Split Federated Learning for AIGC with Resource Cognition in Collaborative Mobile Edge Computing

Oluwaseun T. Ajayi, and Yu Cheng
Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, 60616, USA
Email: oajayi6@hawk.iit.edu; cheng@iit.edu

***Abstract*—The era of artificial intelligence generated content (AIGC) has emerged to offer the benefits of personalized content to meet the needs of different users. A majority of generative AI models require computationally intensive training and/or fine-tuning with large datasets, which make resource-constrained devices, such as mobile edge devices (MEDs) to seldom participate in the training process. In addition, MEDs are not willing to share their private data with a central cloud server to train the models. Distributed machine learning (ML) approaches, such as federated learning (FL) and split learning (SL), offer some benefits in achieving data privacy, as well as model convergence for balanced data distributions. However, in practical settings where the datasets of MEDs are imbalanced, both FL and SL suffer poor performance. This motivates us to propose collaborative split federated learning (CSFL) for AIGC. Our CSFL approach supports resource cognition in which MEDs participate based on their available resources. We demonstrate that CFSL achieves good performance in terms of model convergence speed and training latency, while preserving data privacy when training a denoising diffusion probabilistic model for AIGC.**

***Index Terms*—AIGC, collaborating edge servers, CSFL, DDPM, mobile edge devices.**

I. Introduction

Most of the Web 3.0 applications are driven by artificial intelligence generated content (AIGC) technology, thus creating the possibility for humans to be immersed in their real-time interactive world of integrated text, image, video, and audio [1], [2]. The role of AIGC in revolutionizing content generation is envisioned to influence tremendous progress in different domains, such as fine arts, science, engineering, medicine, and telecommunication.

Despite the benefits of the popular generative AI models in creating high-quality content, AIGC service provision is still facing challenges as it requires huge financial costs that can vary between 4 million to 12 million US dollars [1] for model training, fine-tuning, and deployment, thus making it infeasible for users to generate more personalized contents, although they have available training data. The traditional approach of AIGC model training is to collect from many sources (e.g., public internet, crowdsourcing, customer data, etc.) all the training data in a central location and train the model in a central cloud server that has sufficient resources like high-end GPUs, TByte/PByte storage disk, and high-speed network interfaces. However, this traditional centralized learning approach poses a major concern on data privacy which is critical when collecting data to train the AIGC model, especially in a wireless network environment where an eavesdropper can secretly infer the raw data from the unsecured communication channel. Consequently, users may resist sharing their private local data, thereby reducing the diversity of data used for training the AIGC model [1].

Distributed machine learning (ML) approaches such as federated learning (FL) [3] and split learning (SL) [4] have been proposed to address the data privacy concerns. Both FL and SL train a client-side and a server-side model without the server having access to private local data held by the clients. The computation overhead with potential gradient leakage incurred by FL and the communication overhead by SL make neither one to be well-suited in mobile edge computing environment [5], [6]. In wireless networks, the selection of the cut layer for SL poses an optimization problem, especially when the number of resource-constrained devices is large [7]. Other distributed ML approaches that achieve data privacy, while trying to strike a balance between convergence speed and training latency have been proposed [8]–[10].

Split federated learning (SFL) was first proposed in [8] to mitigate the inherent drawbacks posed by FL and SL and has been adopted in training deep neural networks for image classification in computer vision. Recent works have focused on FL and SL to facilitate edge computing in IoT networks [11], [12] and spectrum sensing in cognitive radio networks [13]. However, SFL is envisioned to be a key learning framework for 6G edge intelligence [14], [15]. The work in [12] proposed a hybrid SFL algorithm in which a portion of user equipments (UEs) train an entire ML model, while the remaining UEs coordinate with the base station to train the ML model for image recognition in wireless unmanned aerial vehicle networks.

In [16], a federated split learning-based generative adversarial network (FSL-GAN) was proposed, where each client trains its own discriminator model on multiple devices to preserve its local data, while the generator model is trained by a central server. The challenges with FSL-GAN are that the remote server needs to share the generated images with the local clients to compute the loss, and clients always depend solely on the central server to generate images at inference time. In contrast to [16], the work in [1] conducted an FL-aided AIGC fine-tuning on the Stable Diffusion model. The FL framework was designed as a sequential learning procedure rather than parallel, which incurs huge computational overhead, storage requirement, and total energy consumption for the last client in the training procedure. Other issues include potential model

ICC 2025 - IEEE International Conference on Communications | 979-8-3315-0521-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICC52391.2025.11162066

poisoning attacks on the FL procedure.

In recognition of the data privacy, convergence speed, and training latency issues facing traditional distributed ML approaches, coupled with the resource constraints of devices that are willing to participate in AIGC model training, we explore a different paradigm of distributed ML in this paper. We propose a collaborative split federated learning (CSFL) approach for AIGC model training in a practical mobile edge computing environment using an imbalanced, independent and identically distributed (IID) data distribution. In CSFL, edge servers collaborate to train the resource-intensive part of an AIGC model, while the mobile edge devices (MEDs) only train a small portion of the model according to their available resources. To demonstrate the performance benefits of CSFL for image generation in AIGC, we trained a denoising diffusion probabilistic model (DDPM) [17] based on the CSFL approach and evaluated the training latency, convergence speed, and memory usage.

The closest to our work is [18] where a collaborative distributed computing approach is proposed for AIGC service execution among resource-constrained devices. The proposed framework in [18] only focused on the inference stage of a pre-trained Stable Diffusion model where only the reverse/denoising process is shared among the resource-constrained devices. However, in this paper, we consider a learning approach that effectively addresses the resource limitation issues for devices that wish to participate in the training process of AIGC models. At inference stage, the MEDs only need to coordinate with any of the collaborating edge servers (CESs) to perform the denoising process in a resource-efficient manner.

Our contributions can be summarized as follows:

- We propose a CSFL approach with resource cognition between the MEDs and CESs to address the resource limitation issues when participating in model training for AIGC, without violating data privacy.
- We address the problem of appropriately selecting the number of CESs in CSFL to improve performance during the training of the DDPM.
- We demonstrate with experiments and results that CSFL outperforms other distributed ML methods.

The remainder of the paper is organized as follows. Section II provides a preliminary on DDPM. Section III introduces the CSFL framework. Section IV details the training and inference process for AIGC in CSFL. Section V presents the experiments and results, while Section VI concludes the paper.

II. Preliminary on DDPM

Probabilistic generative AI models, such as variational autoencoders, GANs, and diffusion models, have shown high image quality synthesis results in AIGC research [17]. DDPM is a category of diffusion models that is based on a *forward process* and a *reverse (denoising) process* [19]. In the forward process, an uncorrupted data (e.g., image) is gradually perturbed by adding Gaussian noise over several steps, and in the reverse process, the original data is recovered by sequentially removing the noise from the perturbed data. At inference time, images are gradually reconstructed starting from a random Gaussian noise [19].

Generally, the *forward process* of a DDPM can be defined by a Markov chain with T-step transitions, i.e.:

$$q(x_{1:T} \mid x_0) := \prod_{t=1}^{T} q(x_t \mid x_{t-1}) \tag{1}$$

where $q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I})$ is the perturbed image at step t according to a variance schedule $\beta_t \in [0, 1)$.

By the reparameterization technique in [17], we can sample x_t in closed form at any arbitrary timestep t, i.e., $q(x_t \mid x_0)$ as follows:

$$x_t = \sqrt{\hat{\beta}_t} \cdot x_0 + \sqrt{(1-\hat{\beta}_t)} \cdot \epsilon_t \tag{2}$$

where $\epsilon_t \sim \mathcal{N}(0, \mathbf{I})$ is the Gaussian noise sampled from a standard normal distribution (with t being drawn from a uniform distribution), $\hat{\beta}_t = \prod_{s=1}^{t} \alpha_s$ is the cumulative noise up to timestep t, and $\alpha_t = 1 - \beta_t$. In this paper, we set $(\beta_t)_{t=1}^{T}$ to be linearly increasing constants, which helps to achieve training stability and faster convergence.

For the *reverse process*, from a noise sample $x_T \sim \mathcal{N}(0, \mathbf{I})$ we follow the reverse steps, i.e., $p_\theta(x_{t-1} \mid x_t)$ in which the noise component $\epsilon_\theta(x_t, t)$ to be subtracted at each time step is estimated by a trained neural network. The learning objective is thus, to minimize the loss L as follows:

$$L = \mathbb{E}_{t\sim[1,T]}\mathbb{E}_{x_0\sim q(x_0)}\mathbb{E}_{\epsilon_t\sim\mathcal{N}(0,\mathbf{I})}\|\epsilon_t - \epsilon_\theta(x_t, t)\|^2 \tag{3}$$

where $\epsilon_t \sim \mathcal{N}(0, \mathbf{I})$ and $\epsilon_\theta(x_t, t)$ are the true noise and the predicted noise at timestep t, respectively. At the final step where $x_{t-1} = x_0$, the original image is reconstructed.

The neural network used for noise prediction is typically a U-Net, which consists of a compression path through an encoder to capture context and a symmetric expanding path through a decoder that enables precise localization [20]. In our CSFL framework, we will demonstrate how to split the U-Net between the MEDs and CESs to facilitate efficient collaboration, while ensuring data privacy.

III. CSFL Framework

Consider an edge computing environment with K groups ($\mathcal{K} = \{G_1, \cdots, G_K\}$ and $K = |\mathcal{K}|$) of resource-constrained MEDs (e.g., smart devices), where $G_k = \{S_k, (k_1, \cdots, k_N)\}$ refers to a group of N devices that communicate with a CES S_k. For a given task $\mathcal{T}$, we want to train a U-Net-based noise predictor model $\mathbf{W}$, leveraging the data available from each MED k_n in all the groups. In CSFL, the full model $\mathbf{W}$ is split based on the available resources of the MEDs into $\mathbf{W}^C$ and $\mathbf{W}^S$, denoting the portion to be trained by the MEDs (i.e., client-side) and the CESs (i.e., server-side), respectively. We consider the following in our CSFL framework:

- The MEDs are grouped based on their real-time communication distance to the CESs.

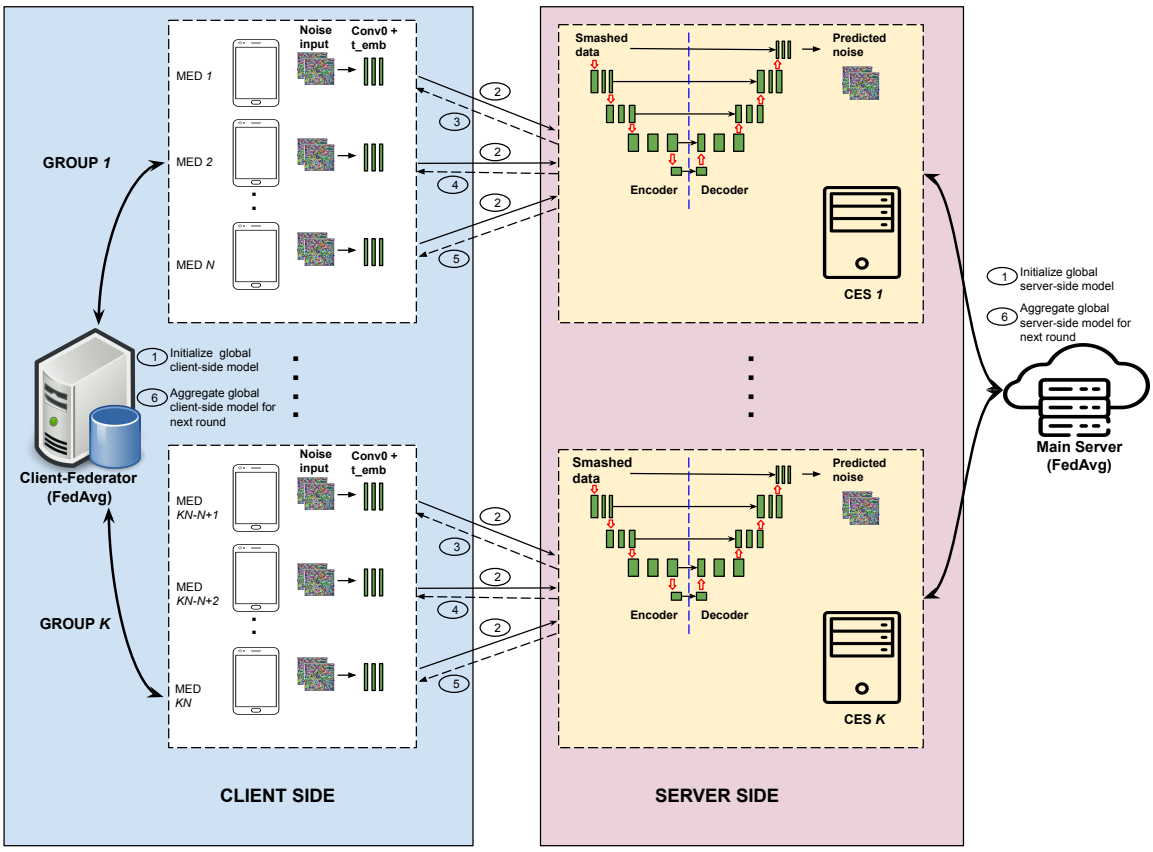


Fig. 1: Collaborative split federated learning framework. Parallel operations are denoted by solid lines, while sequential operations are denoted by broken lines. The number labels denote the chronological order of communication.

- All MEDs train the same number of layers $\mathbf{W}^C$ and all CESs train the same number of layers $\mathbf{W}^S$. Note that $|\mathbf{W}^C| < |\mathbf{W}^S|$ since the MEDs are resource-constrained.
- The data distribution is imbalanced among the groups and among the MEDs in each group.

The step-by-step process in CSFL as shown in Fig. 1 can be described as follows:

1) Steps 1 and 2: For the task $\mathcal{T}$, at round $r = 1$, a client-federator which performs FedAvg operations at the client-side intializes the global client-side model $\mathbf{W}^C$ and sends it to the MEDs in all the K groups, which then perform a computation pass on $\mathbf{W}^C$ in parallel using their private local data (i.e., images). The output of $\mathbf{W}^C$ is a smashed data D_s to be sent to the CES. Similarly, the main server initializes the global server-side model $\mathbf{W}^S$ for the CESs. The CES performs, in a sequential mode, the computation pass on $\mathbf{W}^S$ using D_s and calculates the loss.

2) Steps 3 to 5: In the sequential mode, the CES performs the backward pass on each D_s for the MEDs in its group, and sends the gradients $\nabla L(\mathbf{W}^S; D_s)$ to the MEDs to complete the backward pass to update their local $\mathbf{W}^C$.

3) Step 6: The client-federator then performs a weighted averaging on all the MEDs local updates as: $\mathbf{W}^C = FedAvg(\mathbf{W}^C_1, \cdots, \mathbf{W}^C_{KN})$ and then sends $\mathbf{W}^C$ to the MEDs for the next round $r+1$. Similarly, the main-server aggregates all the server-side updates as: $\mathbf{W}^S = FedAvg(\mathbf{W}^S_1, \cdots, \mathbf{W}^S_K)$ for the next round $r+1$. This process continues until the model converges or the budget limit is reached[1].

IV. Training and Inference for AIGC in CSFL

In this section, we present the training and inference procedures in CSFL, while taking into account the splitting of the U-Net based on the available resources of the MEDs.

A. Resource Cognition-based Model Splitting

Prior to model training, the MEDs communicate their available resources to their CESs. The entire model $\mathbf{W}$ is split based on $min(R_{1_{min}}, \cdots, R_{K_{min}})$, which is the minimum resource of clients in the network, where $R_{k_{min}}$ is the minimum resource of clients in group G_k. As illustrated by Fig. 1, the U-Net is split as: the client-side having just the initial convolution layer and time embedding block, while the server-side contains deep layers of encoder and decoder with residual connections.

As shown by the operational diagram of the U-Net in Fig. 2, the model takes as input both the noisy image $x_T \sim \mathcal{N}(0, \mathbf{I})$ and the time embedding vector t_{emb} (to capture the timestep

[1]This refers to resource budget, either in terms of computing or finances, since in AIGC, resource usage is a critical factor.

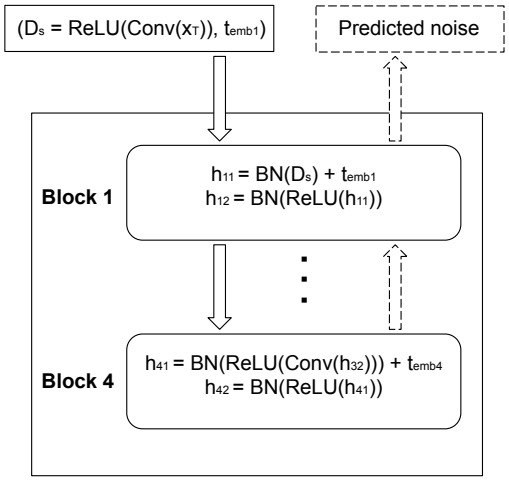


Fig. 2: Operations in the neural network blocks. Downward solid line arrows denote downsampling in the encoder and upward broken line arrows denote upsampling in the decoder.

information) and outputs the predicted noise. Within the encoder and decoder are four block instances for downsampling and upsampling, respectively[2]. Each block represents a series of convolutional (Conv) layers and batch normalizations (BN) with ReLU activations.

B. Training Stage

The MEDs perform their own forward (diffusion) process F_d on their local dataset $\mathcal{X}$ containing M samples of colored images, i.e., $F_d : \mathcal{X} \rightarrow \mathcal{X}_T \in \mathbb{R}^{M\times w\times h\times 3}$. To train the noise prediction model, each MED only performs an initial convolutional operation with ReLU activation on the noisy images $\mathcal{X}_T$ to get the smashed data D_s as shown in Fig. 2. The MED then sends its ground truth noise, smashed data, and the computed time-embedding vectors $\{(\epsilon^i_{t\sim[1,T]}, D^i_s, t^i_{emb})^M_{i=1}\}$ over a communication channel to its corresponding CES. Each CES uses the ground truth noise $(\epsilon^i_t)^M_{i=1}$ they get from their MEDs to calculate the loss and gradient w.r.t. the noise prediction $(\epsilon_\theta(D_s,t)^i)^M_{i=1}$ as:

$$L_{S_k,k_n} = \frac{1}{M}\sum_{i=1}^{M}\|\epsilon^i_t - \epsilon_\theta(D_s,t)^i\|^2, \quad \forall k_n \in G_k, \forall S_k \quad (4)$$

$$\nabla_\theta \mathbf{W}^{S_k}_{k_n} = \frac{1}{M}\sum_{i=1}^{M}\frac{\partial L_{S_k,k_n,i}}{\partial \epsilon_\theta(D_s,t)^i}\times\cdots\times D^i_s, \quad \forall k_n \in G_k, \forall S_k \quad (5)$$

From eq. (4) and eq. (5), the MED's original image x_0 and noisy version $x_{t\sim[1,T]}$ are concealed from the CES to ensure data privacy, since the edge server only sees the smashed data D_s, unlike eq. (3) which reveals x_t to the CESs or main server. Note that the t_{emb} is used by the server-side model to infer the timestep information during training. After the training loss converges, we have the client-side model $\mathbf{W}^C$ and the server-side model $\mathbf{W}^S$.

[2]The number of blocks can be increased to form a deep U-Net.

C. Inference (image generation) Stage

The MEDs randomly sample $x^s_T \sim \mathcal{N}(0,\mathbf{I})$ and pass it as input to $\mathbf{W}^C$. For each timestep t in the reverse process, the MEDs and CESs communicate [3] until $t = 0$, and the user can retrieve a clean image x^s_0. The sampling procedure defined by $p_\theta(x^s_{t-1} \mid x^s_t)$ is given in Algorithm 1. The sample x^s_{t-1} is obtained from the mean noise $\mu_\theta(x_t,t)$ and the posterior variance σ_t (which can be computed from $\beta_{t\sim[1,T]}$), i.e.:

$$\mu_\theta(x_t,t) = \frac{1}{\sqrt{\alpha_t}}\cdot\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\beta}_t}}\cdot\epsilon_\theta(D_s,t)\right) \quad (6)$$

Algorithm 1 Image generation in CSFL

Require: Establish a connection between MED and CES
Ensure: Load $\mathbf{W}^C$ for MED and $\mathbf{W}^S$ for CES
1: Initialize variables β_1, β_T, and T
2: $x^s_T \sim \mathcal{N}(0,\mathbf{I})$
3: **for** $t = T$ to $t = 1$ **do**
4: Client processes x^s_t through $\mathbf{W}^C$ and sends the smashed output D_s to CES
5: CES processes D_s through $\mathbf{W}^S$ to get the predicted noise $\epsilon_\theta(D_s,t)$
6: CES sends $\epsilon_\theta(D_s,t)$ to MED to compute $\mu_\theta(x_t,t)$
7: **if** $t = 1$ **then**
8: $\epsilon = 0$
9: **else**
10: $\epsilon \sim \mathcal{N}(0,\mathbf{I})$
11: **end if**
12: Client computes x^s_{t-1} as follows: $x^s_{t-1} = \mu_\theta(x_t,t) + \sigma_t\epsilon$
13: **end for**
14: **return** Generated image x^s_0

V. Experiments and Results

In this section, we present our experiment settings and results for CSFL. We simulate in a Python environment an edge computing scenario where the MEDs communicate with base stations. The base stations share the total system bandwidth equally, and they serve as the CESs. Note that the number of CESs cannot exceed the number of MEDs to avoid rapidly unstable selection of CESs by the MEDs. To train the DDPM for AIGC, we used the CIFAR 10 dataset which contains 10 classes of colored images: 50000 training images and 10000 test images. In the dataset, each class contains 5000 (and 1000) images for training (and testing), respectively, thus making it a balanced dataset.

Since our goal is to train the DDPM using imbalanced data distribution among MEDs, first, we combine the training data and test data. Then, we distribute the data among the MEDs, allowing some MEDs to hold data with majority of samples

[3]The communication overhead at inference is the tradeoff for data privacy. In our future work, we will explore techniques to reduce communication cost, while maintaining data privacy.

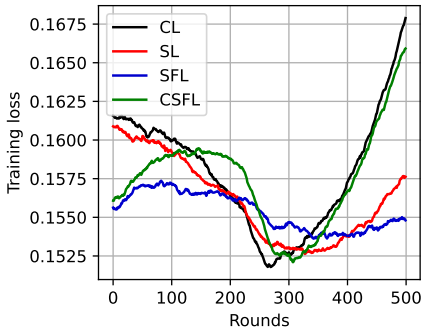


Fig. 3: Training loss comparison of CSFL with SL and SFL.

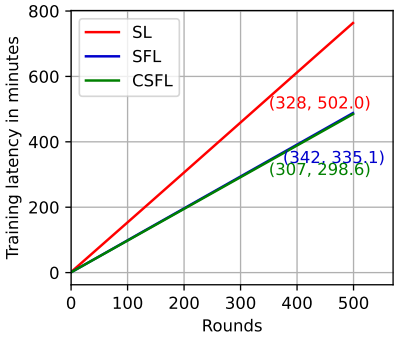


Fig. 4: Training latency comparison of CSFL with SL and SFL. The first value is the round in which the model converges and the second value is the corresponding latency.

belonging to a few classes. Our experiment is divided into two parts: (1) comparing CSFL with SL and SFL, and (2) comparison with FL. The parameter settings are listed in Table 1. We evaluate performance in terms of the loss convergence[4], training latency, and memory usage. The training latency is the sum of the training time and communication time between the MEDs and CESs.

TABLE I: List of parameters

Parameters	**Value**
Number of CESs	2
Resources of MEDs	CPU, and $<$ 64 GB storage
Resources of CESs	GPU, and $>$ 1 TB storage
Communication link speed	10.82 Mbps
T, β_1, and β_T	300, $1e^{-4}$, and $2e^{-2}$
Number of $\mathbf{W}^C$ parameters (size)	2848 (11.4 KB)
Number of $\mathbf{W}^S$ parameters (size)	62436035 (249.8 MB)
Learning rate	$1e^{-3}$
Optimizer	Adam
Batch size	32

A. Performance Comparison with SL and SFL

The distribution of the MEDs to the CESs is uniform and unique, i.e., the CESs coordinate equal number of MEDs and each MED is associated with only one CES throughout the training period. In our setting, the total number of MEDs (and CESs) is 10 (and 2), respectively. The minimum size of training data held by a MED is 1500 and the maximum size is 12000. In our experiment, we set the communication rounds (or simply rounds) to 500 and local epochs to 1. As shown in Fig. 3, CSFL converges to a loss of 0.1520 at 307 rounds (close to the centralized learning (CL) benchmark), while SL and SFL converge at 328 and 342 rounds with a loss of 0.1526 and 0.1536, respectively. The convergence speed of the DDPM training is critical in terms of the resource budget. The training latency is in the minutes order for the model to reach convergence. As shown in Fig. 4, it takes CSFL about 298 minutes to converge, and it takes SL and SFL 502 minutes and 335 minutes to converge, respectively. The SL's training latency is high due to the communication delay incurred since the MEDs need to wait in turn to send their smashed data to the main server. Although the training latency of SFL is close to that of CSFL, it is worthy to note that the training loss of SFL is higher as shown in Fig. 3.

[4]For the loss over the training rounds, we used the popular Savitzky-Golay filter to give a smoothed plot, which still preserves the original shape.

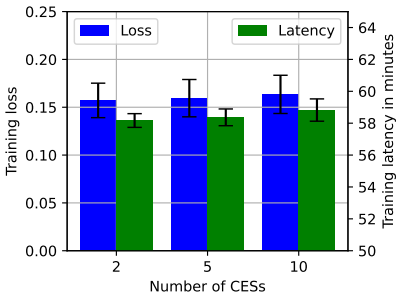
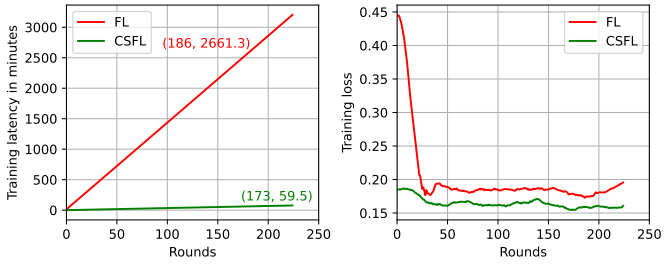


Fig. 5: Performance comparison with different numbers of CESs (average over 500 rounds of training).

We evaluate the performance of CSFL with varying number of CESs, keeping the number of MEDs fixed at 10. As shown in Fig. 5, setting the ratio γ $(0 \leq \gamma \leq 1)$ of CESs to MEDs to 1, increases the training loss and increases the convergence time. This is because a higher ratio of CESs to MEDs incurs more communication rounds for the training loss to converge. On the other hand, a lower ratio value like $\gamma = 0.1$ achieves faster training time, but suffer convergence issues as shown by the SFL in Figs. 4 and 3, respectively. To achieve a good performance on both training latency and loss convergence, our CSFL approach recommends $\gamma \in [0.2, 0.5]$.

B. Performance Comparison with FL

To compare our proposed CSFL approach with FL, we used $\gamma = 0.5$ (i.e., 2 CESs and 4 MEDs). The minimum size of training data held by a MED is 2583 and the maximum size is 8151. In contrast to CSFL where the entire model is split between the MEDs and CESs, in FL, the entire model is trained by the MEDs while the main server performs weighted averaging. The computation cost of training in FL is high compared to CSFL and still does not guarantee fast convergence of the training loss as shown in Figs. 6a and 6b, respectively. At 200 rounds, FL training loss starts to increase,



(a) Training latency comparison (b) Training loss comparison

Fig. 6: Comparison between CSFL and FL.

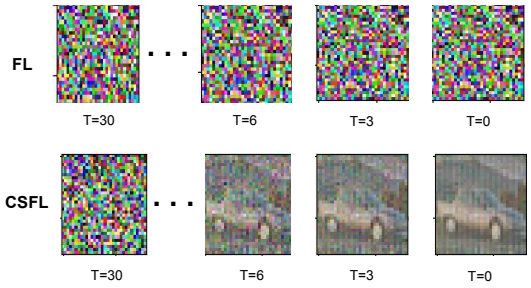


Fig. 7: Image generation quality comparison of FL and CSFL.

while CSFL training loss continues to decrease. We show using Fig. 7 the image generation quality by sampling from a pure Gaussian noise and performing the reverse/denoising steps to get an image. We show the denoised image from $T = 30$ to $T = 0$. FL performs poorly in generating an image, while CSFL generates a car image as observed at $T = 0$.

Regarding memory usage, FL does not offer benefits for resource-constrained MEDs. Although in our work the entire model size is $\approx$ 249.8 MB for 62 million parameters (which is possible for devices with very sufficient resources to train), for larger AIGC models with 100s of billion parameters, resource-limited MEDs cannot participate in the time-consuming diffusion and denoising processes. The memory (disk space/storage) requirement in CSFL for the MEDs is only 11.4 KB, unlike FL which requires 249.8 MB of disk space. While FL can influence users to free up disk space in order to participate in AIGC tasks, CSFL encourages participation even with limited storage capacity.

While our work did not explicitly evaluate performance in terms of power usage and device temperature, we can infer that the longer training time of SL and FL will incur more power usage and a high increase in device temperature.

VI. Conclusion

The cost of training and deploying AIGC models in mobile edge wireless networks have continued to raise concerns. In this paper, we have demonstrated how to train a DDPM for AIGC using our proposed CSFL approach to achieve faster convergence and reduced training overhead. Our work offers benefits in enriching AIGC models with diverse rich datasets from multiple MEDs, while ensuring data privacy.

Acknowledgments

This work was supported in part by the National Science Foundation (NSF) under Grant CNS-2008092.

References

[1] X. Huang, P. Li, H. Du, J. Kang, D. Niyato, D. I. Kim, and Y. Wu, "Federated learning-empowered AI-generated content in wireless networks," *IEEE Network*, 2024.

[2] J. Wu, W. Gan, Z. Chen, S. Wan, and H. Lin, "AI-generated content (AIGC): A survey," *arXiv preprint arXiv:2304.06632*, 2023.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, 2017, pp. 1273–1282.

[4] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[5] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtepe, H. Kim, and S. Nepal, "End-to-end evaluation of federated learning and split learning for internet of things," in *International Symposium on Reliable Distributed Systems (SRDS)*, vol. 4, 2020.

[6] Y. Gao, M. Kim, C. Thapa, A. Abuadbba, Z. Zhang, S. Camtepe, H. Kim, and S. Nepal, "Evaluation and optimization of distributed machine learning techniques for internet of things," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2538–2552, 2021.

[7] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, and W. Shi, "Split learning over wireless networks: Parallel design and resource management," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 4, pp. 1051–1066, 2023.

[8] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[9] S. Zhang, W. Wu, P. Hu, S. Li, and N. Zhang, "Split federated learning: Speed up model training in resource-limited wireless networks," in *Proceedings of IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, 2023, pp. 985–986.

[10] Y. Cheng, B. Yin, and S. Zhang, "Deep learning for wireless networking: The next frontier," *IEEE Wireless Communications*, vol. 28, no. 6, pp. 176–183, 2021.

[11] Q. Duan, S. Hu, R. Deng, and Z. Lu, "Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: State-of-the-art and future directions," *Sensors*, vol. 22, no. 16, p. 5983, 2022.

[12] X. Liu, Y. Deng, and T. Mahmoodi, "Wireless distributed learning: A new hybrid split and federated learning approach," *IEEE Transactions on Wireless Communications*, vol. 22, no. 4, pp. 2650–2665, 2022.

[13] Z. Chen, Y.-Q. Xu, H. Wang, and D. Guo, "Federated learning-based cooperative spectrum sensing in cognitive radio," *IEEE Communications Letters*, vol. 26, no. 2, pp. 330–334, 2021.

[14] Z. Lin, G. Qu, W. Wei, X. Chen, and K. K. Leung, "Adaptsfl: Adaptive split federated learning in resource-constrained edge networks," *arXiv preprint arXiv:2403.13101*, 2024.

[15] L. Wang, X. Wu, Y. Zhang, X. Zhang, L. Xu, Z. Wu, and A. Fei, "DeepAdain-net: Deep adaptive device-edge collaborative inference for augmented reality," *IEEE Journal of Selected Topics in Signal Processing*, vol. 17, no. 5, pp. 1052–1063, 2023.

[16] B. Yin, Z. Chen, and M. Tao, "Predictive gan-powered multi-objective optimization for hybrid federated split learning," *IEEE Transactions on Communications*, vol. 71, no. 8, pp. 4544–4560, 2023.

[17] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[18] H. Du, R. Zhang, D. Niyato, J. Kang, Z. Xiong, D. I. Kim, X. Shen, and H. V. Poor, "Exploring collaborative distributed diffusion-based AI-generated content (AIGC) in wireless networks," *IEEE Network*, vol. 38, no. 3, pp. 178–186, 2023.

[19] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, "Diffusion models in vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 10 850–10 869, 2023.

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proceedings of Medical image computing and computer-assisted intervention*, 2015, pp. 234–241.