Quantized Rank Reduction: A Communications-Efficient Federated Learning Scheme for Network-Critical Applications

Dimitrios Kritsiolis and Constantine Kotropoulos
Department of Informatics
Aristotle University of Thessaloniki
Thessaloniki 54124, Greece
email: {dkritsi, costas}@csd.auth.gr

***Abstract*—Federated learning is a machine learning approach that enables multiple devices (i.e., agents) to train a shared model cooperatively without exchanging raw data. This technique keeps data localized on user devices, ensuring privacy and security, while each agent trains the model on their own data and only shares model updates. The communication overhead is a significant challenge due to the frequent exchange of model updates between the agents and the central server. In this paper, we propose a communication-efficient federated learning scheme that utilizes low-rank approximation of neural network gradients and quantization to significantly reduce the network load of the decentralized learning process with minimal impact on the model's accuracy.**

Keywords-federated learning; Tucker decomposition; SVD; quantization.

I. INTRODUCTION

As artificial intelligence and machine learning evolve, new computational paradigms are emerging to address the increasing demand for privacy, efficiency, and scalability. One such approach is Federated Learning (FL), a decentralized learning technique that enables model training across multiple devices or agents without requiring direct data sharing [1] [2]. In FL, end devices train their model using local data and send model updates to the server for aggregation rather than sharing raw data. This approach enhances data privacy while allowing the server to refine the global model based on updates from multiple devices. FL is a key enabler of artificial intelligence in mobile devices and the Internet of Things (IoT) [3].

One of the key challenges in FL is the significant communications overhead, which does not scale efficiently as the number of participating devices increases [4]. The just-described issue becomes even more pronounced in deep learning, where models consist of voluminous parameters that must be shared by each client with the server at every training iteration. As a result, the communication bottleneck diminishes the advantage of distributed optimization, slowing the overall training process and reducing the efficiency gains expected from decentralized learning [5] [6]. To address this issue, we aim to compress and quantize the updates sent by clients, thereby mitigating the effects of communication overhead without significantly deteriorating the model's performance.

Before explaining the compression and quantization techniques, we formally introduce the distributed learning problem [7] solved by FL, i.e.,

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \sum_{c \in \mathfrak{C}} f_c(\boldsymbol{\theta}) \quad \text{with} \quad f_c(\boldsymbol{\theta}) := \sum_{n=1}^{N_c} J(\boldsymbol{X}_{c,n}; \boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{\theta}$ denotes the parameters of the central model being trained, $\mathfrak{C}$ is the set of clients participating in FL with $|\mathfrak{C}| = C$, $\boldsymbol{X}_{c,n}$ is the n-th data point of client c (which can be a feature matrix or generally a feature tensor), N_c is the total number of data points at client c, $J(\boldsymbol{X}_{c,n}, \boldsymbol{\theta})$ is the loss function used in the FL setting and $f_c(\boldsymbol{\theta})$ is the local loss associated with client c and its data. The overall loss function we optimize is $f(\boldsymbol{\theta})$.

Problem (1) is solved using gradient descent. The gradient descent update at iteration $k+1$ is given by

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \sum_{c \in \mathfrak{C}} \nabla f_c(\boldsymbol{\theta}^k), \tag{2}$$

where $\nabla f_c(\boldsymbol{\theta}^k)$ is the local gradient of client c associated with its data, and α is the learning rate. The sum term in (2) is a distributed version of gradient descent, also known as *Federated Averaging* [8]. Equation (2) implies that each client communicates its local gradient to the server at each training iteration. Depending on the quality of the network connection of each client, a significant overhead is introduced to the FL process. This overhead can surpass the computational cost of training a model for the client. To minimize the data transmission overhead on the distributed training process, we propose to compress the gradient of the loss function, which is reshaped to a matrix or tensor, into a more compact form utilizing a low-rank approximation [9] [10] [11] and then quantize the resulting compact form to reduce further the volume of the data to be transmitted at each iteration. The proposed novel scheme leverages the low-rank approximation of neural network gradients and established quantization algorithms.

The outline of the paper is as follows. Section II briefly describes the preliminaries, i.e., gradient compression and quantization. Section III details the proposed Quantized Rank Reduction (QRR) scheme and discusses the experimental results. Conclusions are drawn in Section IV. The code for QRR can be found at [12].

II. PRELIMINARIES

A. Gradient Compression

Neural network gradients are expressed in matrix or vector form [13]. Suppose we have a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ that maps a vector of length n to a vector of length m:

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(x_1, \ldots, x_n) \\ f_2(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{bmatrix}. \tag{3}$$

The partial derivatives of the vector function are stored in the Jacobian matrix $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$, with $\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}$:

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{4}$$

In the FL context, Jacobian matrices, such as (4), are computed by the clients using the backpropagation algorithm and sent back to the server. The server aggregates them to train the central model via gradient descent. For example, consider the weights of a fully connected layer $\boldsymbol{W} \in \mathbb{R}^{D_{out} \times D_{in}}$ and the bias term $\boldsymbol{b} \in \mathbb{R}^{D_{out} \times 1}$ along with the scalar loss function $J(\cdot)$ used by the neural network, where D_{out} is the size of the fully connected layer output and D_{in} is the size of the input to that layer. After training on its data, each client will derive a gradient reshaped as matrix $\frac{\partial J}{\partial \boldsymbol{W}} \in \mathbb{R}^{D_{out} \times D_{in}}$, as well as the gradient for the bias term $\frac{\partial J}{\partial \boldsymbol{b}} \in \mathbb{R}^{D_{out} \times 1}$. These gradients will be transmitted to the server to train the central model.

Transmitting the gradients to the server can be slow, especially when training a model with many parameters. The biggest communications overhead comes from $\frac{\partial J}{\partial \boldsymbol{W}}$ and not from $\frac{\partial J}{\partial \boldsymbol{b}}$. This is why we seek to compress $\frac{\partial J}{\partial \boldsymbol{W}}$ by applying the truncated Singular Value Decomposition (SVD), transmitting only the SVD components to the server, and reconstructing $\frac{\partial J}{\partial \boldsymbol{W}}$ on the server using the SVD components.

SVD is a matrix factorization technique that decomposes a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ into three matrices:

$$\boldsymbol{A} = \boldsymbol{U}\, \boldsymbol{\Sigma}\, \boldsymbol{V}^\top, \tag{5}$$

where $\boldsymbol{U}$ is an $m \times m$ orthonormal matrix containing the left singular vectors of $\boldsymbol{A}$ in its columns, $\boldsymbol{\Sigma}$ is an $m \times n$ matrix with the singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$, in descending order as its diagonal entries, for $r \leq \min(m, n)$ being the rank of matrix $\boldsymbol{A}$, and $\boldsymbol{V}$ is a $n \times n$ orthogonal matrix containing the right singular vectors of $\boldsymbol{A}$ in its columns. We can approximate the matrix $\boldsymbol{A}$ by keeping only the ν largest singular values:

$$\boldsymbol{A} \approx \boldsymbol{A}_\nu = \boldsymbol{U}_\nu\, \boldsymbol{\Sigma}_\nu\, \boldsymbol{V}_\nu^\top, \tag{6}$$

where $\boldsymbol{U}_\nu \in \mathbb{R}^{m \times \nu}$, $\boldsymbol{\Sigma}_\nu \in \mathbb{R}^{\nu \times \nu}$ and $\boldsymbol{V}_\nu \in \mathbb{R}^{n \times \nu}$ with $\nu < r$. The approximation error of $\boldsymbol{A}$ by $\boldsymbol{A}_\nu$ is given by

$$||\boldsymbol{A} - \boldsymbol{A}_\nu||_F^2 = \sum_{j=\nu+1}^{r} \sigma_j^2, \tag{7}$$

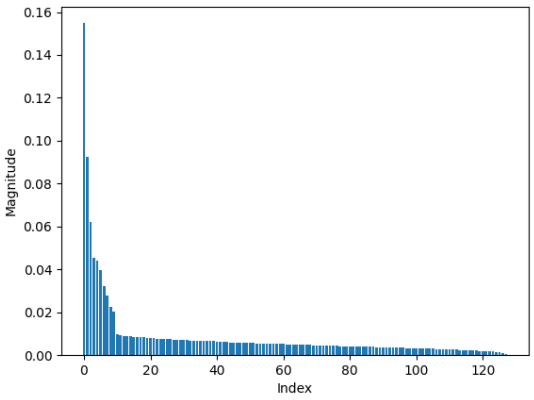


Figure 1. Magnitude of the singular values of the gradient of a fully connected layer.

where $||\cdot||_F$ denotes the Frobenius norm and σ_j, $j > \nu$ are the truncated singular values.

The approximation of $\frac{\partial J}{\partial \boldsymbol{W}} \in \mathbb{R}^{D_{out} \times D_{in}}$ with a truncated SVD is justified because such matrices are generally low-rank and have a few dominant singular values [14]. This was experimentally verified by plotting the magnitudes of the singular values of a fully connected layer's gradient in Figure 1, where only a few of the 128 singular values are significantly larger than 0.

Suppose we only transmit $\boldsymbol{U}_\nu$, $\boldsymbol{V}_\nu$, and the diagonal entries of $\boldsymbol{\Sigma}_\nu$. For the truncated SVD to be more communication-efficient than transmitting the full matrix $\frac{\partial J}{\partial \boldsymbol{W}}$, the following inequality must hold:

$$D_{out} \cdot \nu + \nu + D_{in} \cdot \nu < D_{out} \cdot D_{in}. \tag{8}$$

Factorization can also be applied to convolutional layers. In a convolutional layer, the weights are represented by a 4-dimensional tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{C_{out} \times C_{in} \times H \times W}$, where C_{out} is the number of output channels, C_{in} is the number of input channels and $H \times W$ is the size of the convolutional filter. The bias terms are represented as a vector $\boldsymbol{b} \in \mathbb{R}^{C_{out} \times 1}$. To reduce the communications overhead of transmitting the gradient of a convolutional layer $\frac{\partial J}{\partial \boldsymbol{\mathcal{W}}}$ reshaped to a tensor, we factorize the tensor using the Tucker decomposition [15], which has been used for factorization and compression of neural networks [16] [17].

The Tucker decomposition is a higher-order generalization of SVD. It factorizes a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ into a core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times r_2 \times \ldots \times r_N}$ and a set of factor matrices $\boldsymbol{F}_i \in \mathbb{R}^{I_i \times r_i}$, $i = 1, \ldots, N$, where $r_i < I_i$ are the reduced ranks on each mode. $\boldsymbol{\mathcal{X}}$ is approximated as [18]:

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{F}_1 \times_2 \boldsymbol{F}_2 \times_3 \ldots \times_N \boldsymbol{F}_N, \tag{9}$$

where $\times_n$ denotes the mode-n product of a tensor and matrix. Given a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ and a matrix $\boldsymbol{F} \in \mathbb{R}^{J \times I_n}$ the mode-n product of $\boldsymbol{\mathcal{X}}$ with $\boldsymbol{F}$ is denoted as $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_n \boldsymbol{F}$, where $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \ldots \times I_{n-1} \times J \times I_{n+1} \ldots \times I_N}$ has elements:

$$\boldsymbol{\mathcal{Y}}_{i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N} = \sum_{i_n=1}^{I_n} \boldsymbol{\mathcal{X}}_{i_1, \ldots, i_N} \cdot \boldsymbol{F}_{j, i_n}. \tag{10}$$

When transmitting the gradient of a convolutional layer reshaped to a tensor, $\frac{\partial J}{\partial \mathcal{W}} \in \mathbb{R}^{C_{out} \times C_{in} \times H \times W}$, with reduced ranks for each mode r_1, r_2, r_3, and r_4, we only transmit the core tensor and factor matrices. For the Tucker decomposition to be more communication-efficient, the following inequality must be true:

$$r_1 \cdot r_2 \cdot r_3 \cdot r_4 + C_{out} \cdot r_1 + C_{in} \cdot r_2 + H \cdot r_3 + W \cdot r_4 < C_{out} \cdot C_{in} \cdot H \cdot W. \quad (11)$$

B. Gradient Quantization

To further reduce the communication overhead of the FL setup, in addition to compressing the updates sent by the clients to the server, we also quantize them. Quantizing the gradients of each client leads to a modified version of (2) called Quantized Gradient Descent [19]:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \sum_{c \in \mathfrak{C}} Q\left(\nabla f_c(\boldsymbol{\theta}^k)\right), \quad (12)$$

where $Q\left(\nabla f_c(\boldsymbol{\theta}^k)\right)$ is the quantized gradient update of client c. Methods employing differential quantization of the gradients have also been proposed [20] [21].

The quantization scheme we use resorts to the Lazily Aggregated Quantized (LAQ) algorithm [22]. Specifically, in LAQ, the gradient descent update is given by

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha\, \nabla^k, \text{ with } \nabla^k = \nabla^{k-1} + \sum_{c \in \mathfrak{C}} \delta Q_c^k, \quad (13)$$

where ∇^k is the aggregated quantized gradient updates at iteration k, and $\delta Q_c^k := Q_c(\boldsymbol{\theta}^k) - Q_c(\boldsymbol{\theta}^{k-1})$ is the difference of the quantized gradient updates of client c at iterations k and $k-1$. The quantized gradient update of client c at iteration k is $Q_c(\boldsymbol{\theta}^k)$, and it is computed using the current gradient update $\nabla f_c(\boldsymbol{\theta}^k)$ and the previous quantized update $Q_c(\boldsymbol{\theta}^{k-1})$:

$$Q_c(\boldsymbol{\theta}^k) = \mathbb{Q}\left(\nabla f_c(\boldsymbol{\theta}^k),\, Q_c(\boldsymbol{\theta}^{k-1})\right), \quad (14)$$

where $\mathbb{Q}$ denotes the quantization operator. The operator $\mathbb{Q}$ entails the following quantization scheme.

The gradient update $\nabla f_c(\boldsymbol{\theta}^k)$ is quantized by projecting each element on an evenly-spaced grid. This grid is centered at $Q_c(\boldsymbol{\theta}^{k-1})$ and has a radius of $R_c^k = ||\nabla f_c(\boldsymbol{\theta}^k) - Q_c(\boldsymbol{\theta}^{k-1})||_\infty$, where $||\boldsymbol{x}||_\infty = \max(|x_1|, \ldots, |x_n|)$ is the max norm. The i-th element of the quantized gradient update of client c at iteration k is mapped to an integer as follows

$$[q_c(\boldsymbol{\theta}^k)]_i = \left\lfloor \frac{[\nabla f_c(\boldsymbol{\theta}^k)]_i - [Q_c(\boldsymbol{\theta}^{k-1})]_i + R_c^k}{2\tau R_c^k} + \frac{1}{2} \right\rfloor, \quad (15)$$

with $\tau := 1/(2^\beta - 1)$ defining the discretization interval. All $[q_c(\boldsymbol{\theta}^k)]_i$ are integers in the range $\{0, 1, \ldots, 2^\beta - 1\}$ and therefore can be encoded by using only β bits. The difference δQ_c^k is computed as

$$\delta Q_c^k = Q_c(\boldsymbol{\theta}^k) - Q_c(\boldsymbol{\theta}^{k-1}) = 2\tau R_c^k\, Q_c(\boldsymbol{\theta}^k) - R_c^k \mathbf{1}, \quad (16)$$

where $\mathbf{1} = [1 \ldots 1]^\top$. This quantity can be transmitted with $32 + \beta n$ bits instead of $32n$ bits. That is, 32 bits for R_c^k and β bits for each of the n elements of the gradient update. The computation requires each client to retain a local copy of $Q_c(\boldsymbol{\theta}^{k-1})$. The server can recover the gradient update of client c, assuming it knows the number of bits used for quantization, β, as

$$Q_c(\boldsymbol{\theta}^k) = Q_c(\boldsymbol{\theta}^{k-1}) + \delta Q_c^k. \quad (17)$$

The discretization interval is $2\tau R_c^k$. Therefore, the quantization error cannot be larger than half of the interval

$$||\nabla f_c(\boldsymbol{\theta}^k) - Q_c(\boldsymbol{\theta}^k)||_\infty \leq \tau\, R_c^k. \quad (18)$$

III. PROPOSED SCHEME

A. Quantized Rank Reduction

By combining compression and quantization, we propose a new scheme for communication-efficient FL, namely the QRR. The gradient descent step (2) becomes

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \sum_{c \in \mathfrak{C}} QRR_c\left(\boldsymbol{\theta}^k\right),$$
$$QRR_c\left(\boldsymbol{\theta}^k\right) = \mathbb{C}^{-1}\left(\mathbb{Q}\left(\mathbb{C}\left(\nabla f_c(\boldsymbol{\theta}^k)\right), \mathbb{C}\left(\nabla f_c(\boldsymbol{\theta}^{k-1})\right)\right)\right), \quad (19)$$

where $\mathbb{Q}$ is the quantization operator, $\mathbb{C}$ is the compression operator, and $\mathbb{C}^{-1}$ is the decompression operator. Each client applies the operators $\mathbb{C}$ and $\mathbb{Q}$ to compress and quantize its gradient update, while the server receives the updates and applies $\mathbb{C}^{-1}$ to decompress them and perform gradient descent.

$\mathbb{C}$ entails compressing the gradients using SVD or Tucker decomposition. For the gradient of a fully connected layer of client c at iteration k reshaped to a matrix $\frac{\partial J}{\partial \boldsymbol{W}_c^k} \in \mathbb{R}^{D_{out} \times D_{in}}$ we use a truncated SVD for compression

$$\frac{\partial J}{\partial \boldsymbol{W}_c^k} \approx \boldsymbol{U}_c^k\, \boldsymbol{\Sigma}_c^k\, (\boldsymbol{V}_c^k)^\top, \quad (20)$$

where $\boldsymbol{U}_c^k$, $\boldsymbol{\Sigma}_c^k$ and $\boldsymbol{V}_c^k$ are the SVD components of $\frac{\partial J}{\partial \boldsymbol{W}_c^k}$ retaining only the ν largest singular values.

In case the gradient update is a tensor, such as the gradient of a convolutional layer $\frac{\partial J}{\partial \mathcal{W}_c^k} \in \mathbb{R}^{C_{out} \times C_{in} \times H \times W}$, we compress it using the Tucker decomposition

$$\frac{\partial J}{\partial \mathcal{W}_c^k} \approx \mathcal{G}_c^k \times_1 (\boldsymbol{F}_1)_c^k \times_2 (\boldsymbol{F}_2)_c^k \times_3 (\boldsymbol{F}_3)_c^k \times_4 (\boldsymbol{F}_4)_c^k. \quad (21)$$

The compression is controlled by the parameter p, which represents the percentage of the original rank that is retained. For SVD, the new reduced rank is computed as

$$\nu = \lceil p \cdot \min(D_{out}, D_{in}) \rceil. \quad (22)$$

In the case of the Tucker decomposition, the reduced ranks of the core tensor are computed as

$$r_1 = \lceil p \cdot C_{out} \rceil, \quad r_2 = \lceil p \cdot C_{in} \rceil, \quad r_3 = \lceil p \cdot H \rceil, \quad r_4 = \lceil p \cdot W \rceil. \quad (23)$$

For inequalities (8) and (11) to hold, we typically want p to be small, i.e., $p < 0.5$.

The gradients of the bias terms $\frac{\partial J}{\partial \boldsymbol{b}_c^k} \in \mathbb{R}^{D_{out} \times 1}$ are quantized only without compression.

The operator $\mathbb{Q}$ is described in Section II-B. Each component resulting from the factorization of the gradient update using either SVD or Tucker decomposition is quantized according to this scheme. Client c must store the previous quantized components of its gradient update locally. For each matrix $\frac{\partial J}{\partial \boldsymbol{W}_c^k}$ it has to store $Q(\boldsymbol{U}_c^{k-1})$, $Q(\boldsymbol{\Sigma}_c^{k-1})$ and $Q(\boldsymbol{V}_c^{k-1})$. For each gradient tensor $\frac{\partial J}{\partial \boldsymbol{\mathcal{W}}_c^k}$, it has to store $Q(\mathcal{G}_c^{k-1})$ and $Q((\boldsymbol{F}_1)_c^{k-1}), \ldots, Q((\boldsymbol{F}_4)_c^{k-1})$. For each bias gradient vector $\frac{\partial J}{\partial \boldsymbol{b}_c^k}$ the previous quantized vector $Q(\frac{\partial J}{\partial \boldsymbol{b}_c^{k-1}})$ must also be stored. The parameter β is the number of bits used to encode each element and controls the quantization accuracy.

The server receives each client's gradient updates and computes the current iteration's quantized factor components according to (17). Equation (17) requires that the server also store each client's previously quantized factors. Once the server has the current quantized factors, it applies the operator $\mathbb{C}^{-1}$ to reconstruct the gradient updates of each client. That is, for each client c and each model parameter P in the clients' gradient updates,

- if $P = \boldsymbol{W}_c^k \in \mathbb{R}^{D_{out} \times D_{in}}$:

$$\frac{\partial J}{\partial \boldsymbol{W}_c^k} \approx Q(\boldsymbol{U}_c^k)\, Q(\boldsymbol{\Sigma}_c^k)\, Q(\boldsymbol{V}_c^k)^\top, \tag{24}$$

- if $P = \boldsymbol{\mathcal{W}}_c^k \in \mathbb{R}^{C_{out} \times C_{in} \times H \times W}$:

$$\begin{aligned}\frac{\partial J}{\partial \boldsymbol{\mathcal{W}}_c^k} \approx Q(\mathcal{G}_c^k) \times_1 Q((\boldsymbol{F}_1)_c^k) \times_2 Q((\boldsymbol{F}_2)_c^k) \\ \times_3 Q((\boldsymbol{F}_3)_c^k) \times_4 Q((\boldsymbol{F}_4)_c^k),\end{aligned} \tag{25}$$

- if $P = \boldsymbol{b}_c^k \in \mathbb{R}^{D_{out} \times 1}$:

$$\frac{\partial J}{\partial \boldsymbol{b}_c^k} \approx Q(\frac{\partial J}{\partial \boldsymbol{b}_c^k}). \tag{26}$$

The server then uses the gradient approximations to perform the distributed gradient descent.

B. Experimental Results

Experiments were conducted to compare the performance of the proposed QRR with stochastic federated averaging, referred to as Stochastic Gradient Descent (SGD), and with the Stochastic LAQ (SLAQ) [22]. To measure the performance of each method, we kept track of the loss and accuracy of the model, as well as the number of bits transmitted by the clients during each iteration. Since the SLAQ algorithm skips uploading the gradient update of some clients based on their magnitude, we also recorded the number of communications. Next, we clarify the terms used in the experiments:

- By **iteration**, we mean a full round of FL, which consists of the server passing the central model's weights to the clients, the clients computing their local mean gradient over a single batch and sending it to the server, and the server aggregating the clients' gradients and updating the central model.
- By **communication**, we refer to the data exchange from the client to the server, i.e., when the client sends its local gradient update to the server.
- By **bits**, we measure only the number of bits of the gradient updates transferred from the clients to the server, since the bits required to transmit the model weights from the server to all the clients are constant and common across all methods.

All the experiments used 10 clients and quantized the compressed gradient updates using $\beta = 8$ bits. The learning rate was $\alpha = 0.001$, and the batch size was equal to 512. For the SLAQ algorithm, the parameters used were $D = 10$, $\xi_1, \ldots, \xi_D = 1/D$, and 8 bits for quantization.

The first experiment utilized the MNIST dataset [23] of 28×28 grayscale images of handwritten digits. A simple Multi-Layer Perceptron (MLP) network was employed, comprising a hidden layer with 200 neurons, a Rectified Linear Unit (ReLU) activation function, and input and output layers of size 784 (28×28) and 10, respectively, with a cross-entropy loss function. 60,000 training samples were randomly selected and equally distributed among the 10 clients. A total of 10,000 test samples were used to evaluate the performance of the central model. The results for 1000 iterations are presented in Table I for various values of p in QRR.

QRR achieves an accuracy of around 1-2% lower than SGD and SLAQ. However, it transmits 3.16-9.43% of the bits transmitted by SGD and 14.8-44.05% of the bits transmitted by SLAQ, depending on the choice of the parameter p. In Figure 2, the loss, the gradient ℓ_2 norm, and the accuracy are plotted against each method's number of iterations and bits. QRR has a slower convergence rate with respect to (wrt) the iteration number than SGD and SLAQ. The smaller p is, the slower the loss convergence, as evidenced in Figure 2(a) since we have less accurate reconstructions of the gradients with smaller p values. However, performance wrt the number of bits transmitted is better, as seen in Figures 2(b), 2(d), and 2(f), i.e., a smaller loss, a smaller gradient ℓ_2 norm, and higher accuracy are measured for a fixed number of bits.

The second experiment used the same setup as the first, with the difference that the MLP network was replaced by a Convolutional Neural Network (CNN). The CNN consisted of 2 convolutional layers using 3×3 filters with 16 and 32 output channels, respectively, a max pooling layer that reduced the input size by half, and 1 fully connected layer. The activation function used after each layer was the ReLU function, and the loss function used was the cross-entropy loss.

Table II summarizes the results using the CNN. Figure 3 displays the evolution of the loss, gradient ℓ_2 norm, and accuracy wrt the number of iterations and bits. The curves for loss and accuracy versus iterations or bits are similar to those of the first experiment. QRR scores 1-3% lower in accuracy but requires 2.75-7.84% of the bits of SGD and 13.52-38.52% of the bits of SLAQ, depending on the choice of p.

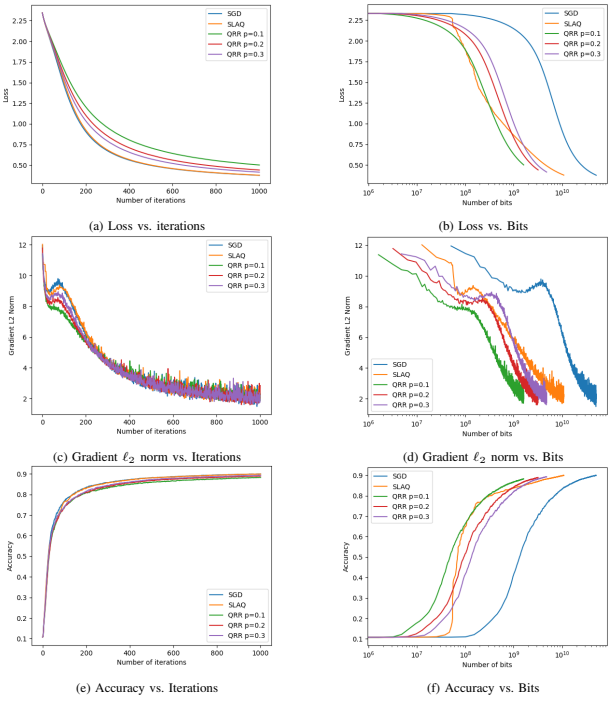
In the third experiment, the CIFAR-10 dataset [24] was used with a small, VGG-like [25] CNN consisting of three convolutional blocks with 3×3 convolutions, ReLU activations,

TABLE I. RESULTS OF QRR COMPARED TO SLAQ and SGD FOR AN MLP APPLIED TO THE MNIST DATASET.

Algorithm	# Iterations	# Bits	# Communications	Loss	Accuracy	Gradient ℓ_2 norm
SGD	1000	5.088×10^{10}	10000	0.376	89.92%	2.297
SLAQ	1000	1.089×10^{10}	8559	0.378	89.89%	2.026
QRR($p = 0.3$)	1000	4.798×10^{9}	10000	0.415	89.20%	1.945
QRR($p = 0.2$)	1000	3.205×10^{9}	10000	0.441	88.93%	2.846
QRR($p = 0.1$)	1000	1.612×10^{9}	10000	0.501	88.22%	1.866

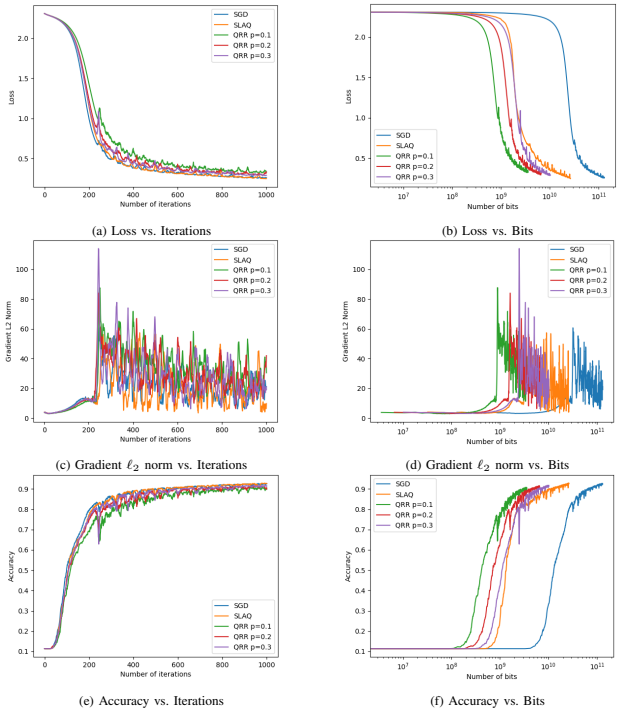
TABLE II. RESULTS OF QRR COMPARED TO SLAQ and SGD FOR A CNN APPLIED TO THE MNIST DATASET.

Algorithm	# Iterations	# Bits	# Communications	Loss	Accuracy	Gradient ℓ_2 norm
SGD	1000	1.302×10^{11}	10000	0.263	92.56%	21.154
SLAQ	1000	2.653×10^{10}	8151	0.251	92.70%	9.769
QRR($p = 0.3$)	1000	1.022×10^{10}	10000	0.291	91.49%	19.287
QRR($p = 0.2$)	1000	6.650×10^{9}	10000	0.335	89.91%	42.026
QRR($p = 0.1$)	1000	3.588×10^{9}	10000	0.330	90.48%	30.455



(a) Loss vs. iterations (b) Loss vs. Bits (c) Gradient ℓ_2 norm vs. Iterations (d) Gradient ℓ_2 norm vs. Bits (e) Accuracy vs. Iterations (f) Accuracy vs. Bits

Figure 2. Loss, gradient ℓ_2 norm, and accuracy plotted against the number of iterations and bits for the MLP network and the MNIST dataset.



(a) Loss vs. Iterations (b) Loss vs. Bits (c) Gradient ℓ_2 norm vs. Iterations (d) Gradient ℓ_2 norm vs. Bits (e) Accuracy vs. Iterations (f) Accuracy vs. Bits

Figure 3. Loss, gradient ℓ_2 norm, and accuracy plotted against the number of iterations and bits for the CNN and the MNIST dataset.

max pooling, and dropout layers, with the number of filters increasing from 32 to 64 and then to 128. We used different values of p to demonstrate that p can be chosen based on the client's connection speed and the amount of data transmitted from that client. Evenly spaced values in $[0.1, 0.3]$ were assigned to the p parameter of each client. The experiment ran for 2000 iterations, using a learning rate of 0.01 for the first 1000 iterations to accelerate convergence, and then 0.001 for the remaining iterations to ensure stable training.

Table III shows that QRR achieves 8–9% lower accuracy than SGD and SLAQ, while transmitting only 3.34% and 15.26% of the bits transmitted by SGD and SLAQ, respectively. Figure 4 plots the loss, gradient ℓ_2 norm, and accuracy versus iterations or transmitted bits for the VGG-like CNN on CIFAR-10. Although the low-rank approximation of the gradients leads to reduced accuracy on this dataset, which is more complex than MNIST, QRR remains useful for quickly reaching a deployable model state before switching to a more accurate one, compared to less network-efficient methods such as SGD or SLAQ.

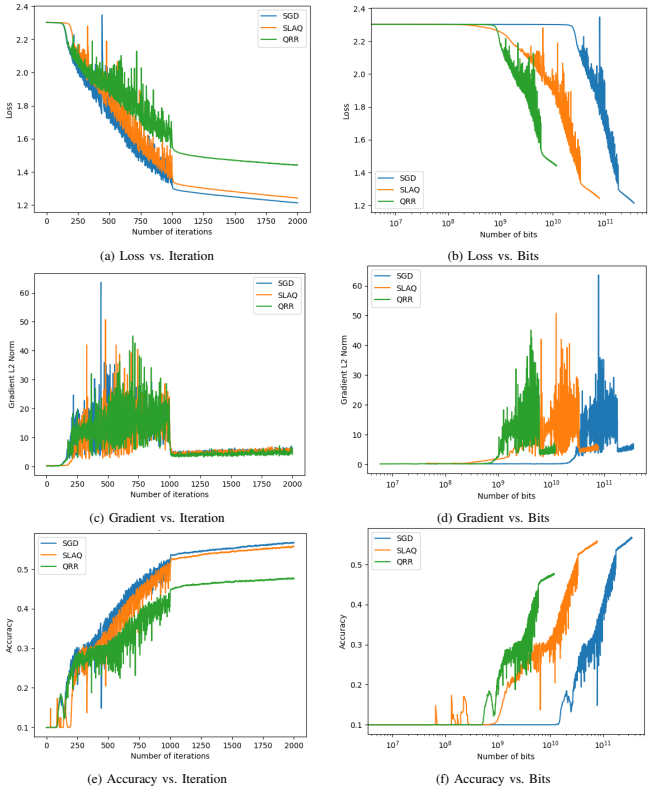
Finally, the client-side overhead of QRR was measured in the setup of the last experiment using SGD as a baseline. On average, QRR needed 1.2× more memory and 3.82× more computation time. For comparison, SLAQ required 13× more memory and 1.08× more computation time.

IV. Conclusions

We have proposed a scheme that leverages the low-rank approximation of neural network gradients and utilizes established quantization algorithms to significantly reduce the amount of data transmitted in an FL setting. The proposed Quantized Rank Reduction scheme has slightly lower accuracy than Federated Averaging or SLAQ, but it transmits only

TABLE III. RESULTS OF QRR COMPARED TO SLAQ and SGD FOR A VGG-LIKE CNN APPLIED TO THE CIFAR-10 DATASET.

Algorithm	# Iterations	# Bits	# Communications	Loss	Accuracy	Gradient ℓ_2 norm
SGD	2000	3.52×10^{11}	20000	1.213	56.72%	6.246
SLAQ	2000	7.72×10^{10}	17548	1.242	55.73%	5.493
QRR	2000	1.17×10^{10}	20000	1.441	47.57%	5.088



(a) Loss vs. Iteration (b) Loss vs. Bits

(c) Gradient vs. Iteration (d) Gradient vs. Bits

(e) Accuracy vs. Iteration (f) Accuracy vs. Bits

Figure 4. Loss, gradient ℓ_2 norm, and accuracy plotted against the number of iterations and bits for the VGG-like CNN and the CIFAR-10 dataset.

a fraction of the bits required by the other methods. It converges more slowly with the number of iterations, but faster when considering the number of bits transmitted. There is an added computational and memory overhead on both the client and server sides. However, this scheme can prove helpful in network-critical applications, where sensors or devices participating in the distributed learning process are located in remote locations with very slow network connections.

REFERENCES

[1] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2017. [Online]. Available: https://arxiv.org/abs/1610.05492

[2] C. Zhang *et al.*, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[3] W. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[4] M. Asad *et al.*, "Limitations and future aspects of communication costs in federated learning: A survey," *Sensors*, vol. 23, no. 17, p. 7358, 2023.

[5] P. Kairouz *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1-2, pp. 1–210, 2021.

[6] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, 2014, vol. 27, pp. 19—-27.

[7] Y. Arjevani and O. Shamir, "Communication complexity of distributed convex learning and optimization," in *Advances in Neural Information Processing Systems*, 2015, vol. 28, pp. 1756—1764.

[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*. PMLR, 2017, pp. 1273–1282.

[9] H. Kim, M. U. K. Khan, and C.-M. Kyung, "Efficient neural network compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 569–12 577.

[10] L. Liu and X. Xu, "Marvel: Towards efficient federated learning on IoT devices," *Computer Networks*, vol. 245, p. 110375, 2024.

[11] Y. Liu and M. K. Ng, "Deep neural network compression by Tucker decomposition with nonlinear response," *Knowledge-Based Systems*, vol. 241, p. 108171, 2022.

[12] "Quantized rank reduction: A communications-efficient federated learning scheme for network-critical applications," [retrieved: May 22, 2025]. [Online]. Available: https://github.com/Kritsos/QRR-code

[13] K. Clark, "Computing neural network gradients," Stanford University, August 2018, Notes.

[14] S. Oymak, Z. Fabian, M. Li, and M. Soltanolkotabi, "Generalization guarantees for neural networks via harnessing the low-rank structure of the Jacobian," 2019. [Online]. Available: https://arxiv.org/abs/1906.05392

[15] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[16] G. G. Calvi, A. Moniri, M. Mahfouz, Q. Zhao, and D. P. Mandic, "Compression and interpretability of deep neural networks via Tucker tensor layer: From first principles to tensor valued back-propagation," 2019. [Online]. Available: https://arxiv.org/abs/1903.06133

[17] J.-T. Chien and Y.-T. Bao, "Tensor-factorized neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 5, pp. 1998–2011, 2017.

[18] L. De Lathauwer, *Signal Processing Based on Multilinear Algebra*. Katholieke Universiteit Leuven, 1997.

[19] 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, vol. 30, pp. 1707—1718.

[20] K. Mishchenko, E. Gorbunov, M. Takáč, and P. Richtárik, "Distributed learning with compressed gradient differences," 2023. [Online]. Available: https://arxiv.org/abs/1901.09269

[21] N. Tonellotto, A. Gotta, F. M. Nardini, D. Gadler, and F. Silvestri, "Neural network quantization in federated learning at the edge," *Information Sciences*, vol. 575, pp. 417–436, 2021.

[22] J. Sun, T. Chen, G. B. Giannakis, Q. Yang, and Z. Yang, "Lazily aggregated quantized gradient innovation for communication-efficient federated learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 4, pp. 2031–2044, 2020.

[23] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[24] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556