GAS: Generative Activation-Aided Asynchronous Split Federated Learning

Jiarong Yang, Yuan Liu

School of Electronic and Information Engineering, South China University of Technology
eejryang@mail.scut.edu.cn, eeyliu@scut.edu.cn

Abstract

Split Federated Learning (SFL) splits and collaboratively trains a shared model between clients and server, where clients transmit activations and client-side models to server for updates. Recent SFL studies assume synchronous transmission of activations and client-side models from clients to server. However, due to significant variations in computational and communication capabilities among clients, activations and client-side models arrive at server asynchronously. The delay caused by asynchrony significantly degrades the performance of SFL. To address this issue, we consider an asynchronous SFL framework, where an activation buffer and a model buffer are embedded on the server to manage the asynchronously transmitted activations and client-side models, respectively. Furthermore, as asynchronous activation transmissions cause the buffer to frequently receive activations from resource-rich clients, leading to biased updates of the server-side model, we propose Generative activations-aided Asynchronous SFL (GAS). In GAS, the server maintains an activation distribution for each label based on received activations and generates activations from these distributions according to the degree of bias. These generative activations are then used to assist in updating the server-side model, ensuring more accurate updates. We derive a tighter convergence bound, and our experiments demonstrate the effectiveness of the proposed method.

Code — https://github.com/eejiarong/GAS

Introduction

Split Federated Learning (SFL) (Jeon and Kim 2020; Thapa et al. 2022) emerges as a promising solution for efficient resource-constrained distributed learning by combining the benefits of both Federated Learning (FL) (McMahan et al. 2017; Singh et al. 2022) and Split Learning (SL) (Gupta and Raskar 2018). Specifically, in SFL, the model is split into two parts: the initial layers are processed in parallel by the participating clients, and the intermediate activations are sent to the server, which completes the remaining layers. The server then sends the backpropagated gradients back to the clients, who use these gradients to update their client-side models. After several iterations, the server aggregates the client-side models to form the globally updated model.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Traditional SFL always assumes synchronous model exchange, where the server waits to receive all client-side models for aggregation. However, since clients have different communication and computational capabilities, client-side models are uploaded at the server asynchronously. The slow clients are referred to as stragglers, delaying the overall training process. Previous works in FL such as FedAsync (Xie, Koyejo, and Gupta 2019) and FedBuff (Nguyen et al. 2022) are proposed to tackle the stragglers issue by allowing clients to update the global model asynchronously. Additionally, CA^{2}FL (Wang et al. 2024) addresses convergence degradation by caching and reusing previous updates for global calibration, ensuring more consistent model updates despite asynchronous conditions.

However, the existing works on stragglers issue in SFL still have limitations. On one hand, the adaptive model splitting methods (Yan et al. 2023; Shen et al. 2023) for addressing this issue in SFL are constrained by the model structure. Specifically, these methods attempt to balance the arrival time of activations by selecting appropriate split layers of the model. Nevertheless, when the sizes of activations output by different model layers are similar or the computational and communication capabilities of clients are highly different, it is impossible to ensure simultaneous arrival of activations, regardless of the chosen split layers. On the other hand, the stragglers issue is more serious in SFL. Specifically, recent SFL methods (Huang, Tian, and Tang 2023; Yang and Liu 2024) assume synchronous activation transmissions with heterogeneous client data, where the uploaded activations are concatenated to update the server-side model centrally, reducing the bias in the deep layers of the model (Luo et al. 2021). However, these methods require the server to wait for stragglers to send their activations at the end of each local iteration, and the frequent transmissions of activations exacerbate the stragglers issue.

To address the above issues, we propose the asynchronous learning framework for SFL, where an activation buffer and a model buffer are embedded on the server to handle asynchronous updates. Specifically, the activation buffer stores the activations uploaded asynchronously. When the buffer is full, the server concatenates these activations and uses them to update the server-side model. Similarly, the model buffer stores the client-side models uploaded asynchronously. When the buffer is full, the server aggregates the

stored client-side models. By introducing the two buffers, we ensure efficient model updates and reduce delays caused by stragglers. However, due to the heterogeneous communication and computational capabilities of clients, the activation buffer may frequently receive activations from resource-rich clients, leading to biased updates in the server-side model (Leconte et al. 2024; Liu et al. 2024). To solve this issue, we propose Generative activation-aided Asynchronous SFL (GAS). Specifically, the server maintains a distribution of activations for each label, dynamically updated based on the uploaded activations. When updating the server-side model, we generate the activations from the distributions according to the degree of bias. Then these generative activations are concatenated with the stored activations to update the server-side model, thereby mitigating the model update bias introduced by stragglers. We summarize our contributions in this paper as follows:

- We propose an asynchronous SFL framework that enables the asynchronous transmissions of activations and client-side models. To our best knowledge, this is first attempt considering asynchronous SFL.
- We propose GAS (Generative activation-aided Asynchronous SFL), where the server updates the activation distribution for each label based on the uploaded activations and generates activations from the distributions to assist server-side model updates, mitigating the model update bias caused by stragglers.
- Several useful insights are obtained via our theoretical analysis: First, GAS can mitigate the gradient dissimilarity introduced by stragglers. Second, GAS achieves a tighter convergence bound. Third, by setting a decaying learning rate, the impact of stragglers can be gradually mitigated as the training progresses.

Related Works

Split Federated Learning

SFL (Thapa et al. 2022) combines the strengths of FL (McMahan et al. 2017) and SL (Gupta and Raskar 2018) to offer a more efficient and scalable learning framework. Recent research has explored various aspects of SFL. To enhance communication efficiency, FedLite (Wang et al. 2022a) employ compression techniques to reduce the volume of activation data transmitted. Simultaneously, the work by (Han et al. 2021) introduces auxiliary networks on the client side, eliminating the need for sending backpropagated gradients. In terms of privacy preservation, ResSFL (Li et al. 2022) and NoPeek (Li et al. 2022) implement attacker-aware training with an inversion score regularization term to counteract model inversion attacks. Additionally, the works by (Xiao, Yang, and Wu 2021) and (Thapa et al. 2022) leverage mixed activations and differential privacy to safeguard against privacy breaches from intermediate activations. To optimize performance for heterogeneous clients, SCALA (Yang and Liu 2024) and MiniBatch-SFL (Huang, Tian, and Tang 2023) employ activation concatenation and implement centralized training on the server, thereby enhancing model robustness and accuracy. Meanwhile, S^{2}FL (Yan et al. 2023) and RingSFL (Shen et al. 2023) address the stragglers issue by employing adaptive model splitting methods. Furthermore, recent works (Lin et al. 2024; Xu et al. 2023) refines SFL for real-world communication environments by selecting model split layers based on client channel conditions.

Asynchronous Federated Learning

Asynchronous FL addresses the limitations of traditional synchronous FL in heterogeneous environments, where "stragglers", or slow clients, can degrade overall training performance and efficiency (Wang et al. 2021). Early asynchronous FL frameworks (Xie, Koyejo, and Gupta 2019; Chen, Sun, and Jin 2019) mitigate the impact of stragglers by adaptively weighting the local updates. ASO-Fed (Chen et al. 2020) employs a dynamic learning strategy to adjust the local training step size, reducing the staleness effects caused by stragglers. FedBuff (Nguyen et al. 2022) introduces a buffering mechanism to temporarily store updates from faster clients, achieving higher concurrency and improving training efficiency. CA^{2}FL (Wang et al. 2024) further advances this approach by caching and calibrating updates based on data properties to handle both stragglers and data heterogeneity. FedCompass (Li et al. 2023) enhances efficiency by using a computing power-aware scheduler to prioritize updates from more powerful clients, thus reducing the waiting time for stragglers. FedASMU (Liu et al. 2024) addresses the stragglers issue through dynamic model aggregation and adaptive local model adjustment methods. Moreover, some works (Lee and Lee 2021; Wang et al. 2022b; Zhu et al. 2022; Hu, Chen, and Larsson 2023) have further optimizes the performance of asynchronous FL in wireless communication environments through staleness-aware model aggregation and client selection schemes.

Note that the asynchronously transmitted activations are concatenated rather than aggregated to update the server-side model in SFL, introducing unique challenges that make previous asynchronous FL methods inapplicable. Furthermore, the more frequent transmissions of activations exacerbate the stragglers issue. Existing SFL methods (Yan et al. 2023; Shen et al. 2023) employ adaptive model splitting to balance activation arrival times; however, they are constrained by the model structure. The above challenges highlight the need for further research to develop a tailored asynchronous framework for SFL. In this paper, we propose GAS to fill the gap by introducing a novel buffer mechanism and generative activations, which address the stragglers issue in SFL and achieve better model performance. Additionally, while CCVR (Luo et al. 2021) and FedImpro (Tang et al. 2024) also employ activation generation to enhance model performance, they require the additional transmission of local activation distributions. GAS distinguishes itself by leveraging the inherent characteristics of the SFL framework to dynamically update activation distributions using the activations continuously uploaded by clients, without incurring extra communication overhead.

Proposed Method

In this section, we systematically introduce GAS, which employs an activation buffer and a model buffer to enable asyn-

chronous transmissions of activations and client-side models, while leveraging generative activations to mitigate update bias caused by stragglers.

Preliminaries

Consider a SFL scenario involving K clients indexed by $\mathcal{K} = \{1, 2, \ldots, K\}$. Each client k holds a local dataset $\mathcal{D}_k$ with $|\mathcal{D}_k|$ data points. The clients collaborate to train a global model $\mathbf{w}$ under the coordination of the server. In SFL, the global model $\mathbf{w}$ is split into two parts: the client-side model $\mathbf{w}_c$ and the server-side model $\mathbf{w}_s$. The clients perform local computations on $\mathbf{w}_c$ and send the activations to the server, which completes the forward and backward passes using $\mathbf{w}_s$. Thus the empirical loss for client k is defined as

$$f_k(\mathbf{w}; \tilde{\mathcal{D}}_k) = l(\mathbf{w}_s; h(\mathbf{w}_c; \tilde{\mathcal{D}}_k)), \tag{1}$$

with h representing the client-side function that maps the sampled mini-batch data $\tilde{\mathcal{D}}_k$ to the intermediate activations, and l representing the server-side function that maps activations to the final loss value. We assume partial client participation and the primary objective is to minimize the global loss function over the participating clients $\mathcal{C}$, formulated as

$$\min_{\mathbf{w}} F(\mathbf{w}) = \frac{\sum_{k \in \mathcal{C}} |\mathcal{D}_k| F_k(\mathbf{w})}{\sum_{k \in \mathcal{C}} |\mathcal{D}_k|}, \tag{2}$$

where $F_k(\mathbf{w})$ is the local expected loss function for client k and it is unbiasedly estimated by the empirical loss $f_k(\mathbf{w}; \tilde{\mathcal{D}}_k)$, such that $\mathbb{E}_{\tilde{\mathcal{D}}_k \sim \mathcal{D}_k} f_k(\mathbf{w}, \tilde{\mathcal{D}}_k) = F_k(\mathbf{w})$.

Overall Structure

In Fig. 1, we illustrate the six key steps of GAS. The pseudocode is illustrated in Technical Appendix A. At the beginning of the training process, the server sets the number of local iterations E and global iterations T, with local iterations indexed by e and global iterations indexed by t. The server then initializes an activation buffer $\mathcal{A}$ to store the received activations and their corresponding labels, and a model buffer $\mathcal{M}$ to store the received client-side models along with the respective client data sizes. Next, the server sets the minibatch size to B, the activation buffer size to $Q_s B$, and the model buffer size to Q_c. Additionally, the server initializes the global model as $\mathbf{w}^0 = [\mathbf{w}_c^0, \mathbf{w}_s^0]$ and selects C initial clients to participate in the training. A detailed description of the training process follows.

- **Forward propagation of the client-side model (Fig. 1①):** The selected client k receives the client-side model and randomly selects a minibatch $\tilde{\mathcal{D}}_k$ with a batch size of B from its local dataset $\mathcal{D}_k$. The minibatch $\tilde{\mathcal{D}}_k$ is defined as $\tilde{\mathcal{D}}_k = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_B, y_B)\}$, where the input samples are $\mathbf{X}_k = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_B\}$ and their corresponding labels are $\mathbf{Y}_k = \{y_1, y_2, \ldots, y_B\}$. The client k then performs forward propagation using the client-side model $\mathbf{w}_c$ to compute the activations $\mathbf{A}_k$ of the last layer of the client-side model, given by

$$\mathbf{A}_k = h(\mathbf{w}_c; \tilde{\mathcal{D}}_k). \tag{3}$$

  Upon completing the computation, the activations $\mathbf{A}_k$ along with the label set $\mathbf{Y}_k$ are sent to the server.

- **Activations generation (Fig. 1②) and server-side model update (Fig. 1③):** The server receives activations from selected client k and stores them in the activation buffer as

$$\mathcal{A} \leftarrow \mathcal{A} \cup (\mathbf{A}_k, \mathbf{Y}_k). \tag{4}$$

  Additionally, the server maintains an activation distribution for each label, which is dynamically updated based on the received activations. (detailed in the next section). When the number of activations in the activation buffer exceeds $Q_s B$, the server generates activations $\widehat{\mathbf{A}}$ from the distribution. The generative activations $\widehat{\mathbf{A}}$ are then concatenated with the activations in the buffer as $\mathbf{A}_{\text{concat}} = \text{concat}(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{Q_s}, \widehat{\mathbf{A}})$. Similarly, the corresponding labels $\widehat{\mathbf{Y}}$ are concatenated with the labels in the buffer as $\mathbf{Y}_{\text{concat}} = \text{concat}(\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_{Q_s}, \widehat{\mathbf{Y}})$. Finally, the concatenated activations $\mathbf{A}_{\text{concat}}$ are used as input to update the server-side model:

$$\mathbf{w}_s^{e+1} = \mathbf{w}_s^e - \eta \nabla_{\mathbf{w}_s^e} l(\mathbf{w}_s^e; \mathbf{A}_{\text{concat}}, \mathbf{Y}_{\text{concat}}). \tag{5}$$

- **Backpropagation of server-side model (Fig. 1④)** The server computes the backpropagated gradients based on the received activations. As logit adjustment (Menon et al. 2021; Zhang et al. 2022) is popular for improving model performance under conditions of data heterogeneity, we apply it to calibrate the loss function of each client, as follows:

$$l_k(\mathbf{w}_s; \mathbf{A}_k, \mathbf{Y}_k) = -\log \left[\frac{e^{s_y} + \log P_k(y)}{\sum_{y'=1}^{M} e^{s_{y'} + \log P_k(y')}} \right], \tag{6}$$

  where s_y is predicted score for label y, $P_k(y)$ is the label distribution of client k and M is the total number of classes. Thus the backpropagated gradient is computed as

$$\mathbf{G}_k = \nabla_{\mathbf{A}_k} l_k(\mathbf{w}_s; \mathbf{A}_k, \mathbf{Y}_k), \tag{7}$$

  which is then sent to client k.

- **Backpropagation of client-side model (Fig. 1⑤):** The client k performs backpropagation using the received gradient and updates its local client-side model using the chain rule, given by

$$\mathbf{w}_{c,k}^{e+1} = \mathbf{w}_{c,k}^e - \eta \nabla_{\mathbf{A}_k} l_k(\mathbf{w}_s; \mathbf{A}_k, \mathbf{Y}_k) \nabla_{\mathbf{w}_{c,k}^e} h_k(\mathbf{w}_{c,k}^e; \mathbf{X}_k). \tag{8}$$

  When client k completes E local iterations, it sends the locally updated client-side model to the server.

- **Update of client-side model (Fig. 1⑥):** The server receives the updated client-side model and stores it in the model buffer as

$$\mathcal{M} \leftarrow \mathcal{M} \cup (\mathbf{w}_{c,k}, |\mathcal{D}_k|), \tag{9}$$

  When the number of models in the model buffer exceeds Q_c, the server aggregates these models as the current client-side model, given by

$$\mathbf{w}_c^{\text{agg}} = \frac{\sum_{(\mathbf{w}_{c,k}, |\mathcal{D}_k|) \in \mathcal{M}} |\mathcal{D}_k| \mathbf{w}_{c,k}}{\sum_{(\mathbf{w}_{c,k}, |\mathcal{D}_k|) \in \mathcal{M}} |\mathcal{D}_k|} \tag{10}$$

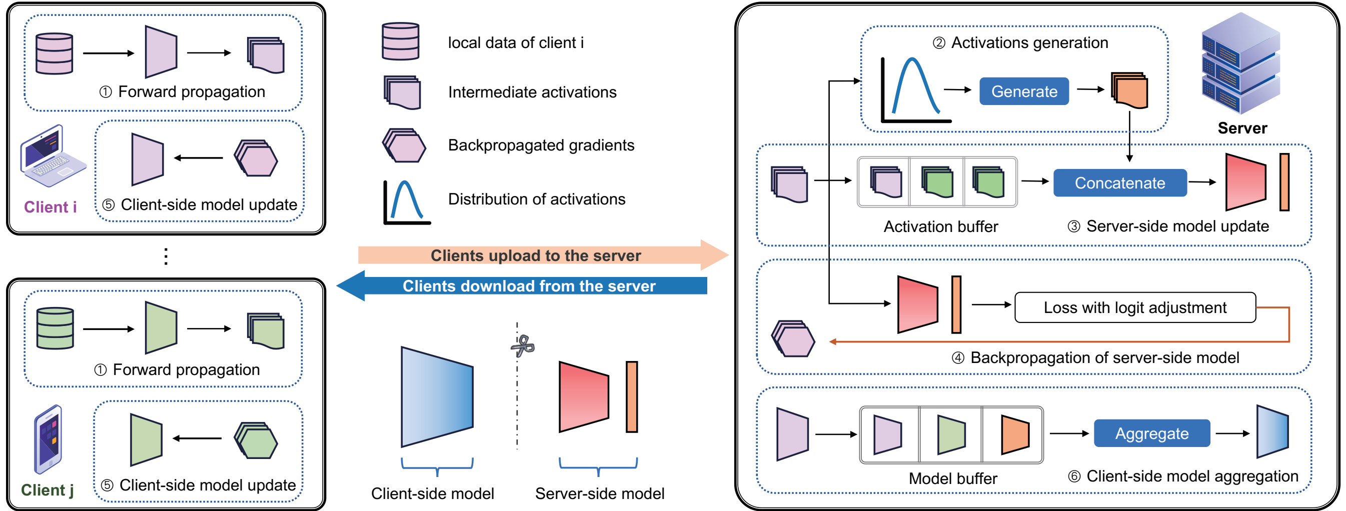


Figure 1: The framework of GAS. The client-side model is updated through four steps: ① Clients perform forward propagation; ④ The server receives the activations and computes backpropagated gradients; ⑤ Clients receive the gradients to update the client-side models, and complete a local iteration. After finishing local iterations, clients send the updated client-side models to the server; ⑥ The server stores these models in the model buffer and, when full, aggregates them to complete a global iteration. The server-side model is updated through two steps: ② Received activations update the distributions of activations. When the activation buffer is full, the server generate activations from these distributions; ③ Activations are stored in the buffer and, when full, the server concatenates them with generative activations to update the server-side model.

Then the server selects new client to participate in the training and sends it the current client-side model.

Note that the server-side model and the client-side models are not updated synchronously. Since the frequency of server-side model updates and client-side model aggregations is determined by the activation buffer size and the model buffer size, we typically set $Q_s = Q_c$ to ensure consistency in model updates. Additionally, to clarify the notation, we define a global iteration as E updates of the server-side model. After $E \times T$ iterations, the trained server-side model $\mathbf{w}_s^T$ is obtained. We define each aggregation of client-side models as a global iteration and after T aggregations, the trained client-side model $\mathbf{w}_c^T$ is obtained.

Generative Activation-Aided Updates

Due to the activations being uploaded asynchronously by selected clients, the activation buffer frequently receives activations from resource-rich clients. This results in a bias in the server-side model updates. To address this issue, we propose a method called Generative Activation-Aided Updates (Fig. 1② and Fig. 1③), where the server maintains the distribution of activations for each label y, represented as a Gaussian distribution $\mathcal{N}_y(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)$. The server generates activations from these distributions to assist in updating the server-side model. The key steps are as follows:

- **Dynamic Weighted Update:** The server dynamically updates the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ of the activation distribution using asynchronously uploaded activations in a weighted manner. Specifically, we define the weighting function as $s(n)$, where n represents the training progress, denoted by the total number of iterations $n = tE + e$. Since activations are uploaded asynchronously, each activation has a different training progress. We define $n(\mathbf{A})$ as the training progress of activation $\mathbf{A}$. The weighted mean for a training progress of N can be expressed as: $\boldsymbol{\mu}_N = \frac{1}{S_N}\sum_{\mathbf{A}\in\mathcal{A}_N} s(n(\mathbf{A}))\mathbf{A}$. And the weighted variance is given by $\boldsymbol{\Sigma}_N = \frac{1}{S_N}\sum_{\mathbf{A}\in\mathcal{A}_N} s(n(\mathbf{A}))(\mathbf{A}-\boldsymbol{\mu}_N)(\mathbf{A}-\boldsymbol{\mu}_N)^T$, where $\mathcal{A}_N$ denotes the set of all activations uploaded to the server up to training progress N, and S_N is the sum of the weights, defined as $S_N = \sum_{\mathbf{A}\in\mathcal{A}_N} s(n(\mathbf{A}))$. Since activations are dynamically uploaded, we adopt a dynamic update approach. Given a newly received activation $\mathbf{A}$, the mean is dynamically updated by

$$\boldsymbol{\mu}_N = \frac{S_{N-1}}{S_{N-1}+s(n(\mathbf{A}))}\boldsymbol{\mu}_{N-1} + \frac{s(n(\mathbf{A}))}{S_{N-1}+s(n(\mathbf{A}))}\mathbf{A}. \tag{11}$$

The variance is dynamically updated by

$$\boldsymbol{\Sigma}_N = \frac{S_{N-1}(\boldsymbol{\Sigma}_{N-1} + (\boldsymbol{\mu}_N - \boldsymbol{\mu}_{N-1})(\boldsymbol{\mu}_N - \boldsymbol{\mu}_{N-1})^T)}{S_{N-1}+s(n(\mathbf{A}))} + \frac{s(n(\mathbf{A}))(\boldsymbol{\mu}_N - \mathbf{A})(\boldsymbol{\mu}_N - \mathbf{A})^T}{S_{N-1}+s(n(\mathbf{A}))}. \tag{12}$$

The Derivation can be founded in Technical Appendix B.

- **Generating and Concatenation:** During the server-side model update, the server generates activations $\widehat{\mathbf{A}}$ by sampling from the distributions according to the skewness of the labels. For instance, the server adjusts the sampling

to ensure that each label has an equal amount of data. These generative activations are then concatenated with the activations in the activation buffer to form the input for updating the server-side model as (5). This method ensures that the server-side model receives a more balanced set of activations, mitigating the bias introduced by stragglers.

Note that we consider newer activations to be more important. Therefore, we define an increasing weighting function, such as an exponential function $s(n) = ae^{bn}$ or a polynomial function $s(n) = an^b$ (Xie, Koyejo, and Gupta 2019; Liu et al. 2024), where stale activations become less significant as training progresses, thereby mitigating the impact of stragglers on the activation distribution updates.

Theoretical Analysis

In this section, we provide a theoretical analysis to better understand the error bound and performance improvement of the proposed GAS. Since the server-side model and the client-side model are updated independently, where the parameters of one model remain fixed while the other is updated, we separately analyze the convergence rates of the server-side model and the client-side model. To ensure clarity, we denote $f_k(\mathbf{w}_c)$ as the local loss function of the client-side model $h(\mathbf{w}_c; \widetilde{\mathcal{D}}_k)$, and $f_s(\mathbf{w}_s)$ as the loss function of the server-side model $l(\mathbf{w}_s; \mathbf{A}_{\text{concat}}, \mathbf{Y}_{\text{concat}})$. Our analysis is based on the following assumptions:

Assumption 1. *(Smoothness) Loss function of server-side model and each local loss function of client-side model are Lipschitz smooth, i.e., for all* $\mathbf{w}$ *and* $\mathbf{w}'$, $\|\nabla_{\mathbf{w}_s} f_s(\mathbf{w}_s) - \nabla_{\mathbf{w}_s} f_s(\mathbf{w}_s')\| \leq \gamma_1 \|\mathbf{w}_s - \mathbf{w}_s'\|$ *and* $\|\nabla_{\mathbf{w}_c} f_k(\mathbf{w}_c) - \nabla_{\mathbf{w}_c} f_k(\mathbf{w}_c')\| \leq \gamma_2 \|\mathbf{w}_c - \mathbf{w}_c'\|$.

Assumption 2. *(Bounded Gradient Variance) The stochastic gradient of server-side model* $\nabla_{\mathbf{w}_s} f_s(\mathbf{w}_s)$ *and the stochastic gradient of client-side model* $\nabla_{\mathbf{w}_c} f_k(\mathbf{w}_c)$ *have bounded variance:* $\mathbb{E}[\|\nabla_{\mathbf{w}_s} f_s(\mathbf{w}_s) - \nabla_{\mathbf{w}_s} F_s(\mathbf{w}_s)\|^2] \leq \frac{\sigma^2}{BQ_s}$ *and* $\mathbb{E}[\|\nabla_{\mathbf{w}_c} f_k(\mathbf{w}_c) - \nabla_{\mathbf{w}_c} F_k(\mathbf{w}_c)\|^2] \leq \frac{\sigma^2}{B}$.

Assumption 3. *(Bounded Dissimilarity) In server-side model updates, gradient dissimilarity is referred to as the bias caused by stragglers, which is bounded as:* $\mathbb{E}\left[\|\nabla_{\mathbf{w}_s} F_s(\mathbf{w}_s) - \nabla_{\mathbf{w}_s} F(\mathbf{w}_s)\|^2\right] \leq \kappa_1^2$. *In client-side model updates, gradient dissimilarity is referred to as the bias caused by data heterogeneity across clients, which is bounded as:* $\mathbb{E}\left[||\nabla_{\mathbf{w}_c} F_k(\mathbf{w}_c) - \nabla_{\mathbf{w}_c} F(\mathbf{w}_c)||^2\right] \leq \kappa_2^2$.

In the proposed GAS, the activation distribution gradually approximates the ground-truth activation distribution through dynamic updates. This leads us to the following lemma:

Lemma 1. *By introducing generative activations, the server-side model update achieves a tighter bounded dissimilarity, as shown below:*

$$\begin{aligned}&\mathbb{E}_{\substack{(\mathbf{A},\mathbf{Y})\sim\mathcal{A}\\ \widehat{\mathbf{A}}\sim\mathcal{N}}}\left[\|\nabla_{\mathbf{w}_s} F_s(\mathbf{w}_s) - \nabla_{\mathbf{w}_s} F(\mathbf{w}_s)\|^2\right] \\ &\quad\leq \mathbb{E}_{(\mathbf{A},\mathbf{Y})\sim\mathcal{A}}\left[\|\nabla_{\mathbf{w}_s} F_s(\mathbf{w}_s) - \nabla_{\mathbf{w}_s} F(\mathbf{w}_s)\|^2\right]. \quad (13)\end{aligned}$$

The Proof can be founded in Technical Appendix C.

This reduction in gradient dissimilarity indicates that the server-side model update becomes less biased by concatenating generative activations. Now, we are ready to state the following theorem, which provides the convergence upper bounds for the proposed GAS, considering both the client-side model and the server-side model.

Theorem 1. *When Assumptions 1-3 hold, given the learning rate* $\eta \leq \frac{1}{\gamma_1}$, *the convergence rate of server-side model is given by*

$$\begin{aligned}&\frac{1}{ET}\sum_{t=0}^{T-1}\sum_{e=0}^{E-1}\mathbb{E}\left[\|\nabla_{\mathbf{w}_s} F(\mathbf{w}_s^{t,e})\|^2\right] \\ &\quad\leq \mathcal{O}\left(\frac{F(\mathbf{w}_s^0) - F^*}{ET\eta} + \frac{\eta\sigma^2}{BQ_s + \widehat{B}} + \kappa_1^2\right), \quad (14)\end{aligned}$$

where $\widehat{B}$ *is the batch size of generative activations.*

Given the learning rate $\eta \leq \frac{1}{20\gamma_2\sqrt{\tau_{\max}}}$ *and the maximum upload delay of client-side model* $\tau_{\max}$, *the convergence rate of client-side model is given by*

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla_{\mathbf{w}_c} F(\mathbf{w}_c^t)\|^2\right] \leq \mathcal{O}\Bigg(&\frac{F(\mathbf{w}_s^0) - F^*}{ET\eta} \\ &+ \left(\frac{\sigma^2}{B} + \kappa_2^2\right)\eta E + \left(\frac{\sigma^2}{B} + \kappa_2^2\right)\eta^2 E^2 \tau_{\max}^2\Bigg). \quad (15)\end{aligned}$$

The Proof can be founded in Technical Appendix D.

From (14), it is evident that stragglers primarily affect the convergence performance through their impact on the bounded dissimilarity of the server-side model κ_1^2. Specifically, if there is a bias in the activations stored in the activation buffer, the bounded dissimilarity increases, leading to an increase in κ_1^2. This, in turn, enlarges the convergence upper bound in (14). According to Lemma 1, the proposed method achieves a tighter bounded dissimilarity by introducing generative activations. As a result, the server-side model attains a tighter upper bound, enhancing convergence performance. From (15), it is evident that stragglers primarily affect convergence performance through $\tau_{\max}^2$, which is multiplied by the learning rate η. By setting a learning rate that decays over the global iterations, i.e., $\eta^t = \eta^0/\sqrt{t}$, the impact of $\tau_{\max}^2$ will be gradually mitigated as the training progresses.

Experiments

Implementation Details

Unless otherwise stated, the number of clients is set to 20, with 10 clients participating in each global iteration. Each client performs 20 local iterations with a learning rate of 0.01 and a minibatch size of 32. We use a linearly increasing weighting function, i.e., $s(n) = n$ and select AlexNet as the model architecture, where we set up the first 6 layers as the client-side model and the last 8 layers as the server-side model. To simulate a real-word communication environment, we consider a cell network with a radius of 1000 meters. The server is placed at the center of the network, with clients randomly and uniformly distributed within the

cell. The path loss between each client and the server is modeled as $128.1 + 37.6 \log_{10}(r)$ dB, where r is the distance from the client to the server in kilometers, according to (Abeta 2010). The client transmit power is uniformly set to 0.2 W. We assume orthogonal uplink channel access with a total bandwidth $W = 10$ MHz and a power spectrum density of the additive Gaussian noise $N_0 = -174$ dBm/Hz. Additionally, clients are assigned random computational capabilities, ranging between 10^9 and 10^{10} FLOPs. More experimental details can be founded in Technical Appendix E.

Baseline Settings

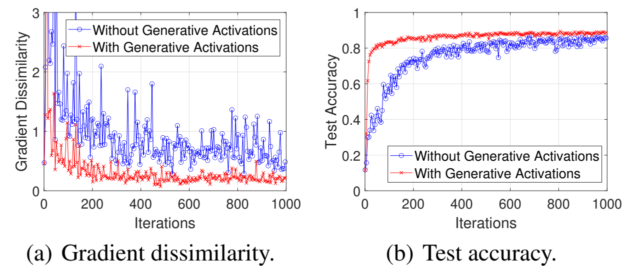
For the baseline comparison, we include both asynchronous and synchronous FL algorithms. The baseline asynchronous FL algorithms are FedBuff (Nguyen et al. 2022) and CA^{2}FL (Wang et al. 2024). FedBuff introduces a buffer mechanism to enable asynchronous FL, while CA^{2}FL builds on FedBuff by incorporating cached update calibration to enhance model performance in the presence of client data heterogeneity. Additionally, we select MiniBatch-SFL (Huang, Tian, and Tang 2023) and S^{2}FL (Yan et al. 2023) as baseline synchronous SFL algorithms. MiniBatch-SFL improves SFL performance by updating server-side model centrally, while S^{2}FL builds on MiniBatch-SFL by introducing adaptive model splitting and activation grouping strategies to address the stragglers issue.

Dataset Settings

The datasets used for evaluation include CIFAR-10 (Krizhevsky 2009), CINIC-10 (Darlow et al. 2018), and Fashion-MNIST (Xiao, Rasul, and Vollgraf 2017). To simulate data heterogeneous, we employ both shard-based and distribution-based label skew methods (Zhang et al. 2022). The shard-based method involves sorting data by labels and dividing it into multiple shards. Each client receives a subset of these shards, resulting in training data with only a few labels for each client. We denote the data heterogeneity of this method by shard, where shard $= 2$ indicates each client has at most 2 types of data. This method represents an extreme form of data heterogeneity. In addition, the distribution-based label skew method allocates data to clients based on a Dirichlet distribution. Each client receives a proportion of samples from each label according to this distribution, resulting in a mix of majority and minority classes, and potentially some missing classes. We denote the data heterogeneity of this method by α, where Dir(α) indicates the Dirichlet distribution. The smaller the value of α, the higher the degree of data heterogeneity. This method better reflects real-world data heterogeneity.

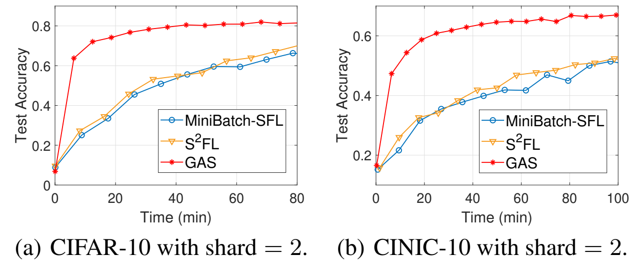
Validation of Theoretical Analysis

In this subsection, we validate our theoretical analysis by assessing gradient dissimilarity both without and with generative activations using the Fashion-MNIST datasets under heterogeneity conditions with shard $= 2$. The total number of clients is set to 10, with 3 clients participating in each global iteration. The experimental results are depicted in Fig. 2. As shown in Fig. 2 (a), the gradient dissimilarity is reduced by introducing generative activations, thereby confirming Lemma 1. This result demonstrates that the proposed method can achieve tighter bounded dissimilarity via the use of generative activations. Fig. 2 (b) further illustrates that the introduction of generative activations enhances convergence speed and achieve better accuracy, thus confirming Theorem 1. This indicates that tighter bounded dissimilarity reduces the upper bound of convergence rate, leading to superior convergence performance.



(a) Gradient dissimilarity. (b) Test accuracy.

Figure 2: Impact of generative activations on gradient dissimilarity and convergence performance.



(a) CIFAR-10 with shard = 2. (b) CINIC-10 with shard = 2.

Figure 3: Test accuracy of GAS compared with the baseline methods on CIFAR-10 and CINIC-10.

Performance Evaluation

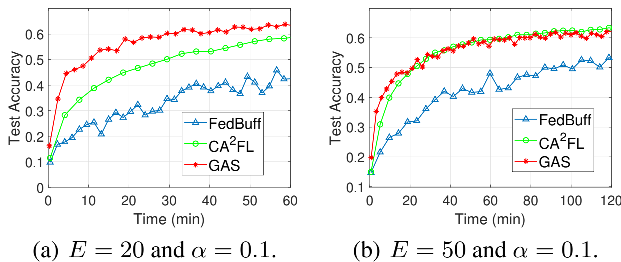
In this subsection, we evaluate the performance of the proposed method across different datasets and varying degrees of data heterogeneity. For Fashion-MNIST, CIFAR-10, and CINIC-10, we employ 1000, 2000, and 2000 global iterations. We first compare our method with baseline asynchronous FL algorithms. Each experiment is run with three random seeds, and the average accuracy and standard deviation are reported in Table 1. The experimental results demonstrate that the proposed method outperforms the baseline methods, particularly under conditions of extreme data heterogeneity. This improvement in model accuracy can be attributed to two key factors. First, the proposed method allows for centralized updates of the server-side model, significantly mitigating the issue of deep model drift caused by data heterogeneity (Luo et al. 2021). Second, by introducing generative activations, the proposed method alleviates the server-side model update bias introduced by stragglers, further enhancing model performance. Additionally, we compare our method with baseline synchronous SFL algorithms in a real-world communication environment, with results shown in Fig. 3. The experimental results indicate

Method	CIFAR-10		CINIC-10				Fashion-MNIST	
	s = 2	$\alpha = 0.1$	s = 2	s = 4	$\alpha = 0.1$	$\alpha = 0.3$	s = 2	$\alpha = 0.1$
FedAvg	$72.88_{\pm 5.71}$	$70.49_{\pm 4.24}$	$52.66_{\pm 6.51}$	$62.26_{\pm 2.52}$	$57.17_{\pm 1.04}$	$65.46_{\pm 2.09}$	$87.99_{\pm 2.12}$	$88.74_{\pm 1.19}$
FedBuff	$69.04_{\pm 3.51}$	$71.82_{\pm 2.86}$	$48.98_{\pm 0.87}$	$58.32_{\pm 2.20}$	$54.23_{\pm 2.11}$	$64.69_{\pm 1.81}$	$84.93_{\pm 4.11}$	$85.81_{\pm 3.68}$
CA^{2}FL	$79.57_{\pm 0.98}$	$78.56_{\pm 0.99}$	$64.29_{\pm 1.45}$	$68.42_{\pm 1.11}$	$64.27_{\pm 0.78}$	$68.77_{\pm 0.92}$	$88.32_{\pm 1.19}$	$89.07_{\pm 0.58}$
Ours	$\mathbf{82.78}_{\pm 0.58}$	$\mathbf{81.72}_{\pm 0.50}$	$\mathbf{68.32}_{\pm 0.17}$	$\mathbf{70.29}_{\pm 0.27}$	$\mathbf{65.94}_{\pm 1.14}$	$\mathbf{69.36}_{\pm 0.65}$	$\mathbf{90.66}_{\pm 0.20}$	$\mathbf{90.58}_{\pm 0.34}$

Table 1: Test accuracy (%) on CIFAR-10, CINIC-10 and Fashion-MNIST.

Method	shard = 2				$\alpha = 0.1$			
	$E = 10$	$E = 20$	$E = 35$	$E = 50$	$E = 10$	$E = 20$	$E = 35$	$E = 50$
FedBuff	$41.52_{\pm 2.48}$	$40.07_{\pm 1.42}$	$44.80_{\pm 4.09}$	$47.84_{\pm 5.69}$	$44.87_{\pm 3.75}$	$51.18_{\pm 1.74}$	$54.03_{\pm 5.01}$	$55.77_{\pm 3.58}$
CA^{2}FL	$58.77_{\pm 0.52}$	$62.12_{\pm 1.39}$	$63.14_{\pm 0.28}$	$63.31_{\pm 1.18}$	$58.39_{\pm 1.15}$	$60.95_{\pm 0.84}$	$62.02_{\pm 2.39}$	$62.73_{\pm 1.16}$
Ours	$\mathbf{63.07}_{\pm 0.08}$	$\mathbf{65.58}_{\pm 0.71}$	$\mathbf{65.09}_{\pm 1.10}$	$\mathbf{62.40}_{\pm 1.97}$	$\mathbf{60.68}_{\pm 1.13}$	$\mathbf{63.39}_{\pm 2.01}$	$\mathbf{63.12}_{\pm 2.29}$	$\mathbf{61.96}_{\pm 3.50}$

Table 2: Test accuracy (%) under different number of local iterations.



(a) $E = 20$ and $\alpha = 0.1$. (b) $E = 50$ and $\alpha = 0.1$.

Figure 4: Impact of local iterations on the performance of GAS compared to baseline methods.

that the proposed method exhibits better convergence performance compared to baseline methods. This improvement is due to the asynchronous transmissions of activations and client-side models, which substantially reduce training time and achieves faster convergence speeds.

Ablation Study on Local Iterations

In this subsection, we conduct an ablation study on the number of local iterations. Unlike FL frameworks, GAS requires the additional transmissions of activations, which are influenced by the number of local iterations. Therefore, we study the impact of different local iteration settings under the real-world communication environment. We conduct experiments with local iteration settings of 10, 20, 35 and 50, while fixing the number of global iterations at 1000. The results are presented in Table 2 and Fig. 4. As shown in Table 2, the accuracy of GAS increases with the number of local iterations initially but decreases thereafter. This indicates that the number of local iterations must be carefully chosen to balance model accuracy and communication load. On one hand, a higher number of local iterations is necessary for sufficient local training. On the other hand, setting the number of local iterations too high can lead to local optima and increased communication load. Additionally, we observe that the accuracy of the baseline methods increase with the number of local iterations. This suggests that the baseline methods do not achieve sufficient training within the given local iteration settings. Even with a local iteration setting of 50, the accuracy of CA^{2}FL remains lower than that of GAS with a local iteration setting of 20, indicating the higher training efficiency of GAS.

From Fig. 4, it is evident that GAS demonstrates faster convergence and higher accuracy compared to the baseline methods at the lower local iteration setting ($E = 20$). This highlights the significant advantage of GAS in real-world communication environments. Note that although CA^{2}FL performs well with $E = 50$, it incurs higher computational load due to the increased number of local iterations. Specifically, CA^{2}FL takes 60 minutes to achieve 60% accuracy, whereas GAS with $E = 20$ achieves the same accuracy in just 30 minutes.

Conclusion

In this paper, we proposed GAS (Generative activation-aided Asynchronous SFL), a distributed asynchronous learning framework designed to address the stragglers issue in SFL. By employing an activation buffer and a model buffer, along with generative activation-aided updates, GAS effectively mitigated the impact of stragglers and improved model convergence. Our theoretical analysis and experimental results demonstrated that GAS achieved higher accuracy and faster convergence compared to baseline FL and SFL methods.

Limitations: Like other SL and SFL algorithms, GAS requires the transmission of labels and activations, which poses a risk of privacy leaks. Incorporating privacy-preserving mechanisms of SFL (Xiao, Yang, and Wu 2021; Li et al. 2022) into GAS to enhance data security and broaden its applicability is a promising direction for future work.

References

Abeta, S. 2010. Evolved Universal Terrestrial Radio Access (EUTRA); Further advancements for E-UTRA physical layer aspects. Technical report, Technical report (TR) 36.814. 3GPP.

Chen, Y.; Ning, Y.; Slawski, M.; and Rangwala, H. 2020. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, 15–24. IEEE.

Chen, Y.; Sun, X.; and Jin, Y. 2019. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE transactions on neural networks and learning systems*, 31(10): 4229–4238.

Darlow, L. N.; Crowley, E. J.; Antoniou, A.; and Storkey, A. J. 2018. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*.

Gupta, O.; and Raskar, R. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116: 1–8.

Han, D.-J.; Bhatti, H. I.; Lee, J.; and Moon, J. 2021. Accelerating federated learning with split learning on locally generated losses. In *ICML 2021 workshop on federated learning for user privacy and data confidentiality. ICML Board*.

Hu, C.-H.; Chen, Z.; and Larsson, E. G. 2023. Scheduling and aggregation design for asynchronous federated learning over wireless networks. *IEEE Journal on Selected Areas in Communications*, 41(4): 874–886.

Huang, C.; Tian, G.; and Tang, M. 2023. When minibatch sgd meets splitfed learning: Convergence analysis and performance evaluation. *arXiv preprint arXiv:2308.11953*.

Jeon, J.; and Kim, J. 2020. Privacy-sensitive parallel split learning. In *2020 International Conference on Information Networking (ICOIN)*, 7–9. IEEE.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. *Master's thesis, University of Tront*.

Leconte, L.; Jonckheere, M.; Samsonov, S.; and Moulines, E. 2024. Queuing dynamics of asynchronous Federated Learning. In *International Conference on Artificial Intelligence and Statistics*, 1711–1719. PMLR.

Lee, H.-S.; and Lee, J.-W. 2021. Adaptive transmission scheduling in wireless networks for asynchronous federated learning. *IEEE Journal on Selected Areas in Communications*, 39(12): 3673–3687.

Li, J.; Rakin, A. S.; Chen, X.; He, Z.; Fan, D.; and Chakrabarti, C. 2022. Ressfl: A resistance transfer framework for defending model inversion attack in split federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10194–10202.

Li, Z.; Chaturvedi, P.; He, S.; Chen, H.; Singh, G.; Kindratenko, V.; Huerta, E. A.; Kim, K.; and Madduri, R. 2023. FedCompass: efficient cross-silo federated learning on heterogeneous client devices using a computing power aware scheduler. *arXiv preprint arXiv:2309.14675*.

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

Liu, J.; Jia, J.; Che, T.; Huo, C.; Ren, J.; Zhou, Y.; Dai, H.; and Dou, D. 2024. Fedasmu: Efficient asynchronous federated learning with dynamic staleness-aware model update. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 13900–13908.

Luo, M.; Chen, F.; Hu, D.; Zhang, Y.; Liang, J.; and Feng, J. 2021. No fear of heterogeneity: Classifier calibration for federated learning with non-iid data. *Advances in Neural Information Processing Systems*, 34: 5972–5984.

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, 1273–1282. PMLR.

Menon, A. K.; Jayasumana, S.; Rawat, A. S.; Jain, H.; Veit, A.; and Kumar, S. 2021. Long-tail learning via logit adjustment. In *International Conference on Learning Representations*.

Nguyen, J.; Malik, K.; Zhan, H.; Yousefpour, A.; Rabbat, M.; Malek, M.; and Huba, D. 2022. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*, 3581–3607. PMLR.

Shen, J.; Cheng, N.; Wang, X.; Lyu, F.; Xu, W.; Liu, Z.; Aldubaikhy, K.; and Shen, X. 2023. Ringsfl: An adaptive split federated learning towards taming client heterogeneity. *IEEE Transactions on Mobile Computing*.

Singh, P.; Singh, M. K.; Singh, R.; and Singh, N. 2022. Federated learning: Challenges, methods, and future directions. In *Federated Learning for IoT Applications*, 199–214. Springer.

Tang, Z.; Zhang, Y.; Shi, S.; Tian, X.; Liu, T.; Han, B.; and Chu, X. 2024. Fedimpro: Measuring and improving client update in federated learning. *arXiv preprint arXiv:2402.07011*.

Thapa, C.; Arachchige, P. C. M.; Camtepe, S.; and Sun, L. 2022. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 8485–8493.

Wang, J.; Charles, Z.; Xu, Z.; Joshi, G.; McMahan, H. B.; Al-Shedivat, M.; Andrew, G.; Avestimehr, S.; Daly, K.; Data, D.; et al. 2021. A field guide to federated optimization. *arXiv preprint arXiv:2107.06917*.

Wang, J.; Qi, H.; Rawat, A. S.; Reddi, S.; Waghmare, S.; Yu, F. X.; and Joshi, G. 2022a. Fedlite: A scalable approach for federated learning on resource-constrained clients. *arXiv preprint arXiv:2201.11865*.

Wang, Y.; Cao, Y.; Wu, J.; Chen, R.; and Chen, J. 2024. Tackling the Data Heterogeneity in Asynchronous Federated Learning with Cached Update Calibration. In *The Twelfth International Conference on Learning Representations*.

Wang, Z.; Zhang, Z.; Tian, Y.; Yang, Q.; Shan, H.; Wang, W.; and Quek, T. Q. 2022b. Asynchronous federated learning

over wireless communication networks. *IEEE Transactions on Wireless Communications*, 21(9): 6961–6978.

Xiao, D.; Yang, C.; and Wu, W. 2021. Mixing activations and labels in distributed training for split learning. *IEEE Transactions on Parallel and Distributed Systems*, 33(11): 3165–3177.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xie, C.; Koyejo, S.; and Gupta, I. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*.

Xu, C.; Li, J.; Liu, Y.; Ling, Y.; and Wen, M. 2023. Accelerating split federated learning over wireless communication networks. *IEEE Transactions on Wireless Communications*.

Yan, D.; Hu, M.; Xia, Z.; Yang, Y.; Xia, J.; Xie, X.; and Chen, M. 2023. Have Your Cake and Eat It Too: Toward Efficient and Accurate Split Federated Learning. *arXiv preprint arXiv:2311.13163*.

Yang, J.; and Liu, Y. 2024. SCALA: Split Federated Learning with Concatenated Activations and Logit Adjustments. *arXiv preprint arXiv:2405.04875*.

Zhang, J.; Li, Z.; Li, B.; Xu, J.; Wu, S.; Ding, S.; and Wu, C. 2022. Federated learning with label distribution skew via logits calibration. In *International Conference on Machine Learning*, 26311–26329. PMLR.

Zhu, H.; Zhou, Y.; Qian, H.; Shi, Y.; Chen, X.; and Yang, Y. 2022. Online client selection for asynchronous federated learning with fairness consideration. *IEEE Transactions on Wireless Communications*, 22(4): 2493–2506.