoch on CycleSFL test loss on the CIFAR-100 dataset is visualized in Figure 13.

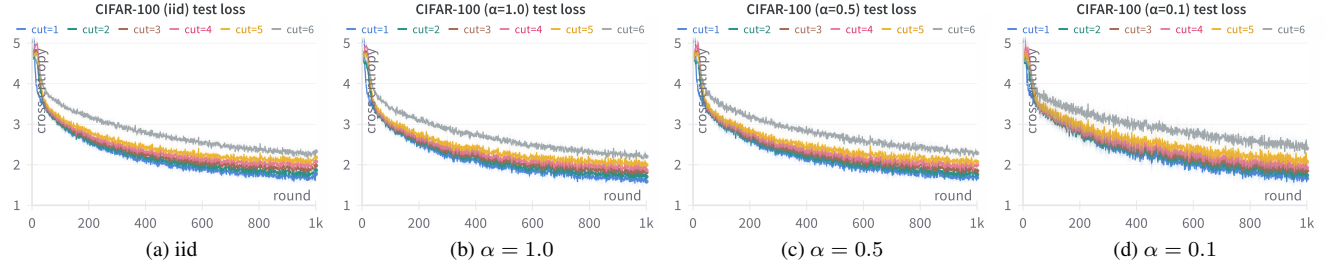


Figure 12. Impact of cut layer on CycleSFL test loss on CIFAR-100.

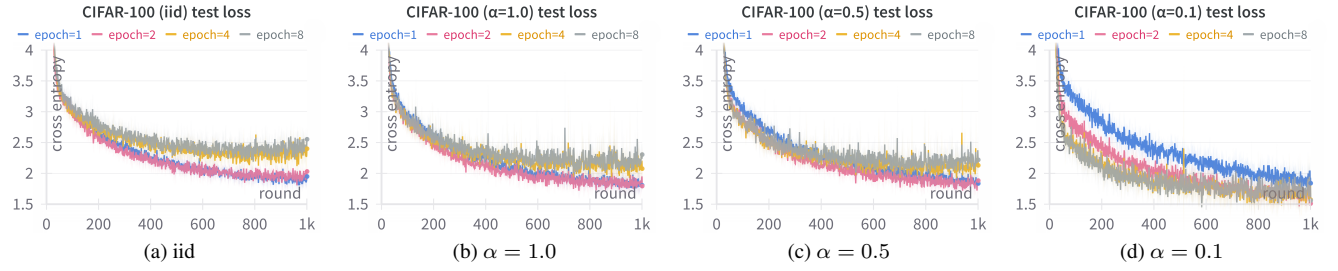


Figure 13. Impact of server epoch on CycleSFL test loss on CIFAR-100.