# Communication-Efficient Federated Learning for Heterogeneous Edge Devices Based on Adaptive Gradient Quantization

Heting Liu, Fang He and Guohong Cao Department of Computer Science and Engineering The Pennsylvania State University Email: {hxl476, fxh35, gxc27}@psu.edu

Abstract-Federated learning (FL) enables geographically dispersed edge devices (i.e., clients) to learn a global model without sharing the local datasets, where each client performs gradient descent with its local data and uploads the gradients to a central server to update the global model. However, FL faces massive communication overhead resulted from uploading the gradients in each training round. To address this problem, most existing research compresses the gradients with fixed and unified quantization for all the clients, which neither seeks adaptive quantization due to the varying gradient norms at different rounds, nor exploits the heterogeneity of the clients to accelerate FL. In this paper, we propose a novel adaptive and heterogeneous gradient quantization algorithm (AdaGQ) for FL to minimize the wall-clock training time from two aspects: i) adaptive quantization which exploits the change of gradient norm to adjust the quantization resolution in each training round; and ii) heterogeneous quantization which assigns lower quantization resolution to slow clients to align their training time with other clients to mitigate the communication bottleneck, and higher quantization resolution to fast clients to achieve a better communication efficiency and accuracy tradeoff. Evaluations based on various models and datasets validate the benefits of AdaGQ, reducing the total training time by up to 52.1% compared to baseline algorithms (e.g., FedAvg, QSGD).

### I. INTRODUCTION

Intelligent applications based on deep neural networks (DNN) have been developed for edge devices such as Internet of things (IoTs) and smart mobile devices over the past years [1], [2]. These applications rely heavily on the knowledge obtained from the big data, and they generate massive amounts of data in return. The most straightforward way to utilize these locally generated data is to upload the data to the cloud and train the DNN models in the cloud [3], [4]. However, sharing data is challenging due to the increasing privacy concerns.

Federated learning (FL) [5], [6] emerges as a solution to privacy-preserving machine learning. In FL, multiple devices train a shared global model without uploading their data to the central server. Specifically, in each round every participating device (i.e., client) does the following. First, it downloads the latest model from the central server. Next, it updates the downloaded model based on its local data using stochastic gradient descent (SGD). Finally, all clients upload their model updates to the central server, where the model updates are aggregated to form a new global model. These steps are repeated until a certain convergence criterion is satisfied.

One important research problem in FL is to address the massive communication overhead resulted from uploading the gradients and downloading in each training round. The model updates (i.e., gradients) can be in the range from megabytes to gigabytes for modern DNN architectures with millions of parameters [7]. Thus, communication will become a bottleneck when applying FL to edge devices where the wireless bandwidth is limited. Existing approaches addressing the communication overhead in FL fall into two folds: i) to reduce the amount of communication by allowing each client to perform multiple local updates between two communication (aggregation) rounds [8], [9]. While the number of communication rounds is reduced, the data size in each communication round is still very high; and ii) to mitigate the communication overhead by using gradient compression [10]-[12], reducing the amount of data transmitted in each training round.

One widely used gradient compression method is gradient quantization, where the gradient is represented by a number of bits which can determine the number of quantization levels and affect the performance of the gradient quantization algorithms. With fewer number of quantization levels (i.e., low quantization resolution) used, the quantization algorithm uses fewer number of bits and then reduces the communication overhead more aggressively. However, it also introduces quantization error in the uploaded gradients, and thus may require more training rounds to converge. With higher quantization resolution, there will be less quantization error, but more data has to be transmitted in each round, increasing the accumulated training time. Thus, the quantization resolution should be carefully determined to minimize the wall-clock training time.

Existing gradient quantization algorithms [11]–[13] rely on fixed and pre-determined quantization throughout the training process. However, different FL task has different characteristics in terms of convergence time, communication cost and network condition, etc., and then it makes pre-determined quantization less effective because the optimal quantization resolution at different time may be different. For example, based on our measurements, the gradient value has large variations during the training process, and thus we should adaptively adjust the quantization based on the training rounds.

Moreover, in mobile edge computing, different edge devices have different communication resources and some of them only have limited wireless bandwidth. Such heterogeneity makes fixed quantization less effective because the training speed is bounded by the slowest client, thus leading to long waiting time for other clients (i.e., straggler effects). Recent work [14]-[18] studied FL under heterogeneous clients. For example, [17], [18] addressed the straggler problem by designing an asynchronous aggregation strategy where clients do not wait for each other every round and simply run independently so that the waiting time of faster clients is reduced. Although asynchronous aggregation can reduce the communication time, delaying gradients of the stragglers may introduce errors, increase the number of training rounds or even cause divergence in model training. We take a different approach to address this problem by assigning fewer number of quantization levels to the slow clients. In this way, the slowest client will transmit less amount of data in each round, and its transmission time can be reduced and aligned with other clients, thus reducing the overall per-round training time.

To realize our ideas, in this paper, we propose an Adaptive and Heterogeneous Gradient Quantization algorithm, namely AdaGQ, which dynamically assigns different number of quantization levels to different clients based on online learning to minimize the wall-clock training time of FL. Specifically, the proposed strategy includes two aspects. (i) The number of quantization levels should be adaptive to the training process in accordance with the gradient norm. Since the gradient norm (indicating the upper bound of the gradient magnitude) has large variations as the training proceeds, different numbers of quantization levels are chosen based on the training round to achieve better tradeoff between communication efficiency and accuracy. (ii) The number of quantization levels should adapt to the clients' communication capability. Specifically, slow clients (i.e., the clients with longer local training and communication time) are assigned fewer number of quantization levels to mitigate the delay of gradient aggregation at the server; while fast clients are assigned more quantization levels to maintain the accuracy achieved by the global model.

This paper has the following main contributions.

- Through extensive experiments, we identify that gradient quantization should be adaptive to the training process and the clients' communication capability to reduce the training time for heterogeneous clients.
- We design AdaGQ, an online learning based adaptive and heterogeneous gradient quantization, to minimize the wall-clock training time.
- We evaluate the proposed scheme through extensive experiments with various datasets and deep learning models. Evaluation results show that AdaGQ reduces the total training time by 34.8%-52.1% compared to baselines.

# II. BACKGROUND AND MOTIVATION

In FL, the goal of the training process is to find the model parameters (weights) w that can minimize a loss function  $L(\mathbf{w}) := \frac{1}{D} \sum_{h=1}^{D} l_h(\mathbf{w})$ , where  $l_h(\mathbf{w})$  is the loss of data sample h and D is the number of data samples. In particular, we minimize  $L(\mathbf{w})$  using SGD algorithm, i.e.,

 $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k g(\mathbf{w}_k)$  for  $k \in \{0, 1, \dots\}$ , where  $g(\mathbf{w}_k)$  denotes the stochastic gradients at iteration k, and  $\eta_k$  is the step size at iteration k. In FL, the training data is spread across a number of edge devices, and FL enables distributed training without sharing data across these clients. Assume there are n clients and a central aggregating server. Each client  $i \in \{1, \dots, n\}$  has a dataset  $D_i$  of size  $m_i$ . In a typical FL algorithm [9], [13], [19], the goal is to train a global model, represented by the parameter vector  $\mathbf{w}$ , which minimizes

$$\min_{\mathbf{w}\in\mathbb{R}^d} L(\mathbf{w}) = \sum_{i=1}^n p_i L_i(\mathbf{w}),\tag{1}$$

where  $L_i(\mathbf{w})$  is the loss function at client *i*, and  $p_i = \frac{m_i}{\sum_{i=1}^n m_i}$  represents the fraction of data stored at client *i*.

## A. Federated Learning with Gradient Quantization

In the conventional FedAvg [8], every client performs a certain number of gradient descent steps locally at each round, and then uploads the updated weights to the central server followed by a global aggregation. This procedure is repeated until the training converges. In quantized SDG [12], the clients upload quantized gradients instead of model weights. Formally, let  $g(\mathbf{w}_k^{(i)})$  denote the stochastic gradient on the *i*<sup>th</sup> client's dataset  $D_i$  at round k. To reduce the communication cost at each round, every client sends quantized weight updates (gradients)  $Q(g(\mathbf{w}_k^{(i)}))$  to the server, where  $Q(\cdot)$  represents a stochastic quantization function. Once the server receives the quantized gradients from all clients, the aggregation is performed to update the global model by Eq. (2):

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \sum_{i=1}^n p_i Q(g(\mathbf{w}_k^{(i)})).$$
(2)

We adopt the commonly used stochastic uniform quantization function (QSGD)  $Q_s(\cdot)$  [11], [12], where  $s \in \mathbb{N} = \{1, 2, \cdots\}$  is the parameter that determines the compression resolution, i.e., the number of quantization levels. Let **v** denote the aligned gradients:  $\mathbf{v} = [v_1, \cdots, v_d] \in \mathbb{R}^d$  with  $\mathbf{v} \neq \mathbf{0}$ . The  $j^{th}$  dimension of  $\mathbf{v}, v_j$ , is quantized to be  $Q_s(v_j)$  as follows,

$$Q_s(v_j) = ||\mathbf{v}||_2 \cdot sign(v_j) \cdot \zeta_j(\mathbf{v}, s), \tag{3}$$

where  $\zeta_j(\mathbf{v}, s)$  is a random variable defined as

$$\zeta_{j}(\mathbf{v},s) = \begin{cases} l/s, & \text{with probability } \left(1 - \frac{|v_{j}|}{||\mathbf{v}||_{2}}s + l\right)\\ (l+1)/s, & \text{otherwise.} \end{cases}$$
Here,  $0 \leq l \leq c$  is an integer such that  $|v_{j}| \leq \left[l/c, (l+1)/s\right]$ 

Here,  $0 \le l < s$  is an integer such that  $\frac{|v_j|}{||\mathbf{v}||_2} \in [l/s, (l+1)/s]$ .  $Q_s(\mathbf{v})$  is defined to be 0 if  $\mathbf{v} = \mathbf{0}$ .

The idea of QSGD can be explained as follows. A gradient  $v_j$  consists of a sign bit and the absolute value  $|v_j|$ . To quantize  $|v_j| \in [0, ||\mathbf{v}||_2]$ , we divide the interval into s-1 bins of equal length, with end points  $0 = \tau_1 < \tau_2 < \cdots < \tau_s = ||\mathbf{v}||_2$ . Given  $v_j$  that belongs to a bin  $[\tau_i, \tau_{i+1})$ , the probability is assigned to represent  $v_j$  to be  $\tau_i$  or  $\tau_{i+1}$  based on its relative location inside the bin. That is,  $\tau_i$  is chosen to represent  $v_j$  with probability  $p = 1 - (v_j - \tau_i)/(\tau_{i+1} - \tau_i)$ , and  $\tau_{i+1}$  is chosen with probability 1 - p (so that we have  $\mathbb{E}[Q_s(v_j)] = v_j$ ).



Fig. 1. Training process of ResNet-18 on Cifar-10.

Then,  $v_j$  is represented by an end point which only needs  $\log_2(s) + 1$  bits (with the sign bit). Different from weight quantization which quantizes the model weights, gradient quantization compresses the gradients without changing the number of bits to represent the model weights.

In QSGD based FL, the number of quantization levels (i.e., quantization resolution) is manually pre-defined, and shared by all clients throughout the model training process, which faces two issues: i) the gradient norm  $||\mathbf{v}||_2$  may change in different rounds of the training process, leading to a varying interval  $[0, ||\mathbf{v}||_2]$ . However, the pre-defined quantization fails to automatically adapt to different intervals; ii) clients have heterogeneous communication resources, which creates opportunities to minimize the total training time by assigning different clients with different quantization resolutions.

Next, we will detail our motivation of adaptive gradient quantization by inspecting the gradient norm during training. We also explain how to assign different quantization resolutions to heterogeneous clients to minimize the training time.

## B. Motivations

In this section, we investigate the idea of *adaptive gradient quantization* and *heterogeneous gradient quantization*. We start with observing the gradient norm during the training of ResNet-18 and GoogLeNet on the Cifar-10 dataset. As shown in Fig. 1(a), the gradient norm has large variations in training ResNet-18, i.e., with a rapid decrease in the early rounds and mild decrease later on (similar observations for GoogLeNet). Based on the aforementioned analysis of QSGD, a larger gradient norm, e.g., the  $||\mathbf{v}||_2$  in the early rounds, results in a wide value range of gradients. Thus, to reduce the quantization error, more quantization levels (i.e., higher quantization resolution) should be used to represent the gradient in the early training rounds.

On the other hand, a small gradient norm in later training rounds suggests that the gradient has a small value range. Then, fewer number of quantization levels will be able to sufficiently represent the gradient with good precision. This idea is supported by some other research [20], [21] that highlights the importance of early training phases. More importantly, low quantization resolution means less amount of data to be uploaded to the server and thus reducing the communication time, potentially reducing the total training time. The above analysis motivates us to adaptively adjust



Fig. 2. Different quantization strategies for heterogeneous edge devices.

the quantization resolution based on the gradient norm to minimize the training time without compromising accuracy. To validate this idea, we adjust the quantization resolution based on the change of gradient norm, i.e.,  $s_k = s_{k-1} + \log \frac{||\mathbf{g}_{k}||}{||\mathbf{g}_{k-1}||}$ , where  $s_k$  denotes the quantization resolution of round k and  $||\mathbf{g}_k||$  denotes the gradient norm of round k, respectively. Fig. 1(b) shows the accuracy in each communication round when training ResNet-18 on Cifar-10 dataset with the above *adaptive quantization*. We observe that adaptive quantization achieves similar final accuracy as that achieved by always using 8-bit quantization, higher than that by always using 2bit quantization. Thus, using lower quantization resolution at later training stage may reduce the total training time (due to less bits transmitted) without degrading the performance. We also have similar observations for GoogLeNet, but not shown due to space limitations.

To further reduce the training time, we study the heterogeneity of the edge devices. Given the heterogeneous communication capability of the edge devices (clients), the training time depends on the slow clients with poor network conditions. To mitigate such straggler problem, we investigate heterogeneous gradient quantization strategies, which use less quantization resolutions for slower clients. Specifically, we train ResNet-18 on CIFAR-10 with four clients: three clients with data transmission rate of 20 Mbps, and one client (straggler) with data transmission rate of 5 Mbps. We evaluate the traditional quantization strategy that uses 6-bit quantization for all clients, and compare it with four heterogeneous quantization strategies by letting the slowest client use 2-bit, 3-bit, 4-bit, and 5-bit quantization, respectively.

Fig. 2(a) shows the total training time of different quantization strategies, to reach the same accuracy of 85.0% (near convergence). We observe that 3-bit, 4-bit and 5-bit quantization strategies all outperform the traditional approach, and the 4-bit quantization strategy has the lowest training time. To find out how the heterogeneous quantization strategies reduce the training time, we draw accuracy as a function of training rounds for different strategies, as shown in Fig. 2(b). From the figure, we observe that 2-bit quantization takes 76 communication rounds to reach the accuracy of 85%, while the traditional quantization takes 51 rounds. However, in the 2-bit quantization strategy, less quantization is used and each round takes less time. As a result, it has similar accumulated time to reach 85% accuracy as that of the traditional quantization (as shown in Fig. 2(a)). Although the 4-bit quantization strategy has 6 more rounds than the traditional quantization strategy, it can significantly reduce the training time since each round takes less time. In summary, we should consider both perround communication time and the number of communication rounds when determining the quantization resolution for heterogeneous clients.

These evaluation results show the potential of using adaptive and heterogeneous quantization to reduce the training time without compromising the model accuracy. However, it is hard to quantify the relationship between the quantization resolution and the training time. For example, clients may have various transmission rates and it is hard to know which clients are the bottleneck at which time, and thus it is a challenge to assign quantization resolutions to heterogeneous clients to minimize the overall training time. In the next section, we propose AdaGQ, an adaptive and heterogeneous gradient quantization algorithm that exploits online learning to adaptively adjust the quantization resolutions based on the gradient norms and the local training and transmission time of the clients.

## III. DESIGN OF ADAGQ

The main challenges of designing AdaGQ are: (1) How to integrate gradient norm with the algorithm to minimize the total training time. To address this challenge, AdaGO dynamically adjusts the number of quantization levels assigned to the clients based on the observed change of gradient norm. Specifically, when observing a larger gradient norm, AdaGQ tends to increase the number of quantization levels to preserve the precision of the gradients to reduce the number of training rounds; while for a smaller gradient norm, AdaGQ assigns fewer numbers of quantization levels to the clients to reduce the communication time, and thus reducing the total training time. (2) How to deal with the training time bottleneck brought by the slowest clients (i.e., straggler effects). To address this challenge, AdaGQ assigns different numbers of quantization levels to different clients based on their computation and communication resources. Intuitively, slow clients (i.e., with less resources) are assigned relatively fewer number of quantization levels to reduce the communication time to mitigate the straggler effects, while fast clients are assigned more quantization levels to reduce the precision loss due to quantized gradients, and then to reduce the number of training rounds. In the following, we first give an overview of AdaGQ, and then present the details of AdaGQ.

## A. Overview of AdaGQ

AdaGQ follows the system design of the state-of-the-art FL system [6], [22] by adopting the adaptive and heterogeneous quantization. Fig. 3 gives an overview of AdaGQ. In step 1, the server sends clients the aggregated gradients collected in the last round to synchronize the saved model parameters. In step 2, the clients collect necessary inputs of the AdaGQ algorithm, e.g., the losses achieved by the model when updated by gradients of different quantization levels and their corresponding training time, which are then sent to the server.



Fig. 3. Overview of AdaGQ.

In step 3 (a), the clients apply stochastic gradient descent to the updated model (with the aggregated gradients received in step 1) and obtain the gradients. Meanwhile, in step 3 (b), the server derives the number of quantization levels for each client with the collected information in step 2. In step 4, the server sends to each client its own number of quantization levels in this round. Finally, each client quantizes the gradients derived in step 3 (a) and sends them back to the server.

We emphasize the novel parts in the AdaGQ design with bold fonts in Fig. 3. Note that AdaGQ collects the necessary algorithm inputs in step 2 and follows an algorithm in step 3 to derive quantization levels for all the clients. More specifically, AdaGQ algorithm first determines the average quantization level of all clients in the current round based on both the loss decrease rate and the change of the gradient norm to facilitate adaptive quantization. Then, AdaGQ derives the quantization levels for heterogeneous clients. In the following, we present the adaptive and heterogeneous quantization in detail, respectively.

# B. Adaptive Quantization

AdaGQ adpats the average number of quantization levels of all clients to minimize the total training time in two steps: (i) to maximize the *loss decrease rate*, and (ii) to calibrate the adaptation in (i) based on the change of the gradient norm.

Let  $s_{i,k}$  denote the number of quantization levels used by client *i* at round *k*, let  $s_k$  denote the average number of quantization levels at round *k*, i.e.,  $s_k = \frac{1}{n} \sum_{i=1}^n s_{i,k}$ , where *n* is the number of clients. Note that  $s_k$  is introduced to assist the design of adaptive quantization and it does not have to be an integer. In the first step, we adapt  $s_k$  to minimize the training time by finding the optimal average number of quantization level  $s_k^*$ . Then, we optimize the *loss decrease rate* of each round, defined as

$$R_k = (L_{k-1} - L_k)/T_{k-1,k},$$
(5)

where  $L_k$  denotes the average loss of all clients achieved at the end of round k, and  $T_{k-1,k}$  denotes the elapsed time between the end of round (k-1) and that of round k. Suppose  $R_k^*$  is the loss decrease rate achieved by  $s_k^*$ , we first construct the loss function

$$f(s_k) = R_k^* - R_k.$$
 (6)

and then explore the idea of online gradient descent based algorithms to use the derivative of the loss function to indicate the direction of the optimal solution, as shown below:

$$s_{k+1} = s_k - \lambda \nabla f(s_k),\tag{7}$$

where  $\lambda$  is the step size (i.e., learning rate) to update  $s_k$ , and  $\nabla f(s_k)$  denotes the derivative of the loss function at  $R_k$ . In practice, it is impossible to obtain the exact value of the derivative  $\nabla f(s_k)$  due to the unknown form of  $f(s_k)$ . Thus, we obtain the sign of the derivative  $\nabla f(s_k)$  which indicates the update direction, instead of the exact value.

In order to obtain the sign of  $\nabla f(s_k)$ , besides the current used  $s_k$ , we use another quantization level  $s'_k$ , which is slightly lower than  $s_k$ , and record the loss decrease rate  $R'_k$  achieved by  $s'_k$ . Then, the sign of  $\nabla f(s_k)$  is derived as

$$sign(\nabla f(s_k)) = sign(\frac{R'_k - R_k}{s_k - s'_k}).$$
(8)

The details of obtaining  $R'_k$  will be explained in Section III-D. After obtaining the derivative sign, our algorithm updates  $s_k$  to the opposite direction of the sign. That is

$$\begin{cases} \hat{s}_{k+1} = s_k - \lambda_1, & if \quad sign(\nabla f(s_k)) = 1\\ \hat{s}_{k+1} = s_k + \lambda_2, & if \quad sign(\nabla f(s_k)) = -1. \end{cases}$$
(9)

where  $\lambda_1$  is set as half of  $s_k$  so that  $\hat{s}_{k+1} = s_k/2$  has one fewer bit than  $s_k$ , and  $\lambda_2$  is set as the same of  $s_k$  so that  $\hat{s}_{k+1} = s_k \times 2$  has one more bit than  $s_k$ . Note that  $\lambda_1$  and  $\lambda_2$  are not designed as constants, and AdaGQ will explore a larger range of  $s_k$ , by increasing or decreasing the number of bits by 1 at a time, and quickly approach to a better setting.

We calibrate  $s_{k+1}$  with the change of the gradient norm. We estimate the change of the gradient norm from round k to (k+1) by the observed gradient norm change from round (k-1) to k. We increase  $s_{k+1}$  when expecting a rise of the gradient norm and decrease  $s_{k+1}$  otherwise. By denoting the aggregated quantized gradients by the server at the end of round k as  $\mathbf{g}_k$  and its norm as  $||\mathbf{g}_k||$ , we calibrate  $\hat{s}_{k+1}$  to be  $s_{k+1}$  by,

$$s_{k+1} = \hat{s}_{k+1} + \lambda_{\mathbf{g}} (\log_2 ||\mathbf{g}_k|| - \log_2 ||\mathbf{g}_{k-1}||)$$
(10)

where  $\lambda_{g}$  is the learning rate to weight gradient norm change.

#### C. Heterogeneous Quantization

The goal of heterogeneous quantization is to let the server receive the gradients of each client at similar times, so that the waiting time of fast clients and server is minimized. We first derive the relationships of the number of quantization levels among clients.

For a client *i*, its *local time*  $t_{i,k}^r$  in the training round k consists of its local training time  $t_{i,k}^{cp}$  spent on stochastic gradient descent to derive new gradients, and the communication time  $t_{i,k}^{cm}$  spent on sending the quantized gradients to the server. Let  $b_{i,k}$  denote the number of bits for a quantized gradient (referred to as quantization bit), which means  $b_{i,k} = \lfloor \log_2(s_{i,k}) + 1 \rfloor$ . The server determines the number

of quantization level for each client in the round (k + 1) as follows.

$$\mathbb{E}(t_{i,k+1}^{r}) = \mathbb{E}(t_{i,k+1}^{cp} + t_{i,k+1}^{cm}) \approx \mathbb{E}(t_{i,k+1}^{cp}) + b_{i,k+1} \times \mathbb{E}(\frac{P}{r_{i,k+1}^{trans}}),$$
(11)

where P denotes the number of gradients which is a constant (same for all clients in all rounds), and  $r_{i,k+1}^{trans}$  denotes the transmission rate of client i in round (k + 1). Then, our goal is to make the expected local time of each client as similar as possible, i.e., to satisfy the condition  $\mathbb{E}(t_{1,k+1}^r) = \mathbb{E}(t_{2,k+1}^r) =$  $\cdots = \mathbb{E}(t_{n,k+1}^r)$ . Note that we omit the time for the server to broadcast the aggregated gradients to clients since it is relatively small. By introducing Eq. (11) as the condition, for any two clients i and j, their quantization bits  $b_{i,k+1}$  and  $b_{j,k+1}$ should satisfy the following:

$$b_{j,k+1} = \frac{1}{\mathbb{E}(\frac{P}{r_{j,k+1}^{trans}})} (\mathbb{E}(t_{i,k+1}^{cp}) - \mathbb{E}(t_{j,k+1}^{cp}) + b_{i,k+1} \times \mathbb{E}(\frac{P}{r_{i,k+1}^{trans}}))$$
(12)

Local training time and transmission rate estimation. In practice, in order to assign  $b_{i,k+1}$  to every client, we have to estimate the local training time  $\mathbb{E}(t_{i,k+1}^{cp})$  and the transmission time coefficient  $\mathbb{E}(\frac{P}{r_{i,k+1}^{trans}})$ . Since the per-round local training time of a client does not vary much,  $\mathbb{E}(t_{i,k+1}^{cp})$  is estimated by the average of all the historical local training times spent by client *i*, i.e.,  $\mathbb{E}(t_{i,k+1}^{cp}) = \frac{1}{k} \sum_{k'=1}^{k} t_{i,k'}^{cp}$ .

On the other hand, the transmission rate may have variations over different training rounds (but usually smooth) and thus we estimate  $\mathbb{E}(\frac{P}{r_{i,k+1}^{trans}})$  based on the same transmission rate of last round, i.e.,  $\mathbb{E}(\frac{P}{r_{i,k+1}^{trans}}) \approx \frac{P}{r_{i,k}^{trans}} = t_{i,k}^{cm}/b_{i,k}$ . Given the number of quantization bits (levels) of one client (e.g., client *i*), the number of quantization bits of other clients can be determined as follows.

$$b_{j,k+1} = \frac{b_{j,k}}{t_{j,k}^{cm}} (\frac{1}{k} \sum_{k'=1}^{k} t_{i,k'}^{cp} - \frac{1}{k} \sum_{k'=1}^{k-1} t_{j,k'}^{cp} + b_{i,k+1} \times \frac{t_{i,k}^{cm}}{b_{i,k}}), \quad (13)$$
$$\forall j \in \{1, \cdots, n\}, j \neq i.$$

where  $b_{i,k+1} = \lfloor \log_2(s_{i,k+1}) + 1 \rfloor$ , for i = 1, 2, ..., n, and  $\frac{1}{n} \sum_{i=1}^{n} s_{i,k+1} = s_{k+1}$ . Thus, we can derive  $b_{i,k+1}$  from Eq. (13) and refine  $s_{i,k+1}$  as  $(2^{b_{i,k+1}} - 1)$ . Once the server determines the number of quantization levels for client i (i.e.,  $s_{i,k+1}$ ), it sends  $s_{i,k+1}$  to client i as its quantization in round (k+1).

# D. Implementation of AdaGQ

As an example, we describe how our algorithm runs in a round (k + 1). As shown in Fig. 4, at the beginning of round (k + 1), the server broadcasts the aggregated gradients  $\mathbf{g}_k$  obtained in the last round (k), and a parameter  $s'_{i,k}$  (introduced later) to the clients (step 1). Next, to estimate  $sign(\nabla f(s_k))$ , AdaGQ has to estimate  $R_k$  and  $R'_k$  based on Eq.(8). Recall that the loss decrease rate  $R_k$  is defined in Eq. (5). Thus, AdaGQ needs to estimate the average loss  $L_{k-1}$ ,  $L_k$  and the round time  $T_{k-1,k}$  to derive  $R_k$ . In addition, AdaGQ needs to estimate  $R'_k$ , which is the loss decrease rate if  $s'_k$  was used instead of  $s_k$  in the round k, requiring further estimation of  $L'_k$ 



Fig. 4. Timeline in Round (k + 1) of AdaGQ.

and  $T'_{k-1,k}$ . Because these average losses and the round times may not be easily measured, AdaGQ estimates their values in step 2. After client i receives the aggregated gradients  $\mathbf{g}_k$ from the server, it quantizes  $g_k$  with  $s_{i,k}$  (i.e., the number of quantization level assigned to client i with  $s_k$ ) and  $s'_{i,k}$ (i.e., the number of quantization level assigned to client *i* with  $s'_k$ ) quantization levels, respectively. Here  $s'_k$  is an auxiliary selected by the server by  $s'_k = \lfloor s_k/2 \rfloor$  (i.e., one bit fewer than  $s_k$ ), and  $s'_{i,k}$  is derived from  $s'_k$  following the same way in which the server derived  $s_{i,k}$  from  $s_k$  in round k. Suppose the obtained quantized gradients are denoted as  $\mathbf{g}_{i,k}^{s}$  and  $\mathbf{g}_{i,k}^{s'}$ The client *i* computes two losses  $L_{i,k}$  and  $L'_{i,k}$ , which are the losses obtained by the models updated with  $\mathbf{g}_{i,k}^s$  and  $\mathbf{g}_{i,k}^s$ respectively. Next, client i uploads the calculated losses  $L_{i,k}$ and  $L'_{i,k}$ , as well as the parameters downloading time  $t^{down}_{i,k}$ , local computation time  $t^{cp}_{i,k}$  and communication time  $t^{cm}_{i,k}$  in round k, to the server. Then, the clients conduct a new round of model training SGD (step 3 (a)).

After the server receives the information from all clients, at the same time with step 3 (a), it needs to compute two estimated loss decrease rates  $R_k$  and  $R'_k$  (step 3 (b)). The server first derives the estimation of the average loss  $L_k$  by averaging the loss  $L_{i,k}$  collected from all the clients, i.e.,  $\bar{L}_k = \frac{1}{n} \sum_{i=1}^n L_{i,k}$  where *n* is the number of the clients. Similarly, the server estimates  $L'_k$  by  $\bar{L}'_k = \frac{1}{n} \sum_{i=1}^n L'_{i,k}$ . As the server saves the status of model parameters and gradients at the beginning of round *k*, it can easily obtain the real loss  $L_{k-1}$  for Eq. (5).

The server estimates the average time of training round k when clients quantize the gradients under the condition of  $s_k$  and  $s'_k$ , denoted as  $T_{k-1,k}$  and  $T'_{k-1,k}$ , respectively.  $T_{k-1,k}$  is determined by the slowest client in round k, i.e., the maximum time spent among all clients, and it is obtained by the server as follows:

$$T_{k-1,k} = \max_{i} \{ t_{i,k}^{cp} + t_{i,k}^{cm} + t_{i,k}^{down} \} + t_{k}^{server}.$$
 (14)

To estimate  $T'_{k-1,k}$ , the main challenge is to estimate the transmission time  $t'_{i,k}^{cm}$  since the main change is the number of bits for transmission. To bridge this gap, AdaGQ computes the number of bits when using  $s_{i,k}$  by  $b_{i,k} = \lfloor \log_2(s_{i,k}) \rfloor + 1$ , and that of using  $s'_{i,k}$  by  $b'_{i,k} = \lfloor \log_2(s'_{i,k}) \rfloor + 1$ . Then the

transmission time  $t'_{i,k}^{cm}$  can be estimated as  $\frac{b'_{i,k}}{b_{i,k}}t_{i,k}^{cm}$ , and thus AdaGQ estimates the training round time  $T'_{k-1,k}$  of using  $s'_{k-1}$  as follows.

$$T'_{k-1,k} = \max_{i} \{ t^{cp}_{i,k} + \frac{\lfloor \log_2(s'_{i,k}) \rfloor + 1}{\lfloor \log_2(s_{i,k}) \rfloor + 1} t^{cm}_{i,k} + t^{down}_{i,k} \} + t^{server}_k.$$
(15)

Then, the server estimates the two loss decrease rates  $R_k$  and  $R'_k$  with Eq. (5) as follows.

$$R_k = (L_{k-1} - \bar{L}_k)/T_{k-1,k}, \quad R'_k = (L_{k-1} - \bar{L}'_k)/T'_{k-1,k}$$
(16)

The server estimates the sign of  $\nabla f(s_{k-1})$  following Eq. (8) using  $R_k$  and  $R'_k$ . To obtain  $s_{k+1}$ , the next step in step 3 (b) is to compute the gradient norm of the aggregated gradients (i.e.,  $\mathbf{g}_k$ ) and update  $s_{k+1}$  following Eq. (9) and Eq. (10). Finally, the server derives  $b_{i,k+1}$  and  $s_{i,k+1}$ , for  $i = 1, 2, \dots, n$ , as introduced in Section III-C.

After the client *i* receives  $s_{i,k+1}$  from the server (step 4) and finishes computing the new model gradients in the current round, client *i* quantizes the newly computed gradients in  $s_{i,k+1}$  quantization levels. The quantized gradients are then uploaded to the server (step 5). Finally, the server collects all quantized gradients from all clients, and conducts a global aggregation on these gradients to generate  $\mathbf{g}_{k+1}$  in step 6. The server also prepares  $s'_{k+1}$  as  $\lfloor s_{k+1}/2 \rfloor$ , derives  $s'_{i,k+1}$  for i = 1, 2, ..., n, and sends it to the clients in the next round, i.e., round (k + 2). The details are shown in Algorithm 2.

# **IV. PERFORMANCE EVALUATIONS**

In this section, we evaluate AdaGQ against four baselines under four federated learning (FL) tasks. We first introduce the evaluation setup and then present the evaluation results of all algorithms based on four FL tasks.

# A. Evaluation Setup

We evaluate the proposed algorithm with non-iid data distribution at 20 clients on various learning tasks and compare its performance with state-of-the-art algorithms.

Models and datasets. We consider two model architectures with different parameter sizes: ResNet-18 [23] and GoogLeNet [24]. ResNet-18 is a CNN network consisting of residual blocks with over 11 million parameters. GoogLeNet is a 22-layer CNN network without any skip connections, which has over 6 million parameters. We evaluate AdaGQ by training ResNet-18 and GoogLeNet on two benchmark datasets: Cifar-10 [25] and FashionMNIST (FMNIST) [26]. The CIFAR-10 dataset consists of 50K color images as the training set and 10K color images for testing, where each image belongs to one of the 10 classes. The FMNIST dataset contains 60K train and 10K test grey scale images of 10 different fashion items. Four FL tasks are used in the evaluation, i.e., training ResNet-18 on Cifar-10, training ResNet-18 on FMNIST, training GoogLeNet on Cifar, and training GoogLeNet on FMNIST (all with crossentropy loss function).

**Methods for comparison.** We compare AdaGQ with the following four baseline approaches.

# Algorithm 1: The AdaGQ Algorithm

1	1 <b>Initialization:</b> global model weight $\mathbf{w}_0$ ; initial model					
	weights of cl	weights of clients $\mathbf{w}_{i,0} = \mathbf{w}_0, \forall i \in \{1, \dots, n\}$ ; initial				
	number of qu	number of quantization levels $s_0$ ;				
	$s_{i,0} = s_0, \forall i$	$\in \{1, \cdots, n\}.$				
2	2 for each $k =$	$1, 2, \cdots$ <b>do</b>				
3	3   for each a	client $i = 1, 2, \cdots, n$ in parallel <b>do</b>				
4	4   $\mathbf{g}_k \leftarrow$	receives aggregated gradients from server;				
5	5 $\mathbf{w}_{i,k+}$	$_1 \leftarrow$ update model parameters with $\mathbf{g}_k$ ;				
6	6 $\mathbf{g}_{i,k}^s$ ,	$\mathbf{g}_{i,k}^{s'} \leftarrow \text{quantize gradients;}$				
7	7 $\mathbf{w}'_i, \mathbf{w}'_i$	$J''_i \leftarrow$ model parameters when updated with				
	$\mathbf{g}_{i,k}^{s}$	$\mathbf{g}_{i,k}^{s'};$				
8	8 L <sub>i,k</sub> ,	$L'_{i,k} \leftarrow \text{losses of } \mathbf{w}'_i \text{ and } \mathbf{w}''_i \text{ in local test set;}$				
9	9 Send	$L_{i,k}, L'_{i,k}, t^{down}_{i,k}, t^{cp}_{i,k}, t^{cm}_{i,k}$ to server;				
10	$g(\mathbf{w}_{i})$	$_{k+1}) \leftarrow \text{Perform SGD locally;}$				
11	1 Recei	ve $s_{i,k+1}$ from server;				
12	$Q_{s_{i,k}}$	$_{i+1}(\mathbf{g}_{i,k+1}) \leftarrow \text{Perform gradient quantization;}$				
13	13 Uploa	d $Q_{s_{i,k+1}}(\mathbf{g}_{i,k+1})$ to server;				
14	4 record	$t_{i,k+1}^{down}, t_{i,k+1}^{cp}, t_{i,k+1}^{cm}$ of this round;				
15	5 end					
16	6 The serve	er does:				
17	7 Send th	e aggregated gradients $\mathbf{g}_k$ together with $s'_{i,k}$ ,				
	i = 1, 2,	$\cdots$ , <i>n</i> to the clients;				
18	8 Receive	$L_{i,k}, L'_{i,k}, t^{down}_{i,k+1}, t^{cp}_{i,k}, t^{cm}_{i,k}$ from all clients;				
19	9 Comput	Compute the expected loss decrease rate $R_k$ , $R'_k$				
	following	g Eq. (16);				
20	o Comput	e $s_{k+1}$ following Eq. (10);				
21	Comput	ie $s_{i,k+1}$ following Eq. (13) and send				
	$s_{i,k+1}, i$	$= 1, 2, \cdots, n$ to clients;				
22	2 Gather	$Q_{s_{i,k+1}}(\mathbf{g}_{i,k+1}), i = 1, 2, \cdots, n;$				
23	$  \mathbf{g}_{\mathbf{k}+1}  $	$  \leftarrow \text{Computes gradient norm};$				
24	$\mathbf{g}_{k+1} \leftarrow \mathbf{g}_{k+1}$	- Aggregate gradients by				
	$\sum_{i=1}^{n} p_i$	$\mathcal{Q}_{s_{i,k+1}}(\mathbf{g}_{i,k+1});$				
25	Set $s'_{k+}$	$s_1$ as $\lfloor s_{k+1}/2 \rfloor$ and derive $s_{i,k+1}$ for				
~	i = 1, 2,	$\dots, n$ based on Eq. (13).				
26 —	6 enu					



Fig. 5. Accuracy v.s. accumulated time of AdaGQ compared to baselines.

• FedAvg [9]: clients communicate updated local parameters with the central server after multiple epochs of local





(d) GoogLeNet on FMNIST Fig. 6. Total training time of AdaGQ compared to baselines.

training and download the aggregated global model. Here the communication period is set to be 5 epochs.

- **QSGD** [12]: clients send quantized gradients to the central server and download the aggregated global model for every epoch. The number of quantization levels is set to be 8-bit.
- Top-k [10] is a sparsification method that compresses the communicated gradients by selecting the largest kelements of the gradients. In this method, clients send sparse gradients to the central server and download the aggregated global model every epoch. We set k to be 10% of the total parameters.
- FedPAQ [13] incorporates periodic averaging into QSGD. In FedPAQ, models are trained multiple epochs at clients and only periodically averaged at the server. Clients quantize their updates before uploading. Similar to FedAvg, we set the communication period to be 5 epochs and the number of quantization levels to be 8bit.

**Hyperparameters.** As the default configuration, we set the local batch size to 32 and assign every client an equally sized subset of the training data. For each client, the data transmission rate is initialized to be a rate sampled randomly between 5 Mbps and 20 Mbps by default. We set the initial learning rates for both ResNet-18 and GoogLeNet to be 0.01 and the decay as 0.995. For AdaGO, the initial number of quantization levels is set to be 8-bit, which is relatively large as suggested in [12]. The step size  $\lambda_g$  is set to 1.

Similar to the definition in [19], we use  $\sigma_d$  to denote the level of non-iid data, which corresponds to the fraction of data that only belongs to one class at each client. For example,  $\sigma_d = 0.2$  means that 20% of the data on one client belongs to one class and the remaining 80% of the data uniformly belongs to other classes. For the baselines in comparison, we set their hyper-parameters (as shown above) the same as those suggested in the corresponding literature.

We evaluate all the algorithms in terms of the total wallclock training time, including computation time, communication time and all extra overhead, when they reach the target accuracy.

$\sigma_d$	Method	Avg. rounds	Avg. data uploaded	Total time (Second)
	FedAvg	13.25	6.52 (1×)	2190.89± 40.43 (1×)
	QSGD	44.50	5.28 (1.23×)	1891.91± 58.21 (1.16×)
0.2	Top-k	40.75	3.92 (1.67×)	1568.57± 62.43 (1.40×)
0.2	FedPAQ	19.25	2.33 (2.80×)	2014.05± 46.32 (1.09×)
	AdaGQ	43.25	3.43 (1.90×)	1033.50± 49.86 (2.12×)
	FedAvg	14.25	7.01 (1×)	2359.41± 44.10 (1×)
	QSGD	48.00	5.72 (1.23×)	$2018.04 \pm 61.20 \ (1.17 \times)$
0.5	Top-k	45.75	4.40 (1.59×)	$1759.86 \pm 64.49 \ (1.34 \times)$
0.5	FedPAQ	21.25	2.57 (2.73×)	2226.06± 47.24 (1.06×)
	AdaGQ	47.00	3.73 (1.88×)	$1129.60 \pm 53.56 \ (2.09 \times)$
	FedAvg	20.25	9.96 (1×)	3370.59 ± 49.25 (1.26×)
	QSGD	69.75	8.28 (1.20×)	$2942.98 \pm 65.32 \ (1.44 \times)$
0.8	Top-k	60.50	5.82 (1.71×)	$2295.50 \pm 71.10 \ (1.85 \times)$
0.8	FedPAQ	39.75	4.81 (2.07×)	$4240.11 \pm 48.20 (1 \times)$
	AdaGQ	68.00	5.40 (1.85×)	$1634.31 \pm 56.71 \; (2.59 \times)$
			TABLE I	

ResNet-18 on Cifar-10 under different  $\sigma_d$ 

### B. Comparison of the Algorithms

First, we compare the wall-clock time of AdaGQ with all baseline algorithms when they reach the same accuracy (with  $\sigma_d = 0.5$ ). Fig. 5 shows the accuracy over accumulated time of the four FL tasks respectively. We observe that AdaGQ takes the least amount of time to reach the accuracy of 85.0% for Cifar-10 and 87.0% for FMNIST, reducing the training time by 29.1%-34.8% compared to the best baselines (i.e., Top-k in Fig. 5(a)(c) or QSGD in Fig. 5(b)(d)), and 45.5%-52.1% compared to FedAvg under the four FL tasks.

Among the baselines, FedPAQ spends longer time than most of other baselines and fails to reach the target accuracy when training GoogLeNet on Cifar-10 and FMNIST. This is because FedPAQ incorporates both periodic averaging and gradient quantization, which incur more information loss in each round to delay the convergence. In addition, we observe that Topk and QSGD spend less training time than FedAvg, which suggests that the gradient compression can save more time than periodic averaging. By comparing AdaGQ with Top-k and QSGD, we observe that AdaGQ outperforms Top-k and QSGD consistently on all four FL tasks, which validates the advantages of adaptive and heterogeneous quantization.

To analyze how AdaGQ reduces the total training time, we separate the communication time and the computation time for all the algorithms, as shown in Fig. 6. We observe that AdaGQ spends similar computation time but significantly less communication time compared to QSGD (the second best algorithm). Because the computation time spent in each round is similar for both algorithms, having similar computation time indicates that both algorithms take similar numbers of rounds to reach the target accuracy. However, AdaGQ saves the communication time in each round, by adjusting the number of quantization levels based on the adaptive and heterogeneous quantization, and thus reduces the accumulated wall-clock time. In addition, among all the baselines, we observe that FedPAQ has the longest computation time, which verifies that it takes more rounds to reach the same accuracy. Though FedPAQ reduces the communication time of each round aggressively, the increased number of training rounds

$\sigma_d$	Method	Avg. rounds	Avg. data uploaded	Total time (Second)
	FedAvg	19.75	5.46 (1×)	2538.56 ± 50.21 (1.31×)
	QSGD	70.50	4.73 (1.15×)	$2144.46 \pm 43.24 \ (1.56 \times)$
0.2	Top-k	78.50	4.21 (1.30×)	$2187.28 \pm 48.43 \ (1.53 \times)$
0.2	FedPAQ	37.75	2.44 (2.24×)	$3338.05 \pm 52.42 \ (1 \times)$
	AdaGQ	69.25	2.39 (2.29×)	1295.97 $\pm$ 40.10 (2.58 $ imes$ )
-	FedAvg	23.00	6.36 (1×)	2919.34 ± 53.90 (1.35×)
	QSGD	82.25	5.50 (1.16×)	$2476.70 \pm 46.32 (1.60 \times)$
0.5	Top-k	85.50	4.61 (1.38×)	$2411.62 \pm 51.20 \ (1.64 \times)$
0.5	FedPAQ	45.25	2.91 (2.19×)	$3952.95 \pm 58.23 \ (1 \times)$
	AdaGQ	82.75	2.86 (2.22×)	$1558.92 \pm 42.36 \; (2.54 \times)$
	FedAvg	28.25	7.81 (1.26×)	3553.98 ± 57.30 (1.31×)
	QSGD	146.50	9.83 (1×)	$4409.73 \pm 50.54 (1.06 \times)$
0.8	Top-k	122.50	6.57 (1.50×)	$3421.14 \pm 54.32 \ (1.36 \times)$
0.8	FedPAQ	52.75	3.41 (2.88×)	$4655.70 \pm 63.47 \ (1 \times)$
	AdaGQ	142.25	<b>4.90</b> (2.00×)	<b>2667.07</b> $\pm$ <b>48.92</b> (1.75 $\times$ )
			TABLE II	

GOOGLENET ON CIFAR-10 UNDER DIFFERENT  $\sigma_d$ 

makes the total training time longer than others.

## C. Different Levels of Non-IID Data

In this section, we evaluate AdaGQ under different levels of non-iid data (i.e., different  $\sigma_d$ ) against the baselines. Table I and Table II show the results by training ResNet-18 and GoogLeNet on Cifar-10, respectively. Owing to space limitation, we do not present the results on FMNIST, which share similar observations as those on Cifar-10. For each FL task, we evaluate the algorithms in terms of the total number of communication rounds, the average amount of uploaded data per client (in GB), and the total time (in second) to reach the same accuracy. We repeat the evaluation four times and report the average of those metrics.

From the tables we observe that AdaGQ outperforms all baseline algorithms in terms of total time under various non-iid levels. Among the baselines, FedAvg has the fewest communication rounds under all levels of non-iid data due to the periodic averaging to reduce the communication frequency. Here, we clarify that FedAvg (and also FedPAQ) has five epochs in each round, so the total number of epochs is five times the number of communication rounds. For example, in Table I when  $\sigma_d = 0.5$ , FedAvg has about 71 (14.25  $\times$  5) epochs which imply a longer computation time than QSGD  $(\sim 48)$ , Top-k  $(\sim 46)$  and AdaGQ  $(\sim 47)$ . Meanwhile, without any gradient compression, the amount of data transmitted each round in FedAvg is much higher than other algorithms, leading to longer communication time and thus longer total time. In addition, we observe that FedPAQ has the most training epochs, which results in the longest computation time and a long total time. An interesting observation is that, in Table I with  $\sigma_d = 0.8$ , AdaGQ has more communication rounds than Top-k (68.0 v.s. 60.5) and 5.92% less data uploaded, while achieving 28.8% less total time. Such a big improvement may be attributed to the heterogeneous quantization. Although AdaGQ has 5.40GB data uploaded on average per client, the slowest clients may have much less data to upload, which greatly reduces the communication overhead caused by waiting for the slowest clients in each round.

$ \begin{array}{r c c c c c c c c c c c c c c c c c c c$	$\sigma_r$	Method	Avg. rounds	Avg. data uploaded	Total time (Second)
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$		FedAvg	14.25	7.13 (1×)	1742.20 ± 38.29 (1.11 ×)
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$		QSGD	45.75	5.49 (1.30×)	$1488.89 \pm 55.32 \ (1.30 \times)$
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	2	Top-k	46.00	4.47 (1.60×)	$1232.43 \pm 56.49 (1.57 \times)$
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	2	FedPAQ	22.50	2.75 (2.60×)	$1938.03 \pm 51.35 (1 \times)$
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$		AdaGQ	43.75	<b>4.07</b> (1.75×)	913.70 $\pm$ 47.24 (2.12 $ imes$ )
$4 \begin{array}{ c c c c c c c c c c c c c c c c c c c$		FedAvg	14.50	7.26 (1×)	2378.65± 43.16 (1×)
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$		QSGD	47.50	5.64 (1.29×)	$2176.67 \pm 66.50 (1.09 \times)$
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	4	Top-k	45.75	4.38 (1.66×)	$1763.90 \pm 67.82 \ (1.35 \times)$
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	4	FedPAQ	23.00	2.81 (2.58×)	2269.42± 52.27 (1.05×)
$6 \begin{array}{c ccccccccccccccccccccccccccccccccccc$		AdaGQ	47.75	3.78 (1.92×)	1134.63 $\pm$ 55.08 (2.10 $\times$ )
		FedAvg	14.75	7.38 (1×)	2996.6 ± 48.23 (1×)
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$		QSGD	48.00	5.84 (1.26×)	$2889.67 \pm 68.90 (1.04 \times)$
$\begin{tabular}{ c c c c c c c c c c c c c c c c c c c$	6	Top-k	47.50	4.66 (1.58×)	$2321.29 \pm 66.35 (1.29 \times)$
AdaGQ         53.25         3.87 (1.91×)         1419.71 $\pm$ 61.34 (2.11×)	0	FedPAQ	23.25	2.75 (2.68×)	$2514.09 \pm 52.54 (1.19 \times)$
		AdaGQ	53.25	3.87 (1.91×)	$1419.71 \pm 61.34 \; (2.11 \times)$

TABLE III

The performance of Resnet-18 on Cifar-10 under different level of resource heterogeneity

The non-iid level of data distribution affects the convergence speed of training. A higher level of non-iid data decreases the convergence speed in general, which results in more communication rounds, thus more data uploaded and longer total time. For example, the average number of communication rounds of AdaGQ when training ResNet-18 on Cifar-10 with non-iid levels of 0.2, 0.5, and 0.8 are 43.25, 47.00, and 68.00, respectively, which is increasing. Similar conclusion is also suggested by other algorithms.

#### D. Levels of Resource Heterogeneity

In this section, we evaluate AdaGQ under different levels of resource heterogeneity. To isolate the effects of resource heterogeneity, we fixed the dataset of each client with a noniid level to be 0.5 for each running of the experiment. We define the resource heterogeneity level  $\sigma_r$  to be the ratio of the data transmission rate of the fastest client and that of the slowest client. We set the transmission rate of the fastest client to be 20Mbps, and the slowest client to be  $20/\sigma_r$  Mbps, and the transmission rates of other clients are sampled randomly between  $[20/\sigma_r, 20]$  Mbps. Similar to Section IV-C, we repeat the evaluation four times and report the average of the metrics. We only present the results for training ResNet-18 on Cifar-10 (in Table III), since other FL tasks share similar observations.

For FedAvg, QSGD, Top-k and FedPAQ, the resource heterogeneity does not affect the number of their communication rounds and the amount of uploaded data, and only changes the communication time of each round due to the delay of aggregation caused by the slowest client. AdaGQ is able to adapt the number of quantization levels based on the clients' resources, thus reducing more total training time under higher resource heterogeneity. For example, when training ResNet-18 on Cifar-10, AdaGQ reduces the total time by 38.8% compared to Top-k (the second best algorithm) when  $\sigma_r = 6$ , which is higher than 25.9% that is achieved when  $\sigma_r = 2$ .

## V. RELATED WORK

**Communication-efficient federated learning.** FL has been widely deployed for mobile and IoT devices. To reduce the

communication bottleneck, various methods have been proposed which fall into two main categories. The first category reduces the communication overhead by periodic averaging which allows clients to perform multiple rounds of local updates and upload the updates less frequently [8], [9], [17], [27]. The second category of research solves this problem by reducing the communication overhead of every communication round [11], [12], [28]–[33]. In this category, a variety of compression schemes have been proposed, including gradient quantization [11], [12], [34], gradient sparsification [28], [30], [35] and low-rank approximation [29]. Seide et al. [34] replaced each weight with just the sign values. Similarly, Wen et al. [11] proposed TernGrad which requires three numerical levels  $\{-1, 0, 1\}$ , to aggressively reduce the communication time. However, these two gradient quantization algorithms lack flexibility in controlling the resolution of quantization. Alistarh et al. [12] proposed quantized SGD (QSGD) that can adjust the number of bits (i.e., quantization resolution) sent per iteration to reduce the bandwidth cost, which provides more flexibility. However, how to find the optimal quantization resolution is not studied. Han et al. [30] proposed an adaptive approach for gradient sparsification (i.e., Top-k) to achieve the near-optimal communication and computation trade-off by controlling the degree of gradient sparsity. Although they seek to find the optimal degree of gradient sparsity, the optimal value is assumed to be fixed. Besides, there are also some literature combines the two directions by integrating gradient quantization in periodic averaging [13]. Different from them, we do not assume a fixed quantization resolution given the variations of gradient value during the training process.

Federated learning under heterogeneous clients. Considering the heterogeneity of edge devices, FL under heterogeneous clients have also been studied in recent literature [14]-[18], [36]–[39]. Some of them consider the data heterogeneity across devices [14], [15], [37]. For example, Li et al. [14] proposed a subnetwork based approach that aims to improve inference accuracy by learning personalized models. Though the proposed framework also reduces the communication cost, the heterogeneous communication resources are not considered and hence the clients with poor communication conditions can still be the bottleneck. In [38], Wang et al. proposed an approach that identifies irrelevant updates of clients and precludes the uploading of these updates to save bandwidth. Considering the resource heterogeneity, the asynchronous aggregation strategy has been designed to address the straggler problem [17], [18], where the server aggregation does not have to wait for all clients. Although the asynchronous aggregation reduces the delay by stragglers, the delayed gradients of stragglers introduce errors or even diverge the learning of the model. Different from them, we propose heterogeneous gradient quantization to reduce the communication time of stragglers without compromising the model performance.

## VI. CONCLUSIONS

In this paper, we proposed AdaGQ, an adaptive and heterogeneous gradient quantization algorithm for communicationefficient federated learning for mobile edge devices. Based on varying gradient norm during training, we proposed an adaptive gradient quantization to seek the optimal quantization resolution in an online manner to minimize the total training time. We further designed heterogeneous gradient quantization to align the training time of slow clients in each round with others to mitigate the straggler effects. Evaluations based on various models and datasets validate the effectiveness of AdaGQ.

#### REFERENCES

- N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile Edge Computing: A Survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.
- [2] H. Liu and G. Cao, "Deep Learning Video Analytics Through Online Learning Based Edge Computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 10, pp. 8193–8204, 2022.
- [3] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.
- [4] H. Liu and G. Cao, "Deep Reinforcement Learning-Based Server Selection for Mobile Edge Computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13351–13363, 2021.
- [5] J. Konečný, 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.
- [6] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan et al., "Towards Federated Learning at Scale: System Design," in *Proceedings* of Machine Learning and Systems (MLSys), 2019.
- [7] S. Chen, C. Shen, L. Zhang, and Y. Tang, "Dynamic Aggregation for Heterogeneous Quantization in Federated Learning," *IEEE Transactions* on Wireless Communications, vol. 20, no. 10, pp. 6804–6819, 2021.
- [8] 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 (AISTATS)*, 2017.
- [9] F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. R. Cadambe, "Local SGD with Periodic Averaging: Tighter Analysis and Adaptive Synchronization," in Advances in Neural Information Processing Systems, 2019.
- [10] A. F. Aji and K. Heafield, "Sparse Communication for Distributed Gradient Descent," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.
- [11] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning," in Advances in Neural Information Processing Systems, 2017.
- [12] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding," in Advances in Neural Information Processing Systems, 2017.
- [13] 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 (AISTATS)*, 2020.
- [14] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: An Efficient Federated Learning Framework for Heterogeneous Mobile Clients," in *ACM MobiCom*, 2021.
- [15] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "A Novel Framework for the Analysis and Design of Heterogeneous Federated Learning," *IEEE Transactions on Signal Processing*, vol. 69, pp. 5234– 5249, 2021.
- [16] E. Diao, J. Ding, and V. Tarokh, "HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients," in *International Conference on Learning Representations (ICLR)*, 2020.
- [17] X. Zhao, A. An, J. Liu, and B. Chen, "Dynamic Stale Synchronous Parallel Distributed Training for Deep Learning," in *IEEE ICDCS*, 2019.
- [18] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang, and H. Huang, "FedSA: A Semi-Asynchronous Federated Learning Mechanism in Heterogeneous Edge Computing," *IEEE Journal on Selected Areas in Communications*, 2021.

- [19] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing Federated Learning on Non-IID Data with Reinforcement Learning," in *IEEE INFOCOM*, 2020.
- [20] G. Gur-Ari, D. A. Roberts, and E. Dyer, "Gradient Descent Happens in a Tiny Subspace," arXiv preprint arXiv:1812.04754, 2018.
- [21] S. Jastrzębski, Z. Kenton, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey, "On the Relation Between the Sharpest Directions of DNN Loss and the SGD Step Length," in *International Conference on Learning Representations (ICLR)*, 2018.
- [22] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical Secure Aggregation for Privacy-Preserving Machine Learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.
- [23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE CVPR*, 2016.
- [24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *IEEE CVPR*, 2015.
- [25] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," *Technical Report*, 2009.
- [26] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A Novel Image Dataset for Benchmarking Machine Learning Algorithms," arXiv preprint arXiv:1708.07747, 2017.
- [27] J. Wang and G. Joshi, "Adaptive Communication Strategies to Achieve the Best Error-Runtime Trade-off in Local-Update SGD," in *Proceedings* of Machine Learning and Systems (MLSys), 2019.
- [28] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient Sparsification for Communication-Efficient Distributed Optimization," in Advances in Neural Information Processing Systems, 2017.
- [29] T. Vogels, S. P. Karimireddy, and M. Jaggi, "PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization," in Advances in Neural Information Processing Systems, 2019.
- [30] P. Han, S. Wang, and K. K. Leung, "Adaptive Gradient Sparsification for Efficient Federated Learning: An Online Learning Approach," in *IEEE ICDCS*, 2020.
- [31] A. Albasyoni, M. Safaryan, L. Condat, and P. Richtárik, "Optimal Gradient Compression for Distributed and Federated Learning," arXiv preprint arXiv:2010.03246, 2020.
- [32] E. Ozfatura, K. Ozfatura, and D. Gündüz, "Time-Correlated Sparsification for Communication-Efficient Federated Learning," in *IEEE International Symposium on Information Theory*, 2021.
- [33] Y. Mao, Z. Zhao, G. Yan, Y. Liu, T. Lan, L. Song, and W. Ding, "Communication-Efficient Federated Learning with Adaptive Quantization," ACM Transactions on Intelligent Systems and Technology, vol. 13, no. 4, pp. 1–26, 2022.
- [34] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit Stochastic Gradient Descent and Its Application to Data-Parallel Distributed Training of Speech DNNs," in *Fifteenth annual conference of the international* speech communication association, 2014.
- [35] S. Li, Q. Qi, J. Wang, H. Sun, Y. Li, and F. R. Yu, "GGS: General Gradient Sparsification for Federated Learning in Edge Computing," in *IEEE ICC*, 2020.
- [36] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust Federated Learning in A Heterogeneous Environment," arXiv preprint arXiv:1906.06629, 2019.
- [37] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and Communication-Efficient Federated Learning from Non-i.i.d. Data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, 2019.
- [38] W. Luping, W. Wei, and L. Bo, "CMFL: Mitigating Communication Overhead for Federated Learning," in *IEEE ICDCS*, 2019.
- [39] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To Talk or to Work: Flexible Communication Compression for Energy Efficient Federated Learning over Heterogeneous Mobile Edge Devices," in *IEEE INFOCOM*, 2021.