FedACA: An Adaptive Communication-Efficient Asynchronous Framework for Federated Learning

Shuang Zhou, Yuankai Huo, Shunxing Bao, Bennett Landman and Aniruddha Gokhale
Dept of Computer Science and Dept of Electrical and Computer Engineering
Vanderbilt University, Nashville, TN, USA
{shuang.zhou,yuankai.huo,shunxing.bao,bennett.landman,a.gokhale}@vanderbilt.edu

***Abstract*—Federated Learning (FL) is a type of distributed machine learning, which avoids sharing privacy and sensitive data with a central server. Despite the advances in FL, current approaches cannot provide satisfactory performance when dealing with heterogeneity in data and unpredictability of system devices. First, straggler devices can adversely impact convergence speed of the global model training. Second, for model aggregation in traditional FL, edge devices communicate frequently with a central server using their local updates. However, this process may encounter communication bottleneck caused by substantial bandwidth usage. To address these challenges, this paper presents an adaptive, communication-efficient and asynchronous FL technique called FedACA comprising feedback loops at two levels. Our approach contains a self-adjusting local training step with active participant selection to accelerate the convergence of the global model. To reduce the communication overhead, FedACA supports an adaptive uploading policy at the edge devices, which leverages the model similarity and L2-norm differences between the current and previous local gradient. It also utilizes contrastive learning to tackle data heterogeneity by regularizing the local training if the local model has deviated from the global model and helps with the model similarity measurement in the uploading policy. Extensive experiments on a benchmark comprising three image datasets with non-independent and identically distributed (non-i.i.d) data show that FedACA adapts well to the straggler effect in asynchronous environments and also provides significant reductions in communication costs compared to other state-of-the-art FL algorithms.**

***Index Terms*—Federated Learning, Communication Efficiency, Self-adaptation, Straggler Mitigation, Contrastive Learning**

I. Introduction

Deep learning models are data hungry—they perform better when trained on a large and representative dataset [1]. However, increasingly data are isolated at distributed locations with relatively smaller sizes (e.g., medical images or data collected from mobile devices). Collecting and sharing data between different facilities is often impractical due to data privacy concerns and intellectual property issues [2], [3].

To that end, Federated Learning (FL) [4], [5] has emerged as a form of distributed training that enables multiple edge devices (clients) to jointly participate in learning a model without exchanging local data among each other. Moreover, FL aims to fit a global model across diverse edge devices without continuously transferring massive amounts of collected data between edge devices of the network or from edge to back-end servers for processing.

There, however, is one major challenge in contemporary FL solutions. For example, FedAvg [5] and its extensions [4], [6], [7] use a synchronous protocol where all the updates are incorporated for global model aggregation in each global epoch. This synchronous approach can become less efficient since the central server needs to wait for all updates from clients [8]. Due to heterogeneity in the system and unpredictable network delays, manifestation of lagging devices (i.e., stragglers, stale workers) is inevitable. These laggers not only result in unpredictably long waiting times for the server but also significantly slow down convergence of the global model.

To address this challenge, prior work has proposed asynchronous FL techniques [8]–[10], which allow the server to perform model aggregation without waiting for the slow devices to complete their local computation. However, a number of issues still remain unresolved. First, the model aggregation step may magnify the straggler effect under the situation where updated models are trained from different global epochs of which some updates may be too stale to provide up-to-date model parameters.

Second, asynchronous FL requires that edge devices send a full model update back to the server. For large models, communication bottleneck can occur due to asymmetric transmission speeds in the internet: the uplink speed is typically much slower than downlink speed. For instance, the US median speed for mobile devices was reported as 61.06 Mbps download vs. 8.40 Mbps upload [11]. Unlike synchronous FL where only a certain fraction of selected edge devices participate in each global iteration, the bottleneck becomes worse in asynchronous FL approaches since the number of participants is not stable: there remains a chance that the server can easily become a communication bottleneck when large numbers of clients update the model simultaneously.

Third, selecting participants (i.e., clients or edge devices) for FL is challenging. Each participant locally processes its own non-i.i.d (non-independent and identically distributed) data which can degrade the performance of FL [12]. In current client selection algorithms [5], [13], the server selects a fraction of edge devices randomly or depending on their value. However, such approaches neglect the asynchronous condition: any non-delayed edge device has a higher opportunity to be chosen since it is always under the state of idle (not working) and others will always be acted as laggers. Unbalanced selection can result in model deviations from the target distribution.

To overcome these challenges, we ask the following questions: *Can we develop an asynchronous federated learning* *framework with communication efficiency? Can the approach be robust and mitigate the effect of stragglers when the network status is not stable?* To that end, this paper proposes **FedACA**: an Adaptive, Communication-efficient and Asynchronous FL technique by focusing on improving accuracy performance and reducing communication costs in asynchronous FL under the circumstance when there are unpredictable transmission latencies in the network.

To achieve communication efficiency, we utilize the bandwidth between the edge devices and servers effectively. In contrast to existing schemes [5] where edge devices reply with the actual gradient to the server, in our approach, the edge devices will not send their actual weights if they are deemed to be a non-informative update due to failing to reach a predefined threshold. We develop a dynamic strategy for computing the threshold value using contrastive learning and computing the norm of the difference between the models. Communication efficiency is also improved by properly selecting the participants: if the edge devices are more informative, they have a higher chance to contribute in later rounds.

Moreover, compared to other asynchronous FL methods which aggregate the global model as soon as receiving an update, our approach lets the central server start aggregation at any time by leaving a certain percentage of the edge clients who are deemed as "laggers" (i.e., stragglers). To handle these lagging devices due to overloads and/or network latencies, we introduce a time-based aggregation method which introduces a factor to represent the timeliness of the devices. Finally, we also propose an adaptive protocol to choose the local step size for each edge device to eliminate the chance that some of them always get determined as laggers.

We present FedACA, a novel adaptive, communication-efficient and asynchronous federated learning framework that updates the global model asynchronously across epochs. The main contributions in this paper are:

- We propose a series of strategies for adaptively controlling the stragglers in asynchronous situations by introducing a time-related factor to mitigate the impact of stragglers on model aggregation across global epochs.
- We propose a dynamic local epoch adaptation strategy by letting the server determine the local epoch on each device to balance the training time on every edge device.
- To solve the non-i.i.d data issues and achieve communication efficiency, we design a new learning strategy and update policy using model similarity computation based on contrastive learning since edge devices only transmit their updates if they satisfy the informative threshold.
- We evaluate FedACA on three datasets suitable for federated learning including CIFAR-10, CIFAR-100 and Fashion-MNIST. We show FedACA outperforms state-of-the-art FL solutions in convergence, is robust to the presence of straggler devices, and is communication efficient through reduction in communication cost by at least 75% to reach the target accuracy.

The rest of the paper is organized as follows: In Section II, we preliminarily introduce related works and compare FedACA with prior efforts; The proposed method, FedACA, is presented in Section III; Section IV provides details of our experimental setup; In Section V, we present the results of our proposed scheme on three datasets; and Section VI concludes this paper along with its limitations and directions for future work.

II. Background and Related Work

In this section, we provide a brief background on synchronous FL and asynchronous FL. We then present related efforts comparing them to our work on FedACA.

A. Federated Learning

FL is a type of distributed machine learning where privacy preservation of data at local sites is critically important. There are two primary types of FL:

Synchronous Federated Learning: The most common type of FL uses the synchronous approach. Federated averaging (FedAvg) [5], a classic case of synchronous FL, was designed to perform synchronous optimization in federated settings. FedAvg operates in the following manner: First, a central server distributes the current global model to the edge devices (clients). Second, the clients then update their models locally by independently performing E epochs of training the global model on their local datasets with the stochastic gradient descent (SGD) optimizer. Third, the local models are subsequently sent to a central server. Last, the server averages the model weights and repeats the steps above.

Asynchronous Federated Learning: Asynchronous training [6] [14] is widely used in traditional distributed learning to eliminate the influence of stragglers [6], [14], [15]. Recently, several works have taken advantage of asynchronous training and combined it with federated learning terming this approach as "Asynchronous FL." Compared to the synchronous FL, asynchronous FL is much more efficient in making the best use of all clients' computational resources.

Wait-free communication and computation [9], [16], [17] are some recent asynchronous FL approaches that address the straggler problem. However, these frameworks spontaneously lead to higher communication frequency which results in higher bandwidth usage and thereby costs. Also, handling asynchronous procedures may cause unnecessary storage usage. For example, [10] introduces their server-side optimizations by using tier-based weight strategies. However, the server needs to keep and maintain the updates from each tier in order to average the updated weight, which increases storage costs.

Compared to these efforts, in FedACA we study a collaborative, asynchronous FL framework to achieve bandwidth and storage savings by only updating with the most informative models when the server starts a new global epoch randomly.

B. Communication Efficiency

Approaches to achieving communication efficiency can be summarized into two categories: (1) reducing the communication frequency and (2) reducing the communication costs, particularly to the uplink.

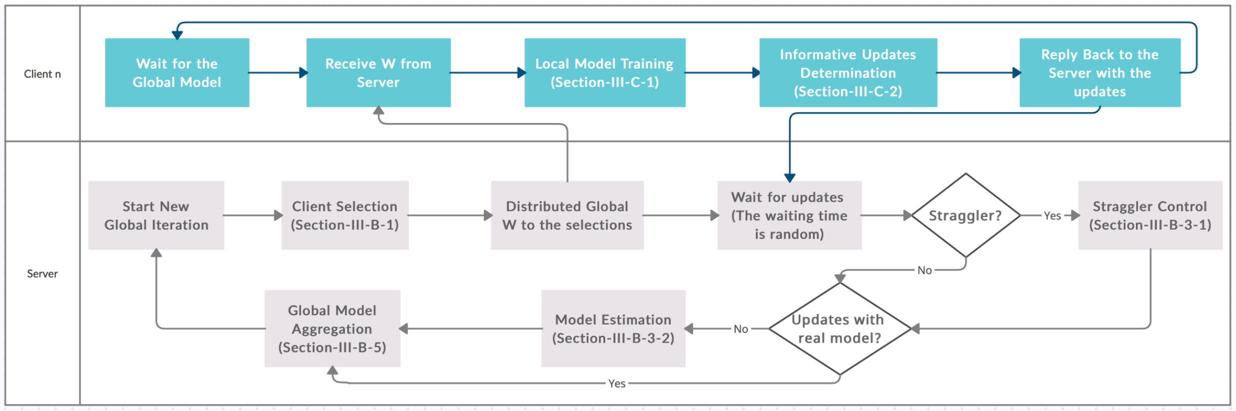

Client n
Wait for the Global Model
Receive W from Server
Local Model Training (Section-III-C-1)
Informative Updates Determination (Section-III-C-2)
Reply Back to the Server with the updates
Server
Start New Global Iteration
Client Selection (Section-III-B-1)
Distributed Global W to the selections
Wait for updates (The waiting time is random)
Straggler?
Yes
Straggler Control (Section-III-B-3-1)
No
Updates with real model?
No
Model Estimation (Section-III-B-3-2)
Global Model Aggregation (Section-III-B-5)
Yes

Fig. 1: Illustration of the Update Procedure in FedACA between the server and one client. The server aggregates after waiting for a random time and shows client n as one of the selected clients.(Note: The time when client n replies may not be in the current global iteration)

To reduce the communication frequency, threshold and node selection methods are widely used independently or jointly. The work in [5] and its extensions [4], [6], [7] reduce communication frequency by working with partial client participation. Threshold methods are used by computing a threshold metric (i.e., a control variable) [18], [19]. Only the clients who satisfy this threshold are permitted to upload their model.

To reduce the communication overhead in FL, a primary approach is to minimize the model parameter size in uplink. Model compression at the clients for uploading is one of the widely used methods. Approaches like Sparsification [20], and quantization methods [21], [22] can be used to reduce the communication overhead. Another method is to upload a subpart of the upload model parameters [23], [24].

These prior efforts are developed only for synchronous frameworks and do not take the straggler issues into consideration. They still face longer waiting times and slower convergence speeds. Our algorithm aims to find a solution to address communication efficiency using asynchronous FL.

C. Contrastive Learning of Representations

Contrastive learning is widely used in semi-supervised or unsupervised learning. SimCLR [25] is a typical contrastive learning framework where the key idea is to reduce the distance between the representations of positive pairs (e.g., different augmented views of the same image), and increase the distance between the representations of negative pairs (e.g., augmented views from different images).

That is, given an image x, SimCLR creates two augmented views (i,j) of image x: x_i, x_j. In the model, a base encoder network $f(\centerdot)$ extracts representation vectors h from augmented views. Then a projection head $g(\centerdot)$ is used to map representations vector h to z. The model is trained by minimizing the loss function for a positive pair. The loss function for the positive pair of samples (i,j) is defined as:

$$l_{i,j} = -log \frac{exp((sim((z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{k \neq i} exp((sim((z_i, z_k)/\tau)}$$

where $sim(v,u) = u^\top v/\left\|u\right\|\left\|v\right\|$ denotes a cosine similarity function and τ denotes a temperature parameter.

Approaches [26]–[28] that combine contrastive learning with FL have recently emerged as an effective approach to tackle the non-i.i.d problem where the local data cannot represent the overall distribution. Under non-i.i.d data scenarios, with the help of contrastive learning, the clients can control the drift between local and global model and fully utilize this knowledge to correct the local training. MOON [27] proposes a design of model-based comparative learning to solve non-i.i.d data problems by maximizing the consistency between the current local model learning representation and the global model learning representation. Besides the unsupervised learning, supervised learning also can be benefited from contrastive learning by incorporating the label information to compose positive and negative images.

In FedACA, instead of focusing on the semi-supervised/supervised learning setting, we take advantage of representation differences to save the communication cost by filtering the non-informative update. We set up a dynamic threshold with the help of similarity computing in contrastive learning.

III. FedACA Methodology

In this section, we present our adaptive FedACA approach with MAPE-K [29] feedback loops at two levels as shown in Figure 1. Using the asynchronous FL approach, the learning process concurrently executes on a centralized server and a set of clients. To avoid the drawbacks of synchronous FL, we run the server in asynchronous steps by randomly terminating the global iteration. To mitigate the impact of stragglers, we set up a scheme with a series of strategies on server explained in Section III-B. To minimize the communication cost overhead, we develop a thresholding algorithm to let the clients decide whether to ignore this update and not reply back to the server as described in Section III-C.

A. System Model and Problem Statement

We assume a computing system with a central server, possibly deployed in the cloud, and K clients (also called devices) at the edge, denoted as $C_1, ..., C_K$. Each client C_k has its own local dataset D_k used in model building that is subject to privacy concerns and hence the need for federated learning. p^k is the proportion of the k^{th} client in the global model aggregation process such that $p_k \geq 0$ and $\sum_{k=1}^{K} p_k = P$. Effectively, it reflects the time-based weight of the client at that instant. Our goal is to find a global model W over the combined dataset $D \triangleq \bigcup D_k$ by aggregating at the server with the aim to reduce the global training convergence overhead, maintain high enough accuracy and preserve privacy.

The objective is to minimize the optimization problem in Equation (1):

$$avg \min_{W} \left\{ \mathcal{L}(W) = \sum_{k=1}^{K} \frac{p_k}{P} \mathcal{L}_k(W) \right\}, \tag{1}$$

where $\mathcal{L}_k(W) \overset{\text{def}}{=} \sum_{i \in D_k} l_i(W; (x_i, y_i))$, is the local empirical loss of client k and $l_i(W; (x_i, y_i))$ is the corresponding loss function for data sample (x_i, y_i).

B. Procedures on Central Sever

Since we use the asynchronous approach, the server will cache all updates and aggregate the global model after a random period of time at the end of each global epoch instead of waiting for all the client updates. Local models from clients that arrive later are allocated to the subsequent global iteration. We set W^{t+1} to be the global model and w_k^t to be the local model from client k at epoch t.

We also define the structure of the message to transmit information between server and the clients. The message includes the weight (the weight in the update message can be the actual gradient or NONE if it is non-informative), local threshold, local loss value which is related to the similarity between the global representation and local representation (Loss), timestamp (TS) and local training step (Step):

$$MSG = [Weight, [Threshold], Loss, TS, Step] \tag{2}$$

Algorithm 1 illustrates all the processes at the central server. The main procedures that execute on the central server are:

1) Client Selection: At the beginning of each global iteration, the participant selection strategy in FedACA allows the server to choose clients for this training epoch. The server maintains a list v of model representation similarity measurements between each client and at the server:

$$v = [Sim(r_{w_1}, r_{W_1}), Sim(r_{w_2}, r_{W_2})..., Sim(r_{w_K}, r_{W_K})], \tag{3}$$

where $Sim(r_{w_i}, r_{W_i})$ is processed at each client and retrieved from the message via the $Threshold$ field. The purpose here is to select those clients depending on the representation similarity between the local model and the global model. For all those clients whose model has less similarity, they have a higher chance of getting involved in this epoch. The details on client selection appear in Lines 31–35 of Algorithm 1.

Algorithm 1: Process at the Server

```
Input: $K$ clients, number of local rounds $N$, number of
       global epochs $E$, pre-defined threshold $th$,
       pre-defined step $s$, similarity list $v$
Output: Updated Global Model $F(w)$
1  Initialize global model $w$ and time-related weight list
     $p = [p_1, ..p_K]$
2  for $t$ = 1: E do
3      Select a subset $C_t$ of total clients $C$:
         $C_t = ClientSelection(K, p)$;
4      Assign the time-related factor to $C_t$:
5      for $k$ in $C_t$ do
6          $p_k^t = p_k^t * \alpha$;
7          Send $w_k^t, th_k^t, s_k^t$ to Client k
8      end
9      Randomly choose a value $T$ to be the waiting time.
10     while Server waits time $> T$ do
11         Receive feedback $o_k^t$ and $w_k^t$ from client $k$:
12         if $k$ has no actual updates then
13             $w_k^t = W^t * \sigma + w_k^{t-1} * (1 - \sigma)$
14         end
15         if $k$ is a straggler then
16             $w_k^t = w_k^t * \omega + W^t * (1 - \omega)$
17             $p_k^t = p_k^t * 1/\alpha$;
18             $s_k^t = s_k^{t-1} - max\{1, log(\tau - x)\}$
19         else
20             $s_k^t = s_k^t + 1$
21         end
22     end
23     Server Updating:
24     for $k$ in $C$ do
25         $p_k^t \leftarrow \frac{p_k^t}{\sum_{k=1}^{K} p_k^t}$ ;
26         $W^{t+1} \leftarrow \sum_{k=1}^{K} p_k^t * w_k^t$ ;
27         $th^{t+1} \leftarrow \frac{\sum_{k=1}^{K} o_k^t}{K}$ ;
28     end
29     Back to the beginning of Server Process;
30 end
31 Function ClientSelection($C, v$):
32     sort($v$);
33     Select $C_{top}$: for clients in Top $n_{th}$ of $v$
34     Select $C_{re}$: fraction $c$ of remaining clients
35     return $C = C_{top} + C_{re}$
```

2) Time-related Factor: We introduce a time-related factor by using the proportion p_k from Algorithm 1. The server dynamically adjusts p_k by overwriting the corresponding values assigned to the chosen clients using a straightforward idea: for those participants which start and finish local training in this iteration, they are assigned with a higher value. This factor will be continuously adjusted in each global iteration as follows:

- After the server selects participants, we use a constant value α ($\alpha > 1$) to represent these clients shown in Equation (4):
$$p_k = p_k * \alpha, \tag{4}$$
- To maintain consistency in the model updating scheme, if the update is considered as coming from a lagger, it will be penalized which is reflected from p_k: p_k will be divided by α:
$$p_k = \begin{cases} p_k/\alpha, & \text{Stragglers} \\ p_k, & \text{Otherwise} \end{cases}, \tag{5}$$
- After the model aggregation, we normalize all time-related factors p_k with P, where $p_k = p_k/P$, and $P = \sum_{k=1}^{K} p_k$ (Line 28 in Algorithm 1).

3) Weight Modification: Whenever the server receives an update, the update will be examined along two with aspects: whether it is a straggler or not, and whether it contains actual model parameters or not. For the stragglers, since its model is trained based on a previous global model, there is a need to keep up with the current state. For the update which does not contain an actual weight, we need to estimate how the updates will appear in the current epoch. To that end, the server will perform weight modification in two steps:

1. Straggler Control: The server will retrieve the $timestamp$ to determine whether the update is a straggler. To address the staleness, we use τ to denote the latency. Besides the time-related factor reduction, for the straggler, it follows up the most up-to-date global model W^t. We merge the updated weights w_k^t with W^t corresponding to their staleness (arrival delay) factor ω. The details of the first step are per Equation (6):

$$w_k^t = \begin{cases} w_k^t, & \text{From current epoch} \\ w_k^t * \omega + W^t * (1-\omega), & \text{Otherwise} \end{cases}, \tag{6}$$

where $\omega = \frac{1}{a-\tau} * \omega$, a is a constant value.

2. Model Estimation: The server will retrieve the $Updated\_weight$ to determine whether it contains the real model parameters. If the server only receives a negative-acknowledgment message – the NONE message – since the clients do not transmit the actual update, the server will estimate their parameters w_k^t in collaboration with the global model W^t and previous local model w_k^{t-1}. The details of the second step are per Equation (7):

$$w_k^t = \begin{cases} w_k^t, & \text{Update = actual model} \\ W^t * \sigma + w_k^{t-1} * (1-\sigma), & \text{Update = "NONE"} \end{cases}, \tag{7}$$

where $w_k^{t-1} = W^t$ if $t = 1$.

4) Dynamic Local Training Epoch: In traditional FL, e.g., FedAvg [5], a local epoch is a fixed number which illustrates the number of iterations through client's local training. Due to the data and local computation heterogeneity or network delay, choosing a perfect local epoch number for each client is challenging.

In FedACA, the server assigns local training step size s^t at global epoch t for each client dynamically to balance their local training time. For the "punctual" clients, which are those that reply in the same epoch, their updates lead to synchronous updates. We increment s^t by 1 to increase their training time (Line 18 in Algorithm 1). When the client k is marked as "straggler", the server will subtract x, which is given by $x = max\,\{1, log(\tau - x)\}$ depending on the latency τ from its local training step size s_k^t (Line 16 in Algorithm 1) and predefined value x. The newest local training step size will be transmitted along with the global model and other information once client k is selected.

5) Cross-Epoch Adaptive Aggregation: FedACA uses a new cross-epoch, weighted aggregation heuristic: the received models are trained which start from different epochs. After adjusting the weight w and time-related factor p (Line 13-21 in Algorithm 1), we define the function of our aggregation at the t_{th} global epoch as shown in Equation (8):

$$f^t(w) = \sum_{k=1}^{K} \frac{p_k^t}{P^t} * w_k^t, \tag{8}$$

Unlike FedAvg [5] and other approaches [12], [30], which aggregate the global model corresponding to the size of the local dataset or just average the local model, we let the server dynamically adjust the proportion of each local update which illustrates its significance in global model.

C. Training on Local Clients

For the local model training, our scheme aims to improve the communication efficiency of asynchronous FL by letting the most informative updates make contributions. In the following, we present the local training objective and the training procedure. Then, we discuss the relation to contrastive learning. Finally, we discuss our informative update scheme.

As shown in Algorithm 2, the processes for each client consist of two parts: local learning and informative update determination.

1) Local Training Objectives: The global model is sent to selected clients at start of each global epoch training process. Each client needs to perform its local computing depending on this global model. For this, after receiving the latest model, each client updates the model with its own local dataset. Our local training framework is designed as a simple and effective approach based on FedAvg [5], with slight modifications in the local training phase.

Our local training (Line 1-10 in Algorithm 2) is based on an intuitive idea: the global model coming from the server is an aggregation of all local updates and hence has a better and fairer view in training than every single local model especially under the scenarios where the client data typically follows a non-i.i.d. distribution. One of our local objectives is to keep the local model close to the received global model during the training to limit the impact of local variable updates in further model aggregation. To illustrate the distance reduction, we minimize the model similarity between the global model and local model.

Algorithm 2: Process at the Edge Devices

```
Input: Local rounds N, global model $W^t$, temperature
       $m$, learning rate $\eta$, weight variance $threshold_1$
       $O_{rep}$, representation variance $threshold_2$ $H_{rep}$,
       hyper-parameter $\beta$, local dataset $D$
Output: $w^t$, $o^t$, $H$
1  Receive local epoch $N$ and global weight $W_t$, start
     time from server.
2  for $i$ = 1: N do
3      for each batch $b = \{x, y\}$ in $D$ do
4          $l_{base} = CrossEntropyLoss(F_{w^t}(x), y)$
5          $neg = r_{w^t}(x), r_{w^{t-1}}(x)$
6          $pos = r_{w^t}(x), r_{W^t}(x)$
7          $l_{con} = -log\frac{exp(sim(pos)/m)}{exp(sim(pos)/m)+exp(sim(neg)/m)}$
8          $loss = \beta * l_{con} + (1-\beta) * l_{base}$
9          $w^t = w^t - \eta \bigtriangledown loss$
10         $H_{x,y,i} = H_{x,y,i} + sim(neg)/m$
11     end
12 end
13 $o^t = \left\|w^{t+1} - w^t\right\|_2$;
14 $H = avg(H_{x,y,i})$
15 if ($o^t < O_{rep}$ & $H < H_{rep}$) then
16     Return ( NONE, $[o^t, H]$, loss, start time, N) to
         server;
17     Update $H_{rep} = H$, $O_{rep} = O$
18 else
19     Return ($w^t$, $[o^t, H]$, loss, start time, N) to server;
20     Update $H_{rep} = H * 0.9$, $O_{rep} = O$
21 end
22 Open for the next local training.
```

Another local objective is to maximize the local model difference between current epoch and previous epoch since we want to utilize each chance for uploading when the main communication cost is to transmit the model parameters. In FedACA, we increase the distance between the local model from current epoch and previous epoch to enforce the newest model is far from the previous.

Particularly, we can achieve these two objectives using contrastive learning. As we mentioned in Section II-C, the learning purpose is to train models by minimizing the distance between representations of positive pairs and maximizing the distance between negative pairs [25].

The local objective is realized by minimizing the loss function. Suppose client k receives model w^t from the server. During the training, we define the total loss in Equation (10):

$$loss = \beta * l_{con} + (1-\beta) * l_{base}, \tag{10}$$

where β is a hyper-parameter to control the weight of contrastive loss and base loss.

For every input x, the base function $loss_{base}$ is given as:

$$l_{base} = CrossEntropyLoss(F_{w_i^t}(x), y), \tag{11}$$

The contrastive loss function l_{con} is given as:

$$l_{con} = -log\frac{exp((sim(pos)/m)}{exp(sim(pos)/m) + exp(sim(neg)/m)}, \tag{12}$$

where the neg is equivalent to pair of representation of x from the current local model $r_{w_k^t}(x)$ and the representation of x from the last local model $r_{w_k^{t-1}}(x)$, pos is equivalent to the pair of representation of x from the current local model $r_{w_k^t}(x)$ and the representation of x from the global model $r_{W_t}(x)$.

The local objective then is to minimize:

$$E_{\{x,y\}\sim D_k}\left[\beta l_{con}(w; W; (x)) + (1-\beta) l_{base}(w; (x, y))\right], \tag{13}$$

2) Informative updates: After local training, an informative update strategy is implemented. Our purpose is to detect non-informative models and avoid uploading them, thereby reducing unnecessary communication costs. Each client measures its update depending on the informative level, which is the dissimilarity between local model in current and past epochs and decides whether to communicate with its updates or not.

The policy of informative determination is based on two schemes (Line 13-14 in Algorithm 2):

- comparison of the cosine similarity between current local model representation R^t and previous local model representation R^{t-1} to an adaptive threshold H.
- comparison of the l_2 norm of the difference between the local model w^t and previous local model w^{t-1} to another adaptive threshold o^t.

For the first scheme in informative determination, we take advantage of the contrastive loss function on negative pairs, which in our scenarios is representation of current local model and representation of previous local model. The first value H is computed by averaging the cosine similarity of representation for all inputs.

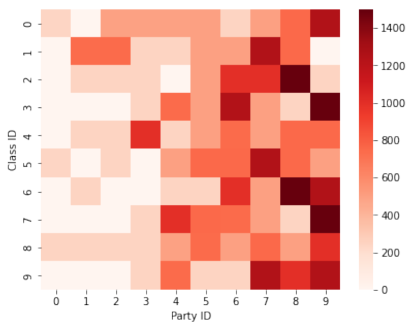
For the second scheme, we compute the norm of the difference between the current and previous local model:

$$o^t = \left\|w^t - w^{t-1}\right\|_2, \tag{14}$$

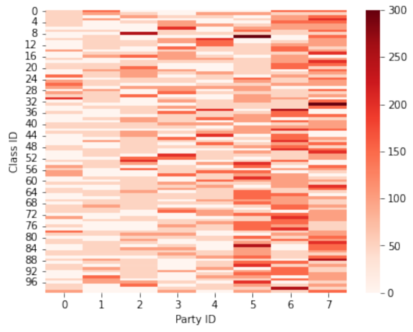
Since the norm of the gradients and the model similarity are expected to decrease along with the training progress especially near the optimal point, a dynamic threshold is more appropriate to filter the non-informative updates by limiting the number of dropping updates. So we keep track of H^{rep} and o. At the end of the local training, the threshold value will be updated compared to the trained result (Line 14-19 in Algorithm 2). The client will not transmit its actual update back to the server only if it satisfies the two schemes above. After these processes mentioned above, the client will finally make its own decision and reply back to the server with its information.

IV. Experimental Setup

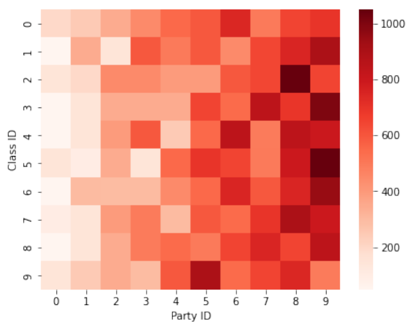
To demonstrate the effectiveness and robustness of FedACA, we present the details of our experimental setup.



(a) CIFAR-10

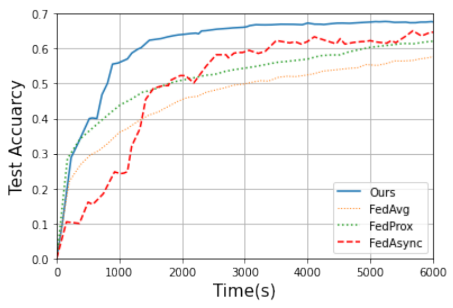


(b) CIFAR-100

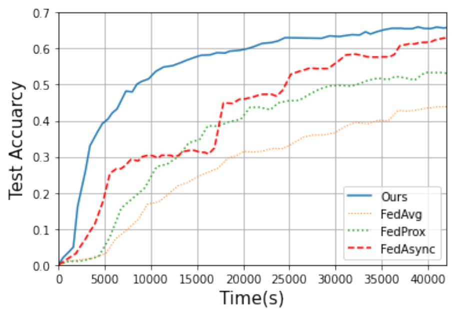


(c) Fashion-MNIST

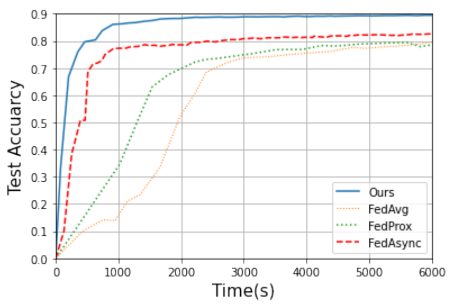
Fig. 2: The data distribution of each party using non-i.i.d data partition. The color bar denotes the degree of the data samples. Each rectangle represents the number of data samples of a specific class in a certain party.



(a) CIFAR-10



(b) CIFAR-100



(c) Fashion-MNIST

Fig. 3: Test accuracy comparison of different FL methods on non-i.i.d. CIFAR-10, CIFAR-100 and Fashion-MNIST datasets. The accuracy results for certain running time (in seconds) as specified in the X-axis' labels.)

A. Datasets

We evaluated FedACA using three different datasets suitable for federated learning.

- **CIFAR-10**: The CIFAR-10 [31] dataset consists of 60,000 32 × 32 color images in 10 classes, with 6000 images per class. Of these, 50,000 are training images and 10,000 test images.
- **CIFAR-100**: The CIFAR-100 [31] dataset has 60,000 32 × 32 colour images in 100 classes with each class having 600 images.
- **Fashion-MNIST**: Fashion-MNIST [32] is a dataset that contains a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes.

We use Dirichlet distribution to generate the non-i.i.d training data partition among devices as shown in prior work [33]. For each class *j*, we sample $p_j \sim Dir_K(\beta)$ where $Dir_K(\beta)$ is the Dirichlet distribution and β is the concentration parameter (0.5 by default). We allocate $p_{j,k}$ proportion of the instances of class *j* of the complete dataset to batch *j* and due to the small value of β, some batches may lack of an entire subset of classes. In our experiment, CIFAR-10 and Fashion-MNIST will be divided into 10 clients and CIFAR-100 will be divided into 8 clients.

After such a non-i.i.d partition strategy, the number of classes and samples at each client vary from each other. The data size in each edge device ranges from 1000 to 8400 with a standard deviation of 2242 on CIFAR-10 and CIFAR-100, and 1100 to 9000 with a standard deviation of 2489 on Fashion-MNIST. The data distributions among clients in default settings are shown in Figure 2.

B. Comparative Methods

To demonstrate the effectiveness of FedACA, we compare it with state-of-the-art federated learning algorithms:

- **FedAvg [5]** is the most commonly used synchronous FL approach.
- **FedProx [30]** is the synchronous federated learning framework with a proximal term on the local objective function to mitigate the data heterogeneity problem. It can handle stragglers due to system heterogeneity by applying different local epochs for clients.
- **FedAsync [15]** is used as a baseline asynchronous FL approach which updates the global model by weighted averaging. To present the best result, we follow FedAsync's [15] strategy by adding a polynomial term to the weighting function as an adaptive mixing hyper-parameter: $S_a(t-\tau) = (t-\tau+1)^{-a}$.

C. Implementation Details

The proposed FedACA and baselines are all implemented in PyTorch [34] and evaluated on three instances on Google cloud. We deploy the FL server exclusively on a server instance without GPU. The clients are divided into two sets and each of the subsets is implemented on the client instance with two 16-core CPUs, 60GB of main memory and one NVIDIA K80 GPU. Each client gets assigned one CPU core.

D. Training Details

Hyperparameters. For FedACA and all state-of-the art approaches, the batch size is set to 16 and the number of local epochs is set to 10 for every dataset. We tuned the value of the Stochastic Gradient Descent (SGD) optimizer and got the best learning rate λ as 0.001. The SGD weight decay is set to 0.01 and the SGD momentum is set to 0.9. We empirically select a fraction C of clients for FedAvg and FedProx as 0.5 and 0.6 after making the trade-off between number of participants and accuracy performance. For FedAsync model, we set ρ = 0.01 and a = 0.5. For FedACA, we tune β from $0.001, 0.01, 0.1 and 0.9$ and select 0.1 for the best result.

Straggler Simulation. To simulate the stragglers and unstable network situations, we add a random offset value to be the latency at each client's side. Since the processing time at local is varied corresponding to its model parameter size and dataset size, to illustrate that the latency is non-negligible, the network delay accounts for at least 10% of total processing time. We set the latency between 0 to 50 seconds for CIFAR-10 and Fashion-MNIST and 0 to 200 for CIFAR-100.

Models. For CIFAR-10 and Fashion-MNIST, we use a convolutional neural network (CNN) network. The network architecture includes two 5x5 convolution layers, each with 32 and 64 filters, followed by 2x2 max pooling and two fully connected layers with ReLU activation (the first with 120 units and the second with 84 units). For CIFAR-100, we use ResNet-18 [35] as the base encoder. For all datasets, we add 2-layer MLP with an output size of number of the classes to be the project header.

V. Experimental Results

A. Accuracy Comparison

Table I presents the results of the prediction performance and the variance of the test accuracy on all the datasets comparing FedACA to the baseline approaches. We report the best test accuracy after each training process converges within a global running time budget: 6000s, 42000s and 8000s.

Across all datasets, FedACA outperforms the best baseline FL method, FedAsync, by 4.20% to 8.04%. FedAvg and FedProx do not perform well within the specified time budgets on the highly unbalanced and non-i.i.d datasets and have no effective way to deal with stragglers. In particular, FedACA outperforms the worst baseline method, FedAvg, by 11.58% to 15.32%.

Using the same new epoch starting scheme as FedAsync, our method achieves higher accuracy than FedAsync for the same running time for all the experiments. Also, FedACA exhibits significantly lower accuracy variance with the non-i.i.d of data distribution. The reasons that we achieve better performance are:

- during the local model training, our local loss function which involves constrastive loss focuses on keeping smaller distances between the local model and the global model and larger distances between previous local model and current local model, and
- our method's time-related weighted aggregation heuristic at the server can more effectively mitigate the influence from straggler clients from past epochs.

TABLE I: The best prediction accuracy and average variance of test accuracy among all devices of FedACA and the other baselines on all datasets with non-i.i.d. case in the specified time budget. The best results are highlighted in bold font.

Method	**CIFAR-10 (Time=6000s)**	**CIFAR-100 (Time=42000s)**	**Fashion-MNIST (Time=8000s)**
FedAvg	58.42%±0.21%	46.62%±0.51%	79.35%±0.15%
FedProx	62.18%±0.23%	53.25%±0.43%	79.15%±0.24%
FedAsync	63.28%±1.91%	58.47%±1.33%	81.75%±0.33%
Our method	**67.48%±0.26%**	**66.51%±0.32%**	**89.51%±0.13%**

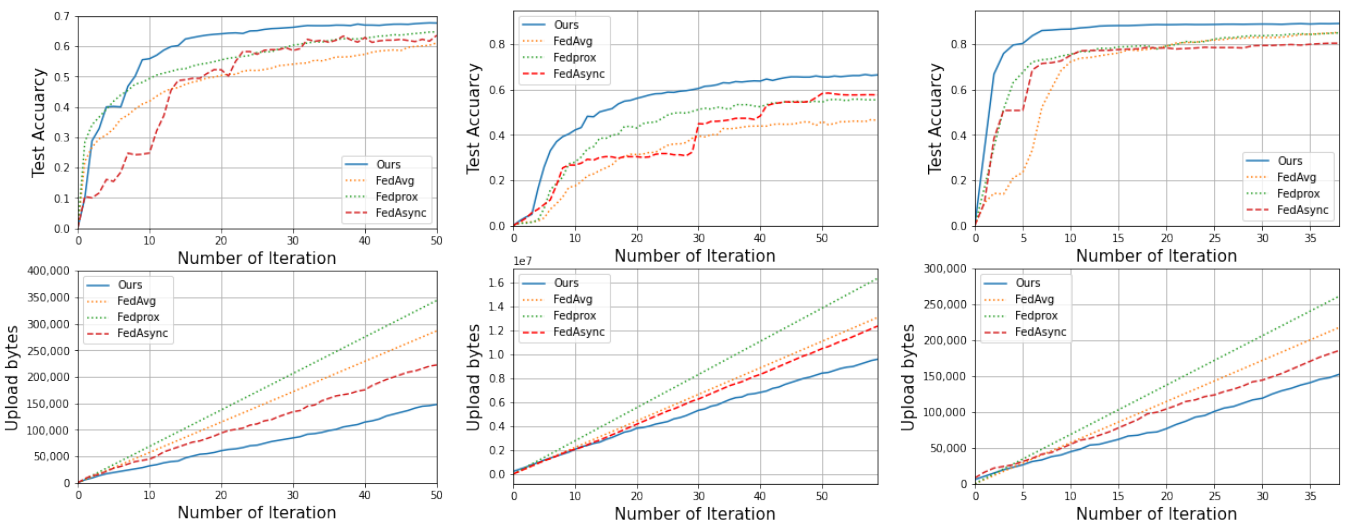
B. Robustness to Stragglers

To illustrate the robustness of our approach to stragglers, we compare the convergence speed and prediction accuracy over time. The performance difference as a function of running time can be noticed from the convergence timeline figure shown in Figure 3. Figure 3 shows that FedACA has the fastest convergence speed in the presence of network latency fluctuations and laggers. The dataset size among all clients varies significantly: the largest one is 10x compared to the smallest one. Since every client has the same computation capacity, the computation time for the largest dataset was multiple times larger than the smallest.

In Table III we report the entire training time of synchronous approaches: FedAvg and FedProx, and asynchronous FL approaches: FedACA and FedAsync, to achieve the target test performance. As discussed previously, to simulate the unstable connection in the network, we add network delays to each edge device for a random value.

As seen in Table III, for synchronous FL approaches, they have the highest computation time cost across all four benchmarks. For example, to achieve an accuracy of 60% for the Cifar-10 using CNN model, FedAvg and FedProx spend 5.05x and 3.34x longer time than FedACA, respectively. The presence of stragglers inevitably leads to longer training times for every global epoch since the aggregation has to wait to receive from the slowest node to finish its local training. This also results in slowness of the model convergence to the optimal solution.

In asynchronous FL, we make sure that over 20% of the total edge devices to have staleness. For asynchronous FL approaches, compared to FedAsync, which spends at least 2.34x longer time to reach the goal, FedACA shows impressive time cost reduction due to our dynamic local training step size



(a) CIFAR-10 (b) CIFAR-100 (c) Fashion-MNIST

Fig. 4: Impact of FedACA's compression precision on the prediction performance and the communication cost across all dataset. 2-class dataset. All results are plotted with the average of every 40 global rounds.

TABLE II: Amounts of data (Bytes) transferred between edge devices and server to reach the target accuracy on all non-i.i.d. datasets. The best results are highlighted in bold font.

Method	**CIFAR-10 (acc. = 0.6)**	**CIFAR-100 (acc. = 0.5)**	**Fashion-MNIST (acc.= 0.8)**
FedAvg	275040.93	–	128845.39
FedProx	213156.23	8.3*1e6	140234.48
FedAsync	111436.76	8.5*1e6	143452.98
Our method	**50193.25**	**2.8*1e6**	**24345.14**

and edge device selection strategy. The client selection method gives the edge devices which model is less similar to the global model a higher chance of getting selected in the next epoch. The adaptive local training epoch modification decreases the training time variance among all devices.

C. Communication Efficiency

We next further compare the communication cost in terms of the amount of data uploaded and downloaded via network. Table II shows the amount of data transferred between the edge clients and the server to reach the target accuracy. To better illustrate how our approach outperforms in communication efficiency, Figure 4 shows the prediction performance and communication cost for the different approaches as a function of the number of global iterations.

As seen in Figure 4, FedAvg and FedProx incur the higher overall communication cost – at least about 1.5x FedACA, since they both use the same synchronous updating mechanism by communicating with a certain fraction of clients. Specifically, to obtain a certain target accuracy as illustrated in Table II the amount of data transferred between edge devices and main server under FedAvg and FedProx is at least 3.1x of FedACA.

Compared to FedAsync, FedACA performs better since it has a lower communication rate by dropping the less informative updates at the edge devices. By adapting the threshold in each round based on the edge devices' norm update, the dropping rate will not increase obviously near the optimal as we discuss in Section III-C2 which impacts the test accuracy.

VI. Conclusions

This paper describes FedACA, which is an adaptive and asynchronous FL technique comprising feedback loops at two levels that overcome communication bottlenecks and straggler problems in prior efforts. To alleviate the communication bottleneck, FedACA adapts the communication rates between the edge devices and the centralized server by calculating the cosine similarity and L-2 norm between the local model parameters and the global model parameters, and allowing only those local updates to propagate from edge to server whose values of the L2 norm and similarity are greater than the adaptive threshold. To mitigate the influence of stragglers, FedACA supports an adaptive participant selection and local training epoch variation. Empirical evaluations comparing FedACA with three contemporary FL solutions on three non-i.i.d datasets show that FedACA significantly reduces the communication cost, handles stragglers and speeds up model convergence while maintaining acceptable accuracy.

Our work only considers the situation that the number of edge devices remains unchanged and neglects the probability that edge devices could be frequently offline during the training process in the real network. Also, this is still an open issue to build efficient communication methods with the changing datasets at the edge devices which lead to the unstable model domain. For future work, a promising direction is to further implement the asynchronous FL procedure on online learning conditions with changing devices and datasets.

TABLE III: The time(s) takes for each FL method to reach a target accuracy of N. (Note that FedAvg will reach to the 60% accuracy later in CIFAR-10 which are not shown in the Fig.2, the test accuracy timeline curves. FedAsync is not able to reach the target accuracy for CIFAR-100, thus is omitted.) The best results are highlighted in bold font.

	CIFAR-10			CIFAR-100			Fashion-MNIST		
	N = 0.50	N = 0.55	N = 0.60	N = 0.40	N = 0.45	N = 0.50	N = 0.60	N = 0.70	N = 0.80
FedAvg	3029.09	5014.36	7483.23	33289.57	47823.69	—	2314.39	2798.01	6658.25
FedProx	1821.76	3169.45	4933.05	18732.95	24509.20	30333.23	1506.45	2004.11	6658.74
FedAsync	1801.45	2367.45	3501.01	17053.92	17502.35	26062.92	393.56	537.09	2422.65
FedACA(ours)	**762.90**	**879.49**	**1478.26**	**5002.14**	**6724.47**	**9024.41**	**187.34**	**243.09**	**743.08**

Acknowledgments

This work was supported in part by NSF ITE #2040462. Any opinions, findings, and conclusions or recommendations in this paper are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] G. Marcus, "Deep learning: A critical appraisal," *arXiv preprint arXiv:1801.00631*, 2018.

[2] Y. Qu, S. Yu, W. Zhou, S. Peng, G. Wang, and K. Xiao, "Privacy of things: Emerging challenges and opportunities in wireless internet of things," *IEEE Wireless Communications*, vol. 25, no. 6, pp. 91–97, 2018.

[3] H. Xiong, H. Zhang, and J. Sun, "Attribute-based privacy-preserving data sharing for dynamic groups in cloud computing," *IEEE Systems Journal*, vol. 13, no. 3, pp. 2739–2750, 2018.

[4] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[5] 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*. PMLR, 2017, pp. 1273–1282.

[6] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[7] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[8] T. Chen, G. Giannakis, T. Sun, and W. Yin, "Lag: Lazily aggregated gradient for communication-efficient distributed learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

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

[10] Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala, "Fedat: a high-performance and communication-efficient federated learning system with asynchronous tiers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–16.

[11] "Speedtest market report," http://www.speedtest.net/reports/united-states/, accessed: 2022-04.

[12] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for on-device federated learning." 2019.

[13] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection," in *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021, pp. 19–35.

[14] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *International Conference on Machine Learning*. PMLR, 2017, pp. 4120–4129.

[15] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[17] G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, and F. Taiani, "Fleet: Online federated learning via staleness awareness and performance prediction," in *Proceedings of the 21st International Middleware Conference*, 2020, pp. 163–177.

[18] M. Ribero and H. Vikalo, "Communication-efficient federated learning via optimal client sampling," *arXiv preprint arXiv:2007.15197*, 2020.

[19] N. Singh, D. Data, J. George, and S. Diggavi, "Sparq-sgd: Event-triggered and compressed communication in decentralized stochastic optimization," *arXiv preprint arXiv:1910.14280*, 2019.

[20] J. Wang, M. Kolar, N. Srebro, and T. Zhang, "Efficient distributed learning with sparsity," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3636–3645.

[21] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[22] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2021–2031.

[23] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 10, pp. 4229–4238, 2019.

[24] C. Chen, H. Xu, W. Wang, B. Li, B. Li, L. Chen, and G. Zhang, "Communication-efficient federated learning with adaptive parameter freezing," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 1–11.

[25] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[26] F. Zhang, K. Kuang, Z. You, T. Shen, J. Xiao, Y. Zhang, C. Wu, Y. Zhuang, and X. Li, "Federated unsupervised representation learning," *arXiv preprint arXiv:2010.08982*, 2020.

[27] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 713–10 722.

[28] X. Mu, Y. Shen, K. Cheng, X. Geng, J. Fu, T. Zhang, and Z. Zhang, "Fedproc: Prototypical contrastive federated learning on non-iid data," *arXiv preprint arXiv:2109.12273*, 2021.

[29] A. Computing *et al.*, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.

[30] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[31] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

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

[33] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440*, 2020.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.