Neural Network Quantization in Federated Learning at the Edge

Nicola Tonellotto[b], Alberto Gotta[a], Franco Maria Nardini[a], Daniele Gadler[d], Fabrizio Silvestri[c]

[a] *ISTI-CNR, Pisa, Italy*
[b] *University of Pisa, Pisa, Italy*
[c] *"Sapienza", University of Rome*
[d] *ONE LOGIC GmbH, Frankfurt, Germany*

Abstract

The massive amount of data collected in the Internet of Things (IoT) asks for effective, intelligent analytics. A recent trend supporting the use of Artificial Intelligence (AI) solutions in IoT domains is to move the computation closer to the data, i.e., from cloud-based services to edge devices. Federated learning (FL) is the primary approach adopted in this scenario to train AI-based solutions. In this work, we investigate the introduction of quantization techniques in FL to improve the efficiency of data exchange between edge servers and a cloud node. We focus on learning recurrent neural network models fed by edge data producers using the most widely adopted neural networks for time-series prediction. Experiments on public datasets show that the proposed quantization techniques in FL reduces up to 19× the volume of data exchanged between each edge server and a cloud node, with a minimal impact of around 5% on the test loss of the final model.

Keywords: Federated Learning, Artificial Neural Networks, Quantization, Internet of Things

1. Introduction

Internet of Things (IoT) is a disruptive technology that is pervasively extending the concept of data collection to everything around us through IoT devices, thus leading to a huge growth of the Internet traffic. Cisco Annual Internet Report forecasts that "*the*

Email addresses: `nicola.tonellotto@unipi.it` (Nicola Tonellotto), `alberto.gotta@isti.cnr.it` (Alberto Gotta), `francomaria.nardini@isti.cnr.it` (Franco Maria Nardini), `daniele.gadler@yahoo.it` (Daniele Gadler), `fabrizio.silvestri@uniroma1.it` (Fabrizio Silvestri)

share of Machine-To-Machine (M2M) connections will grow from 33 percent in 2018 to 50 percent by 2023. There will be 14.7 billion M2M connections by 2023". The huge amount of collected data enables the adoption of Artificial intelligence (AI) analytics currently provided by centralized cloud-based services. However, cloud-based solutions may raise major concerns among users about the privacy of online services. A recent approach to mitigate this issue is to decentralize the computation where data is, i.e., on personal IoT devices connected to the Internet. Edge computing [1] extends Cloud computing, according to the above decentralized approach: data processing and storage capabilities are not exclusive characteristics of centralized data centers, but an additional layer, called *edge*, is placed in the middle between the Cloud and the IoT devices. This edge layer allows for storing data and executing applications on edge servers directly connected with IoT devices. Moreover, edge computing allows to preserve the confidentiality of private user data, which are outsourced to edge servers for storage and computational processes. A privacy perimeter is defined in the edge - hereinafter called *privacy domain* - wherein access control, authentication, encryption, and secure computation are performed. In this scenario, the recent introduction of *artificial-intelligence-as-a-service* [2] tackles significant innovations across all the industrial sectors in particular in the *artificial intelligence of things* (AIoT) [3, 4], where the data required to train AI solutions are kept local to the device without disclosing private or sensitive data.

In medical, industrial, and social IoT scenarios, AIoT solutions must analyze very large and dynamic time-series. Traditional time-series forecasting techniques, such as an autoregressive integrated moving average, rely on highly manually-tuned parameters [5] and are ill-equipped to learn long-range dependencies [4]. Recurrent Neural Networks (RNNs) have been successfully applied to tasks based on time-series data, such as stock forecasting, social behavior analysis, and natural language processing (NLP). Federated learning (FL) is a leading approach for training RNNs adopted in AIoT solutions [6]. In FL, the computation involved in the training of an AI solution, e.g., a Neural Network (NN), is moved closer to where data are produced. FL can naturally apply to the IoT-Edge-Cloud scenario. This scenario may require preserving data privacy and data ownership. In an FL scenario applied to edge computing: i) every edge server receives a partially trained NN model from a cloud node, ii) every edge server performs additional training on data provided by the respective IoT devices to refine the previous model without disclosing any private data, iii) after the local training ends, the refined local model is sent back to the cloud node, and iv) the cloud node collects all the locally-trained models, generates a new global model, and broadcasts the global model back to the edge servers for a new round of local training. By doing so, the complex fusion of Machine Learning (ML) models on a cloud node is decoupled from the storage of training data on edge servers to preserve user data ownership and privacy. Yet, IoT devices are relieved of computationally intensive tasks, like AI services, allowing them to either publish data or execute a limited number of tasks. Learning state-of-the-art NN models in an FL scenario is challenging because of the models' size that range from hundreds of MBs to several GBs [7]. Hence, transferring such models from a cloud node to edge servers and vice-versa would lead to prolonged data exchange, high data transfer costs, and energy drain. In this work, we investigate the introduction of quantization techniques in the FL scenario to improve the efficiency of data exchange between edge servers and a cloud node. We focus on learning accurate NN models in an FL scenario with the model training steps performed on edge servers and the model aggregation performed on cloud nodes.

In detail, the novel and unpublished contributions of this article are the following:

- we propose the application of quantization techniques to the FL scenario by defining of the *Federated Learning with Quantization* (FLQ);
- we discuss how NN quantization techniques could apply to the FLQ scenario by introducing two new FL algorithms, namely FLQ and ΔFLQ.
- we provide a comprehensive analysis of the performance of our FLQ and ΔFLQ algorithms with different quantization techniques in terms of i) the effectiveness of the learned NN model and ii) the data reduction attained on the public WikiText-2 text dataset;
- we further assess the effectiveness and data efficiency performance of our FLQ and ΔFLQ algorithms with the best quantization techniques on different public datasets and NN models, namely the MNIST image dataset with a Convolutional Neural Network (CNN) applied to the task of image classification, and the BAR CRAWL sensor dataset with an RNN applied to a regression task;

- we discuss the performance of the proposed ΔFLQ algorithm with up to 10 edge servers and evaluate our algorithm's performance in the presence of faults.

Our experimental results show that the application of quantization techniques to FL allows us to significantly reduce the total data exchanged between each edge server and a cloud node (up to 19×) with a minimal impact on the test loss of the final NN model (around 5%).

The remainder of the article is organized as follows. After an overview of the current state of the art in Section 2, we outline the NN quantization schemes and the proposed FLQ and ΔFLQ algorithms in Section 3. We provide a thorough explanation of the experimental evaluation of our approach in Section 4, which includes experiments on multiple edge servers (up to 10) and a discussion of the robustness of our best approach in the presence of faults. Then, we discuss the comprehensive experimental evaluation of the quantization schemes and our FLQ and ΔFLQ algorithms in Section 5. Section 5.4 reports additional results on training networks for different datasets, containing namely image and sensor data. Finally, Section 7 draws the main conclusions and discusses future work.

2. Related Work

AI solutions and, in particular, NNs have been recently investigated in different IoT scenarios. Recently, Shanthamallu *et al.* survey AI methods and their applications to the IoT world [3], while Mohammadi *et al.* describe state-of-the-art methods in AI for IoT, big data, and streaming analytics [4]. Li *et al.* introduce a layered AI approach running on edge IoT devices [8], and Tang *et al.* describe methods to enable AI in IoT devices [9]. More recently, Lu *et al.* propose to apply federated learning in industrial IoT [10]. NNs are the current state-of-the-art model for such AIoT scenarios. Deep NNs are made up of multiple hidden layers of neurons connected through weighting matrices trained to accurately approximate a given objective function. Among them, convolutional neural networks (CNNs) are used to process image-based data, while RNNs are used to process variable-length sequences. RNNs are suitable for tasks involving time-series and time-segmented tasks. The Long-Short-Term-Memory (LSTM) is an RNN with long short-term memory blocks that consist of memory cell units. These memory cell units let the LSTM remember the state values for an arbitrarily long time sequence. LSTM networks have been successfully applied to applications with sequential data such as time series prediction [11], NLP [12], and social behavior analysis [13].

Deep NNs are over-parameterized to ease the training process. For example, AlexNet [14] has 60M parameters to be learned. NN pruning [15] consists of removing weights in a neural network to reduce the storage requirements of the network parameters. NN pruning is orthogonal to our proposed approach since every edge server can deploy its pruning solution. NN quantization is another approach proposed to efficiently train NNs. This approach constrains the precision of floating-point 32-bit weights, activation values, and/or gradients used in the training procedure to a fixed-point representation, using $k < 32$ bits. Note that NN quantization only affects the massive computations performed to calculate the updates of the matrix weights. In contrast, the matrix weights are maintained at full precision during the training procedure. In this line of research, DoReFa-Net quantizes the weights of a CNN to 1 bit, activations to 2 bits, and gradients to 6 bits to preserve a high model accuracy [16]. Different quantization approaches for CNNs, i.e., one-bit [17] and multi-bit quantization [18] for limited-precision training, have also been proposed. State-of-the-art methods for uniform quantization are XNOR-Net [19] and binary weight nets [20], which propose a binary quantization mapping weights to -1 and $+1$ and replace operations on the weights with more efficient bit-wise operations. More recently, ternary weight nets [21] have also been introduced, allowing weights to be zero. Zhou *et al.* propose incremental NN quantization [22]: this approach iteratively converts any pre-trained full-precision CNN into a low-precision version, whose weights are constrained to be either powers of two or zeros. Xu *et al.* propose to quantize the weights of a full precision NN model to binary or ternary weights by leveraging an alternating optimization approach applied at training time [23]: the accuracy loss of the resulting model followed to be negligible both for binary and ternary quantization. Ardakani *et al.* propose another binary and ternary quantization approach, where weights are sampled from a Bernoulli distribution, and the obtained values are regularized [24]. While most of the previous techniques applied to CNNs, very few of them applied to RNNs. As far as RNN quantization is concerned, the two proposed techniques are alternating multi-bit quantization [23] and Bernoulli sampling regularization [24]. NN quantization is orthogonal to our proposed solution since every edge server can deploy its quantization locally to speed up the local model training. Nevertheless, we will exploit weight quantization schemes to reduce the size of the NN (see Section 3).

FL and the corresponding decentralized training of NNs have been recently proposed by McMahan *et al.* [25]: the main motivating example for FL arises when the training data comes from users' interaction with mobile applications. Konečný *et al.* [26] propose an FL collaborative approach, which enables smartphones to collaboratively learn a shared prediction model while keeping all the training data on the device, thus decoupling the ability to train an AI model from the need to store the data in the Cloud. The authors describe two ways to reduce the uplink communication costs: i) using a smaller number of parameters for the model and ii) compressing the parameters by using a combination of quantization, random rotations, and sub-sampling before sending the model or the model update to the cloud node. Experiments on both CNNs and RNNs show only the accuracy of the proposed method, while the reduction in communication overhead is only theoretically calculated in two orders of magnitude. More recently, Reisizadeh *et al.* propose a communication-efficient FL method with periodic averaging and quantization [27]: local CNN models are updated at each local device and only periodically averaged at the cloud node; moreover, only a fraction of devices participate at each round of the training; finally, the local devices quantize the parameters of their local models, before uploading them to the cloud node.

Differently from [27], we introduce an intermediate computational layer, represented by the edge servers, which performs the quantization of the NN model, which, being a computationally intensive task, cannot be deployed on constrained IoT devices. To this aim, a publish-subscribe paradigm is applied within each privacy domain to locally transfer data from publisher IoT devices to the relative subscribed edge server. Moreover, we propose to quantize both the NN model broadcasted by the cloud node and the NN models sent by the edge servers to significantly reduce the background traffic related to the NN model training.

3. Proposed Framework

In this section, we present our proposed framework to perform FL of NNs, where: (i) data are produced by IoT nodes and collected by the relative edge servers belonging to the private domain, and (ii) training is jointly performed both on edge servers and in the Cloud. Edge servers perform local training in their isolated privacy domains, while the cloud node performs central aggregation of the locally learned models. To reduce Edge-Cloud data exchange, we propose a novel FL algorithm with quantization (FLQ), aimed at quantizing the NN weights during the FL procedure, i.e., when the NN models are transmitted from the edge servers to the cloud node and vice-versa. In Section 3.1 the communication paradigm underlying the proposed AI/FL algorithm is presented to evaluate the amount of data exchanged among nodes within the IoT-Edge-Cloud scenario. In Section 3.2, we describe our reference scenario and we outline how FL works in such a scenario; in Section 3.3 we describe the quantization approaches for NN that we will use to compress the NN models before transmission; finally, in Section 3.4 we describe our proposed FLQ and ΔFLQ algorithms. For clarity, Table 1 summarizes all notations used herein.

3.1. Communication Paradigm

IoT communication paradigms have been widely studied in the literature, and some studies have assessed the out-performance of information-centric networking, based on the publish/subscribe paradigm, w.r.t. the client/server paradigm [28]. More specifically, concerning IP-based data exchange solutions, the two most diffused IoT application protocols are the Constrained Application Protocol (CoAP) and the Message Queuing Telemetry Transport (MQTT) protocol, the latter being natively publish/subscribe. In MQTT, data producers (publishers) and data consumers (subscribers) are decoupled through a rendezvous node called *broker*. Data streams are organized into logical flows called *topics*. Each data packet is sent to the broker that maintains the list of active subscriptions and topics. Differently, CoAP adheres to the Representational State Transfer (REST) architectural style, providing support for resource-constrained environments. Resources are encapsulated by CoAP servers (data producers) and addressable by uniform resource identifiers. A CoAP client (data consumer) sends its request to retrieve a resource located on an IoT CoAP server. However, a publish/subscribe-like paradigm can be implemented

Table 1: Table of symbols

Symbol	Definition
n	Number of edge servers
$C_1, \dots C_n$	Edge servers
S	Cloud node
A	Generic dataset
D, D_i	A dataset
M, M_i	A NN model
$W, W_i, W_i', \Delta W$	Weights matrices in $\mathbb{R}^{n\times m}$
$\hat{W}, \hat{W}_i, \hat{\Delta} W$	Quantized matrices in $\mathbb{R}^{n\times m}$
τ	Number of local learning epochs
T	Number of federated learning rounds
k	number of quantization bits
$w_{\min}, w_{\max}$	Minimum and maximum values in W
$\boldsymbol{A^{(i)}}$	A matrix in $\mathbb{R}^{n\times m}$
$\boldsymbol{B^{(i)}}$	A matrix in $\{-1, +1\}^{n\times m}$
$\mathbf{1}_{n\times m}$	Element-wise product identity in $\mathbb{R}^{n\times m}$
ϵ_k	k-bit quantization error
$\alpha, \alpha_i, \alpha^*, \alpha_i^*$	Real values
$\boldsymbol{\hat{W}^{(i)}}$	Residual matrix

also in CoAP, by exploiting the observer pattern in RFC 7641 and the proxy functionality in IETF RFC 7252. Therefore, both MQTT and CoAP can be deployed in the proposed framework to implement such a communication paradigm. In [29], the authors showed how, by implementing the publish/subscribe-like exchange advantage, CoAP can outperform MQTT in terms of throughput, efficiency, and error resiliency. At present, however, MQTT is slightly more supported by a larger set of IoT embedded operating systems and micro-controllers, like Arduino, ESP6682, etc., and lightweight implementations of the MQTT broker are available for both constrained and unconstrained platforms [28]. In our reference scenario, illustrated in Figure 1, a set $\{C_1, \ldots, C_n\}$ of *edge servers* are connected to a central *cloud node* S. Every edge server is connected to a certain number of *IoT devices* and defines a *privacy domain*, i.e., a domain wherein data generated by the relative devices can move without privacy and/or security threats. Data contained in a privacy domain cannot move outside of that. Within a privacy domain, an MQTT broker is responsible to dispatch data between producers, i.e. the IoT nodes, and consumers. The MQTT broker service is naturally located on the edge server. Thus, an MQTT topic is applied to messages to identify the data flows from producers to consumers. Yet, a database service is deployed within the security domain and subscribes to the relative MQTT topic. Therefore, each edge server C_i manages, privately, a local dataset D_i, fed by the devices publishing on the relative i^{th} topic. For example, a local dataset can be composed of video and/or audio samples collected at home. Finally, the parameters of a NN are stored on each edge server, which can access the dataset D_i: the training mechanism derives a model M_i from the D_i dataset, which is, then, delivered to the Cloud for the FL task.

3.2. Federated Learning Scheme

During FL, the cloud node S aims at training a NN model encoded as a set of parameters W. The number of parameters depends on the structure of the NN being trained.

Algorithm 1 and Figure 2 illustrate the FL algorithm. The procedure $\textsc{Train}(W, D)$ performs the training of the NN model with parameters W exploiting a generic dataset D over a single training epoch, and returns the trained model. Namely, this procedure computes, in each training round, the loss function produced by the NN model with current parameters W when fed with the training data D. Then, the loss is back-propagated through the network, and the updated network parameters W are returned [30]. After the random initialization of the model W on the cloud node S (line 1 in Algorithm 1, $\textsc{Init}$ box in Figure 2), the federated learning procedure advances in (federated learning) *rounds* (line 2 in Algorithm 1). At the beginning of a round, the cloud node S distributes the current global model W to all the edge servers $C_1, \ldots, C_n$ in a multicast manner (line 4 in Algorithm 1, double-lined arrows in Figure 2). Then, each edge server C_i performs an independent training of the global model W using its own local dataset D_i over

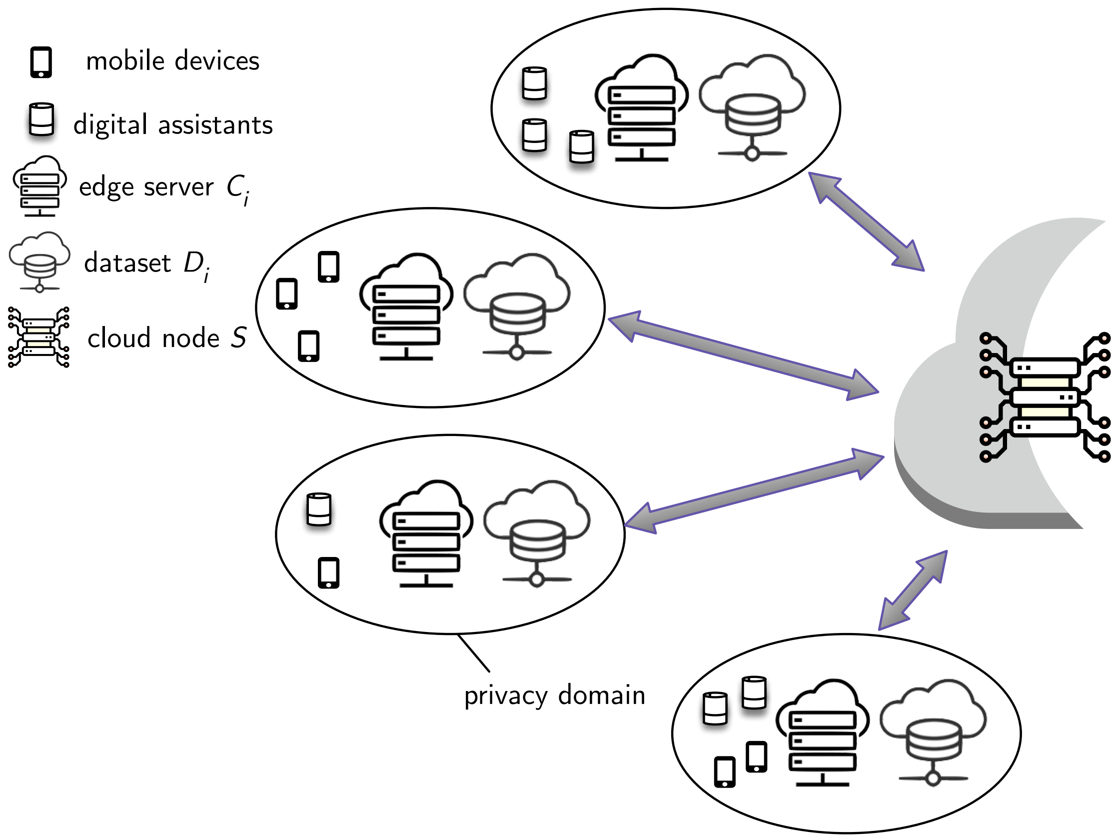

Figure 1: Federated learning reference scenario.

τ (local learning) *epochs*, generating a locally trained model W_i (lines 5-6 in Algorithm 1, TRAIN box in Figure 2). The edge servers send back their locally trained models to the cloud node (line 7 in Algorithm 1, double-lined arrows in Figure 2) and, finally, the cloud node merges the n locally trained models into a new global model (FEDAVG box in Figure 2), to be distributed again during the next round. The merging of the local models is an element-wise weighted average of the matrices $W_1, \ldots, W_n$ (line 8 in Algorithm 1). This procedure is repeated until a maximum number T of rounds is reached, and the final global model W is returned.

The FL algorithm just described allows each edge server to keep its training data, collected through the respective server nodes, in its local privacy domain. In doing so, FL aims at producing a final NN model whose accuracy is as much as possible close to the accuracy of a NN model generated by a global training procedure performed on the aggregated dataset $D_1 \cup \ldots \cup D_n$.

Algorithm 1: The FL algorithm.

Input : n local datasets $D_1, \ldots, D_n$ at edge servers $C_1, \ldots, C_n$
a number of rounds T
a number of epochs τ

Output: A matrix W of NN weights

$\mathrm{FL}(D, D_1, \ldots, D_n, T, \tau)$**:**

Matrix W is randomly initialized
for T rounds **do**
  for $i \leftarrow 1$ **to** n **do**
    S sends W to C_i as W_i
    for τ epochs **do**
      $W_i \leftarrow \textsc{Train}(W_i, D_i)$
    C_i sends W_i to S
  $W \leftarrow \sum_{i=1}^{N} \frac{|D_i|}{|D|} W_i$
return W

3.3. Model Quantization Schemes

Model quantization aims at computing a representation $\hat{\boldsymbol{W}}$ of the NN weights $\boldsymbol{W}$ with a smaller memory footprint. In particular, *k-bit binary quantization* maps the weights of a NN model to $\{-1, +1\}^k$.

For example, the *binary quantization* (BinQ), introduced in [31], assumes $k = 1$, and every weight is trivially quantized according to its sign. This quantization can be easily extended to k bits as shown in [32].

Besides this simple quantization heuristic, the NN quantization schemes can be grouped into *random* and *error minimization* quantizations.

Random quantization. Random quantization schemes select the bits representing every weight according to some probability. These approaches give theoretical guarantees on the expected value of the error introduced by the quantization scheme utilized.

The *probabilistic quantizer* (ProbQ) is a 1-bit random quantization scheme [33]. Let $w_{\max}$ and $w_{\min}$ be the maximum and minimum values among the elements of $\boldsymbol{W}$,

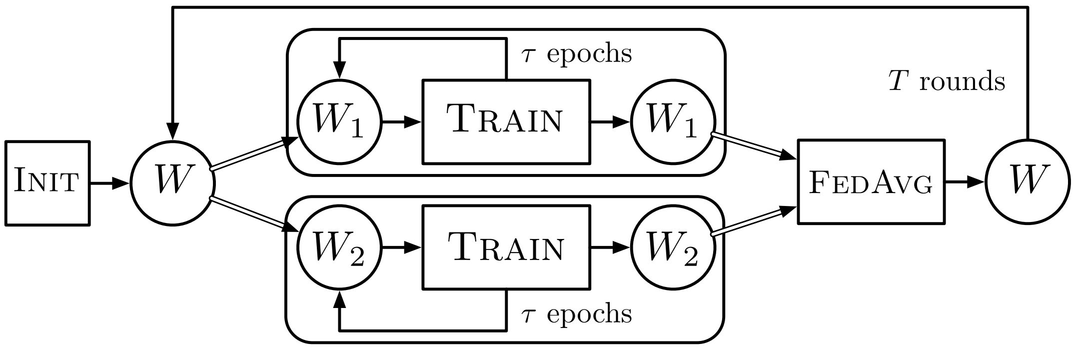

Figure 2: Main phases of the FL algorithm.

respectively. The elements $\hat{w}_{ij}$ of the quantized matrix $\hat{\boldsymbol{W}}$ are quantized as follows:

$$\hat{w}_{ij} = \begin{cases} +w_{\max} & \text{with probability } p(w_{ij}), \\ -w_{\min} & \text{otherwise.} \end{cases} \tag{1}$$

where $p(x)$ is the function:

$$p(x) = \frac{x - w_{\min}}{w_{\max} - w_{\min}}. \tag{2}$$

The *low-precision quantizer* (LowQ) is a multi-level random quantization scheme [34]. Let s denote the number of quantization levels, i.e., $k = \lceil \log_2(s) \rceil$, and let $\omega_{ij} = s\frac{w_{ij}}{\|W\|_F}$, where $\|\cdot\|_F$ is the Frobenius norm. The elements $\hat{w}_{ij}$ of the quantized matrix $\hat{\boldsymbol{W}}$ are

$$\hat{w}_{ij} = \mathrm{sgn}(w_{ij})(\lfloor \omega_{ij} \rfloor + \delta_{ij})\|W\|_F, \tag{3}$$

where the random variable δ_{ij} has the following distribution:

$$\delta_{ij} = \begin{cases} 1 & \text{with probability } \omega_{ij} - \lfloor \omega_{ij} \rfloor, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The $\mathrm{sgn}(x)$ function is the most suited function to quantize positive and negative values to a single bit.

Error minimization quantization. Random quantization schemes do not make any assumption on the actual magnitude of the error introduced by the quantization schemes. Now, given a weight matrix $\boldsymbol{W} \in \mathbb{R}^{n\times m}$, we look for an approximation matrix $\hat{\boldsymbol{W}} \in \mathbb{R}^{n\times m}$ such that:

$$\boldsymbol{W} \simeq \hat{\boldsymbol{W}} = \sum_{i=1}^{k} \boldsymbol{A}^{(i)} \odot \boldsymbol{B}^{(i)}, \tag{5}$$

where $\boldsymbol{A}^{(i)} \in \mathbb{R}^{n\times m}$ is a matrix with some special structure, $\boldsymbol{B}^{(i)} \in \{-1,+1\}^{n\times m}$ is a binary matrix, k is the number of bits used to represent every entry of the original matrix $\boldsymbol{W}$, and $\odot$ is the element-wise multiplication[1] (Hadamard product). We denote the element-wise product identity as $\mathbf{1}_{n\times m}$.

To investigate the optimality of the approximation matrix $\hat{\boldsymbol{W}}$, let's define the *k-bit quantization error* ε_k between the matrix $\boldsymbol{W}$ and its quantized version $\hat{\boldsymbol{W}}$ as:

$$\varepsilon_k(\boldsymbol{W}, \hat{\boldsymbol{W}}) = \|\boldsymbol{W} - \hat{\boldsymbol{W}}\|_F = \left\| \boldsymbol{W} - \sum_{i=1}^{k} \boldsymbol{A}^{(i)} \odot \boldsymbol{B}^{(i)} \right\|_F. \tag{6}$$

The optimal 1-bit quantization is defined as the solution $\boldsymbol{A}^*$, $\boldsymbol{B}^*$ of the following optimization problem:

$$\begin{aligned} &J(\boldsymbol{A}, \boldsymbol{B}) = \varepsilon_1(\boldsymbol{W}, \hat{\boldsymbol{W}}) = \|\boldsymbol{W} - \boldsymbol{A} \odot \boldsymbol{B}\|_F, \\ &\boldsymbol{A}^*, \boldsymbol{B}^* = \underset{\boldsymbol{A},\boldsymbol{B}}{\arg\min}\, J(\boldsymbol{A}, \boldsymbol{B}). \end{aligned} \tag{7}$$

As reported in [32], the optimal solution to problem (7) for $\boldsymbol{A} = \alpha \mathbf{1}_{n\times m}$, where $\alpha \in \mathbb{R}$, is

$$\alpha^* = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} |w_{ij}| \qquad \boldsymbol{B}^* = \operatorname{sgn}(\boldsymbol{W}). \tag{8}$$

In general, directly minimizing the k-bit quantization error described in problem (6) with $k > 1$ is NP-hard [35]. All the proposed schemes minimizing the quantization error address such a problem with algorithms based on some kind of heuristics. All these schemes assume $\boldsymbol{A}^{(i)} = \alpha_i \mathbf{1}_{n\times m}$, where $\alpha_i \in \mathbb{R}$.

The *residual quantization* (ResQ) [36] is a k-bit error minimization quantization leveraging the residual matrices $\hat{\boldsymbol{W}}^{(i)}$, defined as

$$\hat{\boldsymbol{W}}^{(i)} = \boldsymbol{W} - \sum_{j=1}^{i} \boldsymbol{A}^{(j)} \odot \boldsymbol{B}^{(j)} \text{ for } i = 1, \dots, k, \tag{9}$$

with $\hat{\boldsymbol{W}}^{(0)} = \boldsymbol{W}$.

Sequentially, for each $i = 1, \dots, k$, ResQ minimizes the residual errors $\|\hat{\boldsymbol{W}}^{(i)} - \boldsymbol{W}\|_F$ one at a time. The optimal solutions to these k minimization problems are similar to the

[1]It produces a matrix with the same dimensions as the operands, where each output element i, j is the product of corresponding elements i, j of the two input matrices.

optimal solution of the 1-bit quantization scheme, i.e.:

$$\begin{aligned} \alpha_i^* &= \frac{1}{nm}\sum_{j=1}^{n}\sum_{k=1}^{m}|\hat{w}_{jk}^{(i-1)}|, \\ \boldsymbol{B^{(i)*}} &= \operatorname{sgn}(\boldsymbol{\hat{W}^{(i-1)}}). \end{aligned} \tag{10}$$

In most cases, solutions to these k minimization problems will not be optimal for the original minimization problem in (6). Hence, the α_i values can be recomputed at every step, once the first j optimal $\boldsymbol{B^{(i)*}}$ matrices have been computed. This is carried out by solving the following minimization problem:

$$\begin{aligned} J(\alpha_1,\ldots,\alpha_j) &= \left\|\boldsymbol{W} - \sum_{i=1}^{j}\alpha_i \boldsymbol{B^{(i)*}}\right\|_F, \\ \alpha_1^*,\ldots,\alpha_j^* &= \operatorname*{arg\,min}_{\alpha_1,\ldots,\alpha_j} J(\alpha_1,\ldots,\alpha_j). \end{aligned} \tag{11}$$

Let us define the *vectorization* operator $\text{vec}(\boldsymbol{X})$, which returns a column vector, whose elements are the stacking of the columns of the matrix $\boldsymbol{X}$ on top of one another, and let $\boldsymbol{B_j}$ be the matrix, whose columns are $\text{vec}(\boldsymbol{B^{(1)*}}),\ldots,\text{vec}(\boldsymbol{B^{(j)*}})$. The least squares solution of problem (11) at step j is given by:

$$[\alpha_1,\ldots,\alpha_j]^\top = (\boldsymbol{B_j}^\top \boldsymbol{B_j})^{-1}\boldsymbol{B_j}^\top \text{vec}(\boldsymbol{W}). \tag{12}$$

During the re-computation of the α_j values in Eq. (12), the computed $\boldsymbol{B^{(i)^*}}$ matrices are no longer optimal for problem (7). Starting with the solution given by Eq. (10), the *iterative quantization* (IterQ) [23] iteratively re-computes these matrixes as follows:

1. compute the $\alpha_1,\ldots,\alpha_k$ values with Eq. 12 with all $\boldsymbol{B^{(1)^*},\ldots,B^{(k)^*}}$ matrices known;
2. build all possible 2^k combinations of the $\alpha_1,\ldots,\alpha_k$ values with $-1,+1$ and store them in a binary search tree data structure[2];
3. for each element of $\boldsymbol{W}$, select the closest combination and assign the corresponding values to the $\boldsymbol{B^{(1)^*},\ldots,B^{(k)^*}}$ matrices accordingly.

3.4. Federated Learning with Quantization

We propose to improve the efficiency of FL by reducing the amount of data being transferred from the edge servers to the cloud node. To do so, we propose to apply

[2]For $k=2$, assuming $\alpha_1>\alpha_2>0$, the combinations are $-\alpha_1-\alpha_2$, $-\alpha_1+\alpha_2$, $\alpha_1-\alpha_2$ and $\alpha_1+\alpha_2$.

Algorithm 2: The proposed FLQ algorithm.

Input : n local datasets $D_1, \ldots, D_n$ at edge servers $C_1, \ldots, C_n$

a number of rounds T

a number of epochs τ

Output: A matrix W of NN weights

FLQ$(D, D_1, \ldots, D_n, T, \tau)$:

1 Matrix W is randomly initialized
2 **for** T rounds **do**
3   $W \leftarrow$ Quantize(W)
4   **for** $i \leftarrow 1$ **to** n **do**
5   S sends W to C_i as W_i
6   **for** τ epochs **do**
7       $W_i \leftarrow$ Train(W_i, D_i)
8     $W_i \leftarrow$ Quantize(W_i)
9   C_i sends W_i to S
10   $W \leftarrow \sum_{i=1}^{n} \frac{|D_i|}{|D|} W_i$
11 **return** W

quantization schemes to the models being transferred during the execution of Algorithm 1. In particular, we propose to perform a first quantization after the model is updated by the edge servers, and a second quantization, before the global model is transferred from the cloud node to the edge servers.

Algorithm 2 and Figure 3 illustrate our proposed federated learning with quantization (FLQ) algorithm. FLQ aims at obtaining a NN model W while i) reducing the volume of data (in bytes) transferred among the cloud node and the edge servers while exchanging the model updates, and ii) preserving the privacy of the data stored on the edge servers.

In the FLQ algorithm, an FL round is now made up of 6 steps: (1) the cloud node applies quantization to its global model (line 3 in Algorithm 2, Q box on the left in Figure 3), (2) the cloud node sends its quantized global model (the red W on the left in Figure 3) to every edge server (line 5 in Algorithm 2, double-lined arrows in Figure 3), (3) each edge server trains its own local model, starting from the received quantized

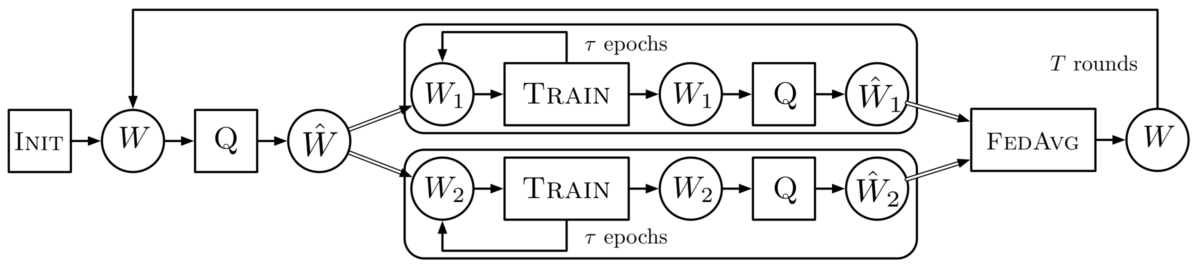

Figure 3: Main phases of FLQ algorithm.

global model, on its data (line 7 in Algorithm 2, TRAIN box in Figure 3), (4) each edge server applies quantization to its locally trained model (line 8 in Algorithm 2, Q box on the right in Figure 3), (5) each edge server sends its quantized locally trained model (W on the right in Figure 3) back to the cloud node (line 9), and (6) the cloud node applies weighted averaging to the quantized local models received (line 10 in Algorithm 2, FEDAVG box in Figure 3). The procedure QUANTIZE(W) performs the quantization of the parameters W exploited by NN model. In principle, it may implement any of the quantization approaches described in Section 3.3.

Since quantization can introduce large errors in the model weights in later rounds due to the smaller and smaller changes applied to the weights during training, we also propose ΔFLQ, a variant of the FLQ algorithm, illustrated in Algorithm 3. The ΔFLQ algorithm applies the FLQ algorithm not to the full models, but on their difference before and after local training, at the edge servers, and before and after federated averaging, at the cloud node. Initially, a random matrix is initialized on the cloud node and distributed to all edge servers[3] (lines 1-3). Then, at every round, the ΔW matrix, initially set to 0, is quantized (line 6) and transferred to the edge servers (line 8). Each edge server sums this matrix to its local model (line 9), and then it performs the local training on a *copy* W_i' of their local models (lines 10-12). At the end, each edge server computes the ΔW_i matrix, i.e., the difference between the local model after and before the training, quantizes it and transfers it back to the cloud node (lines 13-14). Finally, the cloud node computes the weighted average of the received differences and stores it in the ΔW matrix (line 15).

In the next section, we experimentally measure the accuracy of the final learned models produced by FLQ and ΔFLQ w.r.t. other training algorithms, such as local learning

[3]In practice the matrices can be initialized by communicating the random seed used to generate the random weights.

Algorithm 3: The proposed ΔFLQ algorithm.

Input : n local datasets $D_1, \ldots, D_n$ at edge servers $C_1, \ldots, C_n$,
a number of rounds T
a number of epochs τ

Output: A matrix W of NN weights

ΔFLQ$(D, D_1, \ldots, D_n, T, \tau)$:

Matrix W is randomly initialized
for $i \leftarrow 1$ **to** n **do**
    S sends W to C_i as W_i
$\Delta W \leftarrow 0$
for T rounds **do**
    $\Delta W \leftarrow \textsc{Quantize}(\Delta W)$
    for $i \leftarrow 1$ **to** n **do**
        S sends ΔW to C_i as ΔW_i
        $W_i \leftarrow W_i + \Delta W_i$
        $W_i' \leftarrow W_i$
        for τ epochs **do**
            $W_i' \leftarrow \textsc{Train}(W_i', D_i)$
        $\Delta W_i \leftarrow \textsc{Quantize}(W_i' - W_i)$
        C_i sends ΔW_i to S
    $\Delta W \leftarrow \sum_{i=1}^{n} \frac{|D_i|}{|D|} \Delta W_i$
    $W \leftarrow W + \Delta W$
return W

and federated learning, and we evaluate the benefits of our FLQ and ΔFLQ algorithms at reducing the data transmitted among the cloud node and edge servers when different quantization approaches are exploited.

4. Experimental Setup

We experiment FLQ and ΔFLQ on the *next word prediction* task that, given a sentence context, i.e., a sequence of words, aims at learning a language model, i.e., a probability distribution over the words conditioned on a given context, to predict the *next* most likely word that appears after the context [23]. As such, this task can be seen as a time series prediction problem where data points are words in a given vocabulary, and given the first n words, we aim at predicting the $n+1$-th word. It is a common use case used in mobile applications like, for example, predictive keyboards or personal voice assistants. In our scenario, we assume that every edge server has a private collection of text written by the user, and we aim at learning a global language model without disclosing the private collections.

Dataset. We conduct the experimental evaluation using the WikiText-2 dataset [37]. The WikiText-2 dataset is composed of 720 text articles: 600 articles in the training set and 60 in both the validation and the test sets, respectively. The training set consists of $2,088,628$ tokens, while the validation and test sets consist of $217,646$ and $245,569$ tokens, respectively. The vocabulary consists of $33,278$ words. In the following analysis, we use the training set to train our models while the validation and test sets are used for early stopping the training of the model, and to measure its final performance, respectively.

Neural Architecture. We perform next word prediction [12] by training a Long Short Term Memory Network (LSTM). In our experiments, following the setting and hyperparameters laid out in [23], we train an LSTM model with an input embedding layer with 200 neurons, two hidden LSTM layers, each one composed of 512 neurons, and an output linear layer with $33,278$ outputs, one per word in the vocabulary. We use a batch size of 100 words. The training of the network is regularized with learning rate decay, i.e., if the validation loss does not decrease in an epoch, the learning rate, initially set to 20, is decreased by a factor of 4, up to a minimum value of 10^{-4}. For regularization purposes, we also employ a dropout strategy by setting the dropout rate to 0.5. Moreover, to address the gradient explosion problem when training the LSTM, we set the gradient clipping to 0.25, the gradient norm clipping to 0.3, and the weight clipping to 1.0. The training of the LSTM is performed by minimizing the cross-entropy loss until the learning rate decreases below the minimum value of 10^{-4} (early-stopping condition) or for a maximum of 80 epochs.

Implementation Details. The experimental framework is implemented in PyTorch 1.4.0[4]. Experiments are performed using a Tesla T4 GPU on an AMD EPYC CPU clocked at 2.2 GHz and 24 GB of RAM. The machine works as both cloud node and edge server and we experiment with 2 edge servers. In the evaluation of federated scenarios, edge servers are emulated by training the LSTM models locally on the machine on equally-sized and disjoint partitions of the training set. All the quantization strategies tested in this paperwork quantize independently the weight matrices of the LSTM. No quantization is performed on the input embedding layer.

5. Experimental Evaluation

We now present a comprehensive analysis of the performance of the FLQ and ΔFLQ algorithms by investigating two main research questions (RQs):

RQ1. What is the impact of the FLQ and ΔFLQ algorithms on the accuracy of the learned models, measured in terms of validation and test losses?

RQ2. What is the reduction in terms of data transmitted between edge servers and cloud node by the FLQ and ΔFLQ algorithms w.r.t. standard FL approaches, i.e., without quantization?

To investigate our RQs, we designed a comprehensive experimental setting structured in the following four scenarios.

Local Learning (LL). In this scenario, we perform the training of the LSTM locally on the cloud node on the full training dataset.

Local Quantization (LQ). In this scenario, we perform the training of the LSTM locally on the cloud node by locally applying the quantization schemes introduced in Section 3. As for LL, the training of the LSTM is performed on the full training dataset.

Federated Learning (FL): In this scenario, we experiment with the standard FL algorithm. We learn the LSTM on each edge server on its partition of the training data. At the end of each round, the edge servers send their LSTM model to the cloud node, which performs the federated averaging step. The model is then sent back to all the edge

[4]We plan to release the code upon acceptance of the paper.

servers. The edge servers compute the loss on the training set, while the validation and test losses are computed on the cloud node after the federated averaging step.

Federated Learning with Quantization (FLQ): In this scenario, we experiment with the FLQ and ΔFLQ algorithms. The edge servers perform model quantization before sending their model to the cloud node. Then, the cloud node performs a federated averaging of the received models and quantizes again the result before sending the model back to the edge servers.

Firstly, we assess the performance of the FL scenario w.r.t. the LL scenario. Secondly, we apply quantization both in local scenario (LQ) and in the federated scenario (FLQ). In both scenarios, we first assess the performance of our FLQ algorithm, then we assess the performance of our ΔFLQ algorithm. The rationale behind this experimental methodology is to evaluate the impact of federated learning on the performance of the LSTM, i.e., the effect of working on partitioned training data for each worker. Then, we assess the impact of quantization techniques both in the LQ scenario, i.e., where no data partitioning is introduced by federated learning, and in the FLQ scenario, i.e., with both quantization and data partitioning. Finally, we compare the size of the models transmitted in the federated scenarios to quantify the data volume reductions yielded by the quantization schemes considered.

We perform our experiments by reporting the effectiveness of the LSTM in terms of: i) test loss on the final model and ii) validation loss observed during the training process to analyze the convergence speed of each method. The size of the models is computed by counting the number of bits of both the quantized and un-quantized parameters of our LSTM.

A note on time units. Local and federated scenarios employ different portions of the training data, depending on the number of workers, and perform different local training epochs and federated training rounds. The total wall-clock time required to train the different models depends on four factors: (i) the number of workers processing their portions of the dataset, (ii) the number of learning epochs used by each worker to locally learn its model, (iii) the number of federated rounds used to collect the local models and to compute their federated averaging, and (iv) the quantization process. In the following, we report on the x-axis the training time measured in *time unit*s. We assume that the time required to perform a single training epoch on a single worker on the whole dataset corresponds to 1-time unit. On w workers, the whole dataset, divided among all workers, requires $1/w$ time units to be processed in a single learning epoch in parallel on w workers. On w workers, τ training epochs require τ/w time units to perform the local training. Finally, by measuring the time required by the quantization, and the model transmission processing, we found out that their overhead is negligible w.r.t. the time required to perform a single learning epoch. For this reason, we do not take it into account.

5.1. Local Learning and Federated Learning Scenarios

In this section, we propose an experimental analysis of the performance of the LSTM network when trained both in LL and FL scenarios. Figure 4 reports the validation loss of the LSTM during the training in the FL scenario according to Algorithm 1 by varying τ in $\{1, 2, 4, 8, 16\}$, i.e., the number of epochs performed by two edge servers before transmitting the model to the cloud node for federated averaging.

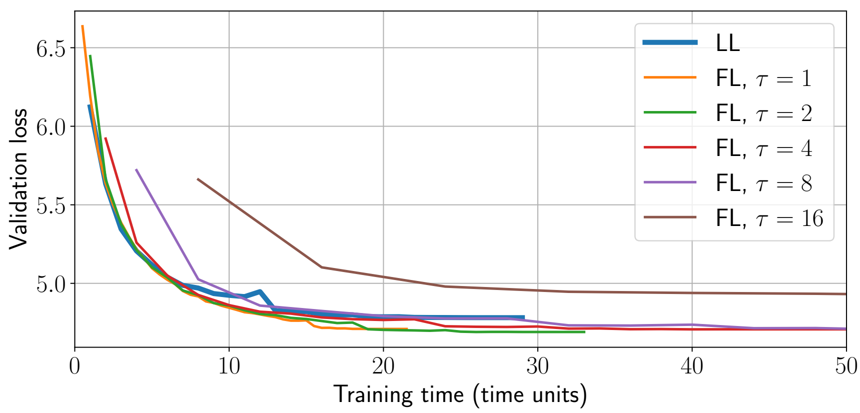

Figure 4: Validation loss of the LSTM network in the FL scenario by varying the number of epochs τ in a round of federated learning. We also report the performance on the LL scenario (red bold line).

We compare the performance achieved in the FL scenario with the one achieved in the LL scenario (bold red line). When training in LL, the validation loss drops significantly very soon, reaching the minimum value of 4.73 after 29 units of time. Moreover, the plot shows a clear trend: the more epochs are performed locally to each worker in a round of federated learning, i.e., the larger values of τ, the slower is the convergence of the validation loss to its minimum. The rationale of this behavior lies in the role of federated averaging, i.e., (FedAvg), which allows sharing the knowledge learned by the two workers in isolation in a single LSTM model that is then used locally by edge servers to continue the learning. Moreover, in the FL scenario, the neural network achieves a better performance in terms of validation loss than the LL corresponding loss with small values of τ, i.e., $\{1, 2, 4, 8\}$. This does not hold for $\tau = 16$, where the FL minimum validation loss does not outperform the corresponding loss of the LL scenario. The rationale behind the better performance shown in the FL scenario w.r.t. the LL scenario relies in the federated averaging step that introduces a regularization effect in the resulting merged model, thus achieving a better generalization of the model learned.

Table 2: Rounds and time units to minimum validation loss (Rounds & Time), minimum validation loss (Val. loss) and test loss of the LSTM network model in the LL and FL scenarios, by varying τ.

	τ	Rounds	Time	Val. loss	Test loss
LL	–	27	27	4.78	4.73
FL	1	37	18.5	4.71	4.64
	2	25	25	4.69	4.64
	4	19	38	4.71	4.63
	8	27	108	4.70	4.66
	16	28	224	4.81	4.76

To better show the impact of τ on the convergence speed, we deepen the analysis by reporting, given τ, the number of federated rounds needed by the LSTM to achieve the minimum observed validation loss. Note that the time units required to perform a federated learning round depend on τ: a single federated round with 2 workers requires $\tau/2$ time units. We report the results in Table 2. By increasing τ, the results show a clear slow down of the time units required by the NN to reach the minimum validation error. In a LL scenario, the LSTM needs 27 epochs, i.e., 27 time units, to reach the minimum validation loss. Instead, in a FL scenario, when $\tau \in \{1, 2\}$, the FL algorithm is able to reach the minimum validation loss in just $37 \times 1/2 = 18.5$ and $25 \times 2/2 = 25$ time units, respectively, while it needs $28 \times 16/2 = 224$ time units to reach the minimum validation loss when $\tau = 16$. Table 2 also reports the final test loss achieved by the LSTM in the LL and FL scenarios. The results consistently report the same trend identified for the validation loss, i.e., the test loss achieved in the FL scenario outperforms the one achieved in the FL one for $\tau \in \{1, 2, 4, 8\}$ but not for $\tau = 16$.

To conclude, federated learning allows to gain effectiveness on the final LSTM model with the big advantage of keeping data stored on edge servers in a decentralized manner.

5.2. Local Quantization and Federated Learning with Quantization Scenarios

We now focus our analysis on the impact of quantization when employed in the LQ and FLQ scenarios. Table 3 reports the test loss achieved when training an LSTM model with the 1-, 2-, and 3-bit quantization schemes discussed in Section 3 with $\tau = 1$ in the LQ and FLQ scenarios.

Table 3: Test loss of the LSTM network model in the LQ and FLQ scenarios for 1-, 2-, and 3-bit quantization schemes.

Quantization	LQ			FLQ		
	1 bit	2 bits	3 bits	1 bit	2 bits	3 bits
BinQ	33.19	-	-	39.32	-	-
ProbQ	13.28	-	-	9.71	-	-
LowQ	-	6.91	6.88	-	6.99	6.95
ResQ	-	7.04	7.03	-	6.85	6.86
IterQ	-	7.62	6.89	-	6.89	6.85

Comparing the test losses in the LQ scenario, the two 1-bit random quantization schemes, i.e., BinQ and ProbQ, result in larger test losses than the other schemes, i.e., LowQ, ResQ, and IterQ. This is due to the very limited precision obtained by the former schemes in quantizing each model weight on a single bit, while the latter schemes, using 2 bits, result in smaller quantization errors. For sake of space, we do not include further results of BinQ and ProbQ schemes, as their performance is not competitive with respect to LowQ, ResQ, and IterQ. Concerning the FLQ scenario, the test losses of LowQ, ResQ, and IterQ are very close to the value in the corresponding LQ scenario, i.e., -1.2%, $+2.8\%$, and $+0.3\%$ respectively.

Figure 5 reports the validation loss achieved by LowQ, ResQ, and IterQ when training an LSTM model with 2-bit quantization and $\tau = 1$ in the LQ and FLQ scenarios. Similar results are obtained with higher values of τ and with 3-bit quantization, with just a slight increase in convergence speed, so we do not report them. In both scenarios, the

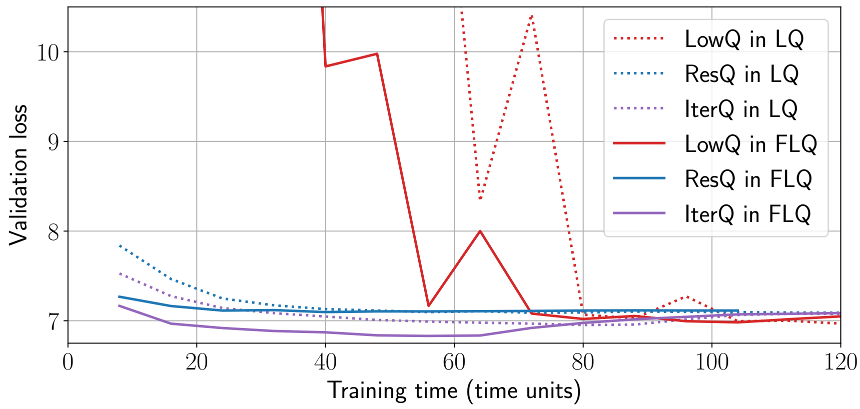

Figure 5: Validation loss of the LSTM network in the LQ and FLQ scenarios (dashed and solid lines, respectively) with 2-bit quantization schemes and $\tau = 1$.

ResQ and IterQ quantization schemes converge to the minimum validation loss quickly, i.e., in 4 time units or less. The LowQ scheme starts with a higher loss value even if it eventually converges towards the minimum value. However, the time needed by LowQ to converge is 33% more than the time needed by the previous two quantization schemes. By comparing the LQ and FLQ scenarios, both the ResQ and IterQ quantization schemes in the FLQ scenario converge slightly slower than in the LQ scenario, while the LowQ quantization scheme converges faster in the FLQ scenario. We can conclude that the considered quantization schemes can be successfully applied in federated learning scenarios. Our experiments show that quantization in the FLQ scenario achieves the same performance of the LQ scenario with no degradation in the effectiveness of the LSTM model.

We now investigate the impact of the number of epochs τ on the performance. In particular, we assess if larger values of τ allow improving the effectiveness of the LSTM in the FLQ scenario, measured in terms of test loss. As τ controls the number of epochs performed on each edge server, the intuition is the following: the more epochs are performed on the edge servers, the more the error introduced by quantization decreases, due to the full precision used in the local learning process on the edge servers. In fact, no quantization is performed during the training process on an edge server, as it is performed only before sending the model or after receiving it from the cloud node.

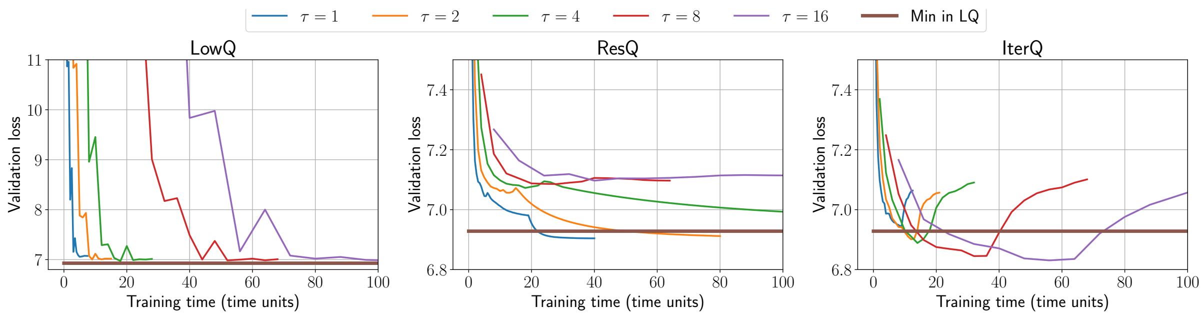

Figure 6: Validation loss of the LSTM network in an FLQ scenario using the FLQ Algorithm with 2-bit quantization schemes, by varying τ.

Table 4: Minimum validation and test losses of the LSTM network model in the FLQ scenario using the FLQ Algorithm with 2-bit quantization schemes, by varying τ.

τ	Minimum validation loss			Test loss		
	LowQ	ResQ	IterQ	LowQ	ResQ	IterQ
1	7.06	6.90	6.95	6.99	6.85	6.89
2	7.00	6.91	6.90	6.94	6.86	6.86
4	6.97	6.97	6.89	6.92	6.93	6.85
8	6.99	7.09	6.85	6.93	7.04	6.80
16	6.98	7.10	6.83	6.93	7.05	6.77

Figure 6 reports the validation losses achieved using the FLQ Algorithm with 2-bit schemes LowQ, ResQ, and IterQ, by varying τ in $\{1, 2, 4, 8, 16\}$, and Table 4 reports the corresponding minimum validation and test losses. As expected, the results show that, for all quantization schemes, the convergence is slower as τ increases. The results for ResQ show that the validation and test losses are negatively affected by larger values of τ. Moreover, ResQ and LowQ do not allow the LSTM to achieve the minimum validation loss reported in the LQ scenario. In particular, only for $\tau \in \{1, 2\}$, ResQ is able to reach and outperform the minimum validation loss observed in the LQ scenario, even if, for $\tau = 2$, the convergence to the minimum is significantly slower than for $\tau = 1$. Instead, the validation and test losses using IterQ benefit from larger values of τ. The reported results show that, when increasing τ, the effectiveness of the model, i.e., the minimum validation loss, decreases significantly below the minimum achieved in the LQ scenario. A side effect of the increase of τ for IterQ is the convergence speed, which results to be slower. The convergence speed is important in the FLQ scenario because it potentially affects the number of data transmissions between edge servers and the cloud node, i.e., the slower the convergence, the larger the number of transmissions. When increasing τ, the slower convergence speed is reported for all the three quantization schemes. However, when IterQ is used, larger τ values boost the final effectiveness. Moreover, while the minimum validation loss for $\tau = 1$ is reached after 8 units of time, i.e., $2 \times 8 = 16$ model transmissions, with $\tau = 16$ the minimum validation loss is reached after 48 units of time, i.e., $2 \times 48/16 = 6$ model transmissions, thus more than halving the number of transmissions required to achieve a more accurate model.

To conclude, we observe that a larger number of epochs performed locally on the edge servers slows down the convergence of the whole training, even if this does not always imply a larger number of transmissions. However, we found that large values of τ impact positively on the validation and test losses of the LSTM when employing the IterQ quantization scheme. In this case, the final model in the FLQ scenario outperforms the final model in the LQ scenario for all τ in $\{2, 4, 8, 16\}$.

5.3. Federated Learning with Delta Quantization

While in the FLQ scenario our FLQ algorithm produces a LSTM network with validation and test losses almost identical to the values of the LSTM network trained in the LQ scenario, the performance of the models trained with FLQ (test loss 6.93) are higher than the performance of the models trained with FL (test loss 4.73). We ascribe this decrease in test loss to quantization, since it can introduce errors in the model weights. In particular, during the training of the LSTM, the high variance in the small changes applied to the weights at later rounds can have a negative impact on the whole training. To mitigate this effect, in Section 3 we proposed the ΔFLQ algorithm, a variant of the FLQ algorithm. The ΔFLQ algorithm quantizes the *changes* in the network weights exchange between the edge servers and the cloud node. We now investigate the performance of ΔFLQ when applied to the FLQ scenario. As before, we report the performance of LowQ, ResQ, and IterQ when varying τ in Figure 7 and Table 5.

The introduction of changes quantization to LowQ negatively impacts the performance of the final LSTM network. Indeed, the validation loss never converges to the minimum achieved in LQ (green horizontal line). Moreover, for larger values of τ, the

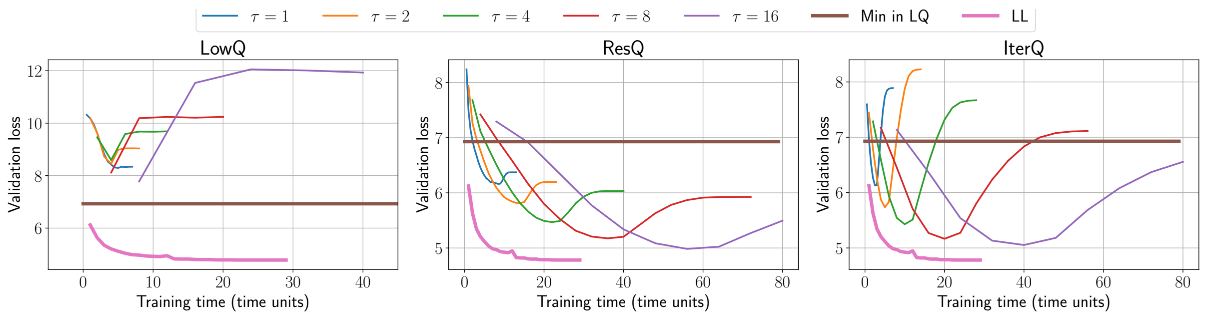

Figure 7: Validation loss of the LSTM network in the FLQ scenario using the ΔFLQ Algorithm with 2-bit quantization schemes, by varying τ.

Table 5: Minimum validation and test losses of the LSTM network model in the FLQ scenario using the ΔFLQ Algorithm with 2-bit quantization schemes, by varying τ.

τ	Minimum validation loss			Test loss		
	LowQ	ResQ	IterQ	LowQ	ResQ	IterQ
1	8.29	6.16	6.13	8.27	6.10	6.06
2	8.45	5.82	5.74	8.42	5.76	5.66
4	8.59	5.47	5.43	8.54	5.41	5.36
8	8.17	5.17	5.17	8.07	5.11	5.10
16	7.78	4.98	5.05	7.73	4.91	4.99

validation loss grows significantly, thus revealing the presence of heavy overfitting on the local training data. On the other side, ResQ and IterQ show a completely different behaviour. For these two quantization schemes, the introduction of changes quantization improves the performance of the FLQ algorithm. Indeed, ResQ and IterQ outperform the performance achieved using FLQ for all values of τ considered. Moreover, when increasing τ, the validation loss gets closer to the value achieved in the LL scenario (pink line), even though convergence is slower.

Finally, we evaluate the impact of the number of edge servers on the best 2-bit quantization scheme in the FLQ scenario using the ΔFLQ Algorithm, i.e., IterQ with $\tau = 16$ local learning epochs. We report the performance of IterQ when varying the number of edge servers w in Figure 8. When more edge servers are deployed, the time units required to reach the minimum validation loss decrease with the number of servers, since more workers can carry out independently and in parallel the local learning over smaller por-

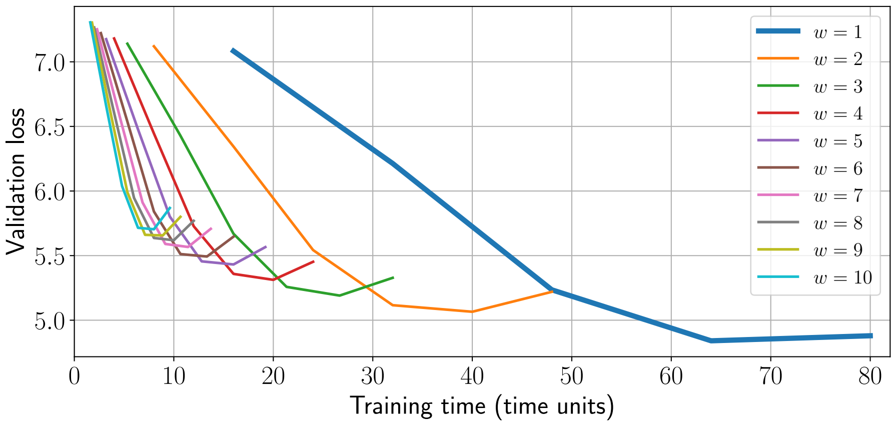

Figure 8: Validation loss of the LSTM network in the FLQ scenario using the ΔFLQ algorithm with 2-bit IterQ quantization and $\tau = 16$ by varying the number of edge servers w.

tions of the whole dataset. At the same time, the smaller local dataset portions available at each edge server lead to a decrease in the performance of the whole learned model. In fact, the minimum validation loss of the final model increases as the number of edge servers increases.

Figure 9 illustrates the validation loss with 4 and 8 edge servers in the FLQ scenario using the FLQ and ΔFLQ algorithms with 2-bit IterQ quantization and $\tau = 16$, with a fraction $f = 0\%$, 25%, 50%, and 75% of faults. The faults happen after the second federated round, and the faulty edge servers are randomly selected at every federated round until the minimum validation loss is reached. For both 4 and 8 edge servers, the ΔFLQ algorithm reaches a smaller validation loss, confirming the better performance of the ΔFLQ algorithm w.r.t. the FLQ algorithm reported in Tables 4 and 5 even in the presence of faults. As expected, as the number of faults increases, the minimum validation loss increases, since the dataset portions of the faulty edge servers no longer contribute to the accuracy of the global model learned via federated averaging. Moreover, the minimum validation loss increases with the number of edge servers also in the presence of faults, due to the smaller local dataset portions available to each server, confirming the experimental results illustrated in Figure 8.

To conclude on RQ1, we experimentally showed that:

- the FLQ algorithm with LowQ, ResQ, and IterQ quantization schemes is able to train an LSTM network with performances similar to a LSTM network trained in the LQ scenario. In particular, the LSTM network trained with IterQ in the FLQ

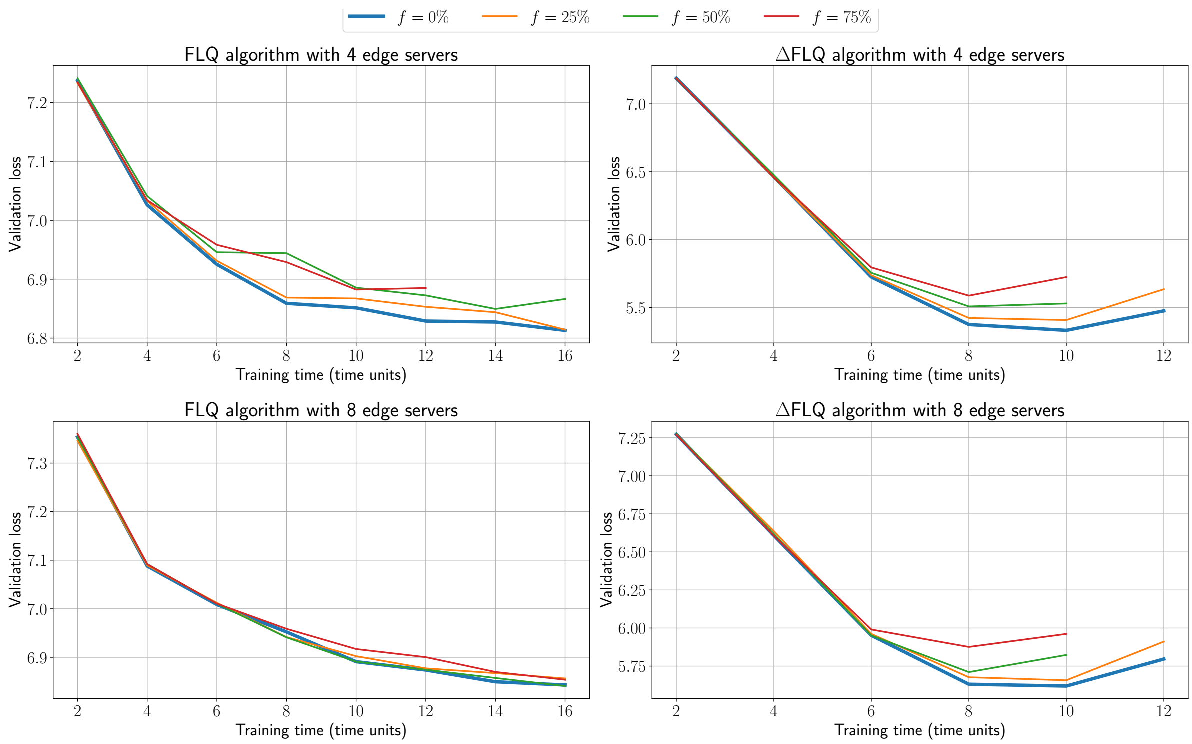

Figure 9: Validation loss of the LSTM network in the FLQ scenario using the FLQ (left) and ΔFLQ (right) algorithms with 2-bit IterQ quantization and $\tau = 16$, with 4 (top) and 8 (bottom) edge servers, with a fraction f of faults after the second federated round.

scenario outperforms the LSTM network in the LQ scenario with any quantization scheme for all τ in $\{2, 4, 8, 16\}$;

- the ΔFLQ algorithm with ResQ and IterQ allows to train an LSTM network with performances similar to an LSTM network trained in the LL scenario. Moreover, we show that IterQ allows for a faster convergence speed, thus allowing for a reduced number of model transmissions between edge servers and the cloud node, even in the presence of edge server faults.

5.4. Analysis of Model Transmission Costs

We now evaluate the reduction in terms of data transmitted between the edge servers and the cloud node by the FLQ/ΔFLQ algorithms w.r.t. the FL algorithm without quantization. Our LSTM model is composed of $27,249,264$ parameters in total. As reported in Section 4, we do not apply quantization to the input embedding layer, whose weight matrix contains $6,655,600$ elements. The other layers store $20,593,664$ model weights, which are represented according to our quantization schemes.

By storing each parameter as a 32-bit floating point number, the LSTM network requires 109.16 MB $\approx$ 0.11 GB, while the 1-bit random quantization schemes BinQ and ProbQ store the quantized LSTM network in 29.36 MB, with a space reduction of 3.71$\times$, but their performance in term of test loss are poor, as we reported in Table 3. The 2-bit and 3-bit quantization schemes, i.e., LowQ, ResQ, and IterQ, store the quantized LSTM network in 31.94 MB (with a space reduction of 3.42$\times$) and 34.51 MB (with a space reduction of 3.16$\times$), respectively.

Table 6: Rounds to reach the minimum validation loss with the FLQ and ΔFLQ algorithms, and total data exchanged between the cloud node and a single edge server (in GB) for the 2-bit quantization schemes.

τ	Rounds to minimum loss			Total data exchanged		
	LowQ	ResQ	IterQ	LowQ	ResQ	IterQ
			FLQ			
1	10	78	17	0.64	4.98	1.09
2	9	80	12	0.57	5.11	0.77
4	9	80	7	0.57	5.11	0.48
8	13	9	8	0.83	0.57	0.51
16	13	7	7	0.83	0.48	0.48
			ΔFLQ			
1	9	17	5	0.57	1.09	0.32
2	4	14	5	0.26	0.89	0.32
4	2	11	5	0.13	0.70	0.32
8	1	9	5	0.06	0.57	0.32
16	1	7	5	0.06	0.48	0.32

Table 6 reports the number of federated rounds required to reach the minimum validation loss with the FLQ and ΔFLQ algorithms, for the 2-bit LowQ, ResQ, and IterQ quantization schemes. This number represents the optimal number of rounds to obtain the best LSTM network performance during training according to the validation loss, i.e., before the validation loss starts increasing and the local models start overfitting on the local data. The number of rounds T determines how many times the LSTM model is transferred from an edge server to the cloud node and vice-versa. Regarding the FLQ algorithm, LowQ is the best quantization scheme in terms of data exchanged when $\tau \in \{1, 2\}$. For larger τ values, the best scheme is IterQ, requiring almost half of the rounds required by LowQ. As we have shown in Figure 5 and Table 4, in such cases IterQ produces a LSTM network with better performance than the LSTM produced by LowQ. With $\tau = 16$, FLQ with IterQ quantization attains a validation loss of 6.83 in 7 rounds (56 time units) with a total of 0.48 GB $\times\ 2 = 0.96$ GB transferred (taking into account both workers), while FL attains a validation loss of 4.81 in 28 rounds (28 time units) with a total of 0.11 GB $\times\ 2 \times 2 \times 28 = 12.23$ GB transferred, by taking into account the full model size, the number of workers, the two-way transmissions and the number of rounds.

Regarding ΔFLQ, LowQ still requires very few rounds to converge, but, as reported in Figure 7 and Table 5, the performance of the resulting LSTM network is far from being competitive. However, IterQ is always producing a LSTM network with performance close to the FL algorithm without quantization in very few rounds, namely 5, with just 0.32 GB of data transferred. With $\tau = 16$, ΔFLQ with IterQ quantization gets a validation loss of 5.05 in 5 rounds (20 time units) with a total of 0.32 GB $\times\ 2 = 0.64$ GB transferred (taking into account both workers).

To conclude on RQ2, we experimentally showed that both the FLQ and ΔFLQ algorithms are able to largely reduce the data transferred between the cloud node and the edge nodes in a federated learning scenario. In particular, when used with the IterQ quantization scheme and using 16 epochs for local training at each worker, the FLQ algorithm trains the LSTM model with a 14% degradation in the validation loss while reducing by a factor of 13× the total data transmitted over the network during federated learning. The ΔFLQ algorithm trains the LSTM model with just a 5% degradation in the validation loss while reducing by a factor of 19× the total data transmitted over the network during federated learning.

6. Additional Experiments

In this section, we conduct additional experiments to assess the performance of the FLQ and ΔFLQ algorithms on different datasets, namely image (Section 5.4) and sensor data (Section 5.4). All experiments are conducted on the same experimental framework detailed in Section 4. Given the results reported in Section 5, we limit our experiments to 2-bit IterQ quantization scheme and 2 edge servers, because IterQ yielded the best performance among the quantization methods considered.

6.1. Experiments on the MNIST image dataset

We conduct experiments on the popular MNIST dataset. The dataset consists of a training set of $60,000$ and a test set of $10,000$ 28×28 gray-scale images. We randomly sample the 10% of the training set to be used as the validation set during training. The remaining training set is uniformly split among the edge servers. We use a CNN composed of two 5×5 convolutional layers, followed by two linear layers. We choose rectified linear units as activation functions, and we use 2×2 max-pooling layers and two dropout layers. Training is performed using a batch size of 10 and stochastic gradient descent with a learning rate of 0.01 and momentum 0.5. The training of the CNN is performed by minimizing the cross-entropy loss until the learning rate decreases below the minimum value of 10^{-4} (early-stopping condition) or for a maximum of 80 federated learning rounds.

Table 7: Minimum validation, test losses, rounds to reach the minimum validation loss and the total data (both directions) exchanged between the cloud node and a single edge server (in MB) of the CNN network model in the FL and FLQ scenario using the FLQ and ΔFLQ algorithms with 2-bit quantization scheme and 2 workers, by varying τ.

τ	Min. val. loss			Test loss			Rounds			Total data		
	FL	FLQ	ΔFLQ	FL	FLQ	ΔFLQ	FL	FLQ	ΔFLQ	FL	FLQ	ΔFLQ
1	0.0509	0.0596	0.0638	0.0351	0.0453	0.0460	15	16	11	23.10	2.00	1.37
2	0.0412	0.0461	0.0502	0.0357	0.0408	0.0400	14	13	11	21.56	1.62	1.37
4	0.0451	0.0482	0.0507	0.0353	0.0370	0.0357	10	10	7	15.40	1.25	0.87
8	0.0451	0.0441	0.0499	0.0320	0.0329	0.0340	8	8	7	12.32	1.00	0.87
16	0.0428	0.0429	0.0495	0.0308	0.0333	0.0381	5	7	5	7.70	0.87	0.62

Our CNN model is composed of $24,090$ parameters in total. We do not apply quantization to the 90 one-dimensional bias parameters. The 4D tensors of the convolutional layers have been decomposed into matrices along the first two dimensions and each matrix has been quantized independently. The uncompressed CNN model requires 770.88 KB $\approx$ 0.77 MB, while the 2-bit IterQ quantization scheme stores the network in 62.40 KB $\approx$ 0.06 MB, with a space reduction of 12.35$\times$. This higher space reduction w.r.t. the LSTM model investigated in Section 5 takes into account the larger weight matrices in the fully connected layers of the CNN model.

Table 7 reports the validation and test losses with the FLQ and ΔFLQ algorithms. Comparing the corresponding values with the validation and test losses in the FL scenario, where no quantization is employed, the networks, trained by the two algorithms for different τ values, exhibit slightly worse performance. Moreover, Table 7 reports the number of federated rounds required to reach the minimum validation loss with the FLQ and ΔFLQ algorithms. Recall that this number represents the optimal number of rounds to obtain the best CNN network performance during training before the validation loss starts increasing and the local models start overfitting on the local data. The number of rounds determines how many times the CNN model is transferred from an edge server to the cloud node and vice-versa. The table also reports the number of optimal rounds in the FL scenario.

In the FL scenario, the best validation loss is 0.0428, obtained with $\tau = 16$ and with 5 federated learning rounds, with a total of 7.70 MB $\times$ 2 = 15.40 MB transferred (taking into account both edge servers). When the 2-bit IterQ quantization scheme is adopted, the best validation losses are 0.0429 (in 7 federated rounds) and 0.0495 (in 5 federated rounds) for the FLQ and ΔFLQ algorithms, respectively, obtained with $\tau = 16$. Hence, FLQ transfers a total of 1.74 MB, while ΔFLQ a total of 1.24 MB over the network, taking into account both edge servers.

Finally, when used with the 2-bit IterQ quantization scheme and using 16 epochs for local training at each worker, the FLQ algorithm trains the CNN model with almost no degradation in the validation loss while reducing by a factor of 8.8$\times$ the total data transmitted over the network during federated learning. The ΔFLQ algorithm trains the CNN model with a 15% degradation in the validation loss while reducing by a factor of

12.4× the total data transmitted over the network during federated learning.

6.2. Experiments on the Bar Crawl dataset

We perform additional experiments on time series data by employing the Bar Crawl dataset[5]. The dataset collects acquisitions from the accelerometer and the transdermal alcohol content from a college bar crawl and it is used in literature to predict heavy drinking episodes via mobile data [38]. For our experiments, we employ the raw data collected from smartphones' accelerometers at a sampling rate of 40Hz. We choose to make use of this dataset because of its size (i.e., several millions of observations collected). In detail, each data point of the accelerometer's observations contains five fields, i.e., the timestamp, a participant ID, and a sample from each of the three axes of the accelerometer. Data was collected from a mix of 11 iPhones and 2 Android phones. The whole dataset is fully anonymized.

We preprocess the data by deriving, from each acquisition, the average acceleration. We then perform a per-user min-max standardization to normalize the data in the range $[0, 1]$. We employ the derived time series to learn a model that predicts the "next" acceleration value, i.e., given the sequence of previous N acquisitions we query the model to predict the $N + 1$ value. We model this task as a regression problem and we employ an LSTM to solve it. The employed LSTM is made up of one layer with 2 hidden nodes and one output node. We train the network by employing the Mean Squared Error loss function and by using a window of $N = 16$ previous acquisitions. We split the dataset in train/validation/test set on a per-user basis, i.e., we employ 8 users as the training set, 2 users as the validation set and 3 users as test set. The preprocessed dataset divided in train/validation/test is made available to allow for reproducibility of the results[6]. The size of the model is $1,376$ bytes for the non-compressed model, while the 2-bit IterQ quantization scheme stores the network in $1,096$ bytes. Differently from the previous experiments, we train the network for a maximum of 40 federated learning rounds with a fixed learning rate. Early-stopping is performed when no improvement in the validation

[5] `https://archive.ics.uci.edu/ml/datasets/Bar+Crawl:+Detecting+Heavy+Drinking`

[6] `http://hpc.isti.cnr.it/~nardini/datasets/data_timeseries.tar.gz`

loss is observed for 5 local training epochs. We also employ a batch size of 256 training samples and a learning rate of 0.01.

Table 8: Minimum validation, test losses, rounds to reach the minimum validation loss and the total data (both directions) exchanged between the cloud node and a single edge server (in KB) of the LSTM network model in the FL and FLQ scenario using the FLQ and ΔFLQ algorithms with 2-bit quantization scheme and 2 workers, by varying τ.

τ	Min. val. loss			Test loss			Rounds			Total data		
	FL	FLQ	ΔFLQ	FL	FLQ	ΔFLQ	FL	FLQ	ΔFLQ	FL	FLQ	ΔFLQ
1	0.0784	0.0956	0.1576	0.107	0.112	0.160	40	40	40	110.08	87.68	87.68
2	0.0711	0.0759	0.1550	0.101	0.103	0.151	40	40	22	110.08	87.68	48.22
4	0.0691	0.0728	0.1489	0.091	0.098	0.150	40	40	7	110.08	87.68	15.34
8	0.0690	0.0714	0.1191	0.089	0.092	0.121	24	38	4	66.04	83.29	8.76
16	0.0687	0.0701	0.0841	0.086	0.089	0.099	36	34	3	99.07	73.16	6.45

Table 8 reports the validation and test losses with the FL, FLQ,and ΔFLQ algorithms. Comparing the corresponding values with the validation and test losses, in the FL scenario, where no quantization is employed, the networks, trained by the two algorithms for different τ values, exhibit slightly worse performance. Table 8 also reports the number of federated rounds required to reach the minimum validation loss with the FL, FLQ, and ΔFLQ algorithms. We recall that this number represents the optimal number of rounds to obtain the best LSTM network performance during training before the local models start overfitting on the local data and the validation loss starts increasing. The number of rounds determines how many times the LSTM model is transferred from an edge server to the cloud node and vice-versa. The table also reports the number of optimal rounds in the FL scenario.

In the FL scenario, the best validation loss is 0.0687, obtained with $\tau = 16$ and with 36 federated rounds, with a total of 99.07 KB $\times\ 2 = 198.14$ KB transferred (taking into account both edge servers). The use of the 2-bit IterQ quantization scheme allows us to achieve 0.0701 minimum validation loss (in 36 federated rounds) and 0.0841 minimum validation loss (in 3 federated rounds) for the FLQ and ΔFLQ algorithms, respectively, obtained with $\tau = 16$. Hence, FLQ transfers a total of 146.32 KB, while ΔFLQ transmits a total of 12.90 KB over the network, taking into account both edge servers.

To conclude, the 2-bit IterQ quantization scheme allows to effectively train an LSTM model for the "next value" prediction task on time series data. In detail, when using 16 epochs for local training at each edge server, the FLQ algorithm trains the LSTM model with no degradation in the validation loss while reducing by 25% the total data transmitted over the network during federated learning, and the ΔFLQ algorithm trains the LSTM model with a 20% degradation in the validation loss while reducing by a factor of $16\times$ the total data transmitted over the network during federated learning.

7. Conclusion

In this work, we presented a federated learning platform for the IoT built upon an edge computing framework and according to a publish/subscribe communication paradigm. We introduced the edge computing layer to let the neural network reside as close as possible to the IoT data producers and within the privacy domain of IoT device owners. Data privacy has been assumed by design as a mandatory requirement, leading to data retention on the edge. We developed two novel federated learning quantization algorithms, namely FLQ and ΔFLQ. The FLQ algorithm introduces quantization into the state-of-the-art FL algorithm. Instead, ΔFLQ is designed from scratch to reduce the amount of traffic exchanged among edge servers and the Cloud. Differently from [23], our FLQ and ΔFLQ algorithms quantize both the NN model broadcasted by the cloud node and the NN models sent by the edge servers to significantly reduce the background traffic related to the NN model training. The results achieved by ΔFLQ outperform those of FLQ both in terms of validation loss and efficiency in data exchange and show good tolerance to server faults. We conducted experiments on public datasets by training a Long Short Term Memory neural network on an edge node to solve a next word prediction task as a case study. Only the federation task is left to the Cloud, or, eventually, to a centralized entity, not necessarily located in the core network. To validate the generality of the proposed algorithms, we also applied both ΔFLQ and FLQ to different use cases and other deep learning models. In detail, we experimented our proposed algorithms with CNNs on image classification tasks using the popular MNIST dataset and with LSTMs on next value prediction using time series from the Bar Crawl dataset. In both cases, the ΔFLQ was able to reduce by $8-16\times$ the total amount of traffic among edge servers and the Cloud. Last but not least, the proposed federated learning approach allows for reducing the time to reach the minimum validation loss w.r.t. a centralized Cloud approach, with a negligible exchange of data at each training round, thanks to ΔFLQ and FLQ algorithms. The comprehensive experimental evaluation compares our approach to state-of-the-art centralized learning, i.e., on the Cloud, namely local learning, and federated learning techniques employing both full precision and several quantization schemes. We remark that local learning violates the requirement of data privacy. Still, it stands here as the optimal performance metric of comparison, in terms of validation loss, when the whole dataset is centralized. The measurement campaign shows that the introduction of quantization techniques in federated learning allows to significantly reduce the data exchanged between each edge server and a cloud node, providing a measured improvement with a minimal impact on the validation loss of the final model. In detail, FLQ outperforms full-precision federated learning of an LSTM for next-word prediction by reducing by a factor of 13$\times$ the total data transmitted over the network, but leads to an increase in validation loss by 14%. On the other hand, ΔFLQ can further reduce the total data transmitted up to 19$\times$, with a validation loss increase of less than 5%. Subsection 5.4 also provides a numerical quantification of the effective reduction in data volume exchanged with the 2- and 3-bit quantization schemes. Such promising results pave the way for deepening research on federated learning quantization, exploring how different schemes and techniques could impact the efficiency of the federated learning process.

Future investigations will also address more in-depth communication models taking into account 5G mobility scenarios and investigate the fault-tolerance of federated learning architectures in real emerging contexts.

Acknowledgements

This work is partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence), by the BIGDATAGRAPES and the TEACHING projects funded by the EU Horizon 2020 research and innovation program under grant agreements No. 780751 and No. 871385, respectively, and by the OK-INSAID project funded by the Italian Ministry of Education and Research (MIUR) under grant agreement No. ARS01_00917. This work is partially carried out in the framework of the project AUTENS (Sustainable Energy Autarky) funded by the University of Pisa (PRA 2020 program).

References

[1] Viraaji Mothukuri, Reza M. Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[2] Miguel De Prado, Jing Su, Rabia Saeed, Lorenzo Keller, Noelia Vallez, Andrew Anderson, David Gregg, Luca Benini, Tim Llewellynn, Nabil Ouerhani, Rozenn Dahyot, and Nuria Pazos. Bonseyes AI Pipeline – Bringing AI to You: End-to-End Integration of Data, Algorithms, and Deployment Tools. *ACM Transaction on Internet of Things*, 1(4), 2020.

[3] Farzad Samie, Lars Bauer, and Jörg Henkel. From cloud down to things: An overview of machine learning in internet of things. *IEEE Internet of Things Journal*, 6(3):4921–4934, 2019.

[4] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[5] Weishan Zhang, Wuwu Guo, Xin Liu, Yan Liu, Jiehan Zhou, Bo Li, Qinghua Lu, and Su Yang. LSTM-based analysis of industrial IoT equipment. *IEEE Access*, 6:23551–23560, 2018.

[6] Jed Mills, Jia Hu, and Geyong Min. Communication-efficient federated learning for wireless edge intelligence in iot. *IEEE Internet of Things Journal*, 7(7):5986–5994, 2019.

[7] Peter Kairouz and et al. Advances and open problems in federated learning. *ArXiv*, abs/1912.04977, 2019.

[8] He Li, Kaoru Ota, and Mianxiong Dong. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Network*, 32(1):96–101, 2018.

[9] Jie Tang, Dawei Sun, Shaoshan Liu, and Jean-Luc Gaudiot. Enabling deep learning on IoT devices. *Computer*, 50(10):92–96, 2017.

[10] Ahmed Imteaj and M. Hadi Amini. Distributed sensing using smart end-user devices: Pathway to federated learning for autonomous iot. In *Proc. CSCI*, pages 1156–1161, 2019.

[11] Felix A. Gers, Douglas Eck, and Jürgen Schmidhuber. Applying LSTM to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.

[12] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for Natural Language Inference. In *Proc. ACL*, pages 1657–1668, 2017.

[13] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. NIPS*, pages 1097–1105, USA, 2012.

[15] Russell Reed. Pruning algorithms-a survey. *IEEE transactions on Neural Networks*, 4(5):740–747, 1993.

[16] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv e-prints*, page arXiv:1606.06160, 2016.

[17] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[18] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proc. ICML*, pages 1737–1746, 2015.

[19] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proc. ECCV*, pages 525–542. Springer, 2016.

[20] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proc. NIPS*, pages 4107–4115, 2016.

[21] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. In *Proc. ICLR*, 2017.

[22] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *Proc. ICLR*, 2017.

[23] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. Alternating multi-bit quantization for recurrent neural networks. In *Proc. ICLR*, 2018.

[24] Arash Ardakani, Zhengyun Ji, Sean C Smithson, Brett H Meyer, and Warren J Gross. Learning recurrent binary/ternary weights. In *Proc. ICLR*, 2019.

[25] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. AISTATS*, 2017.

[26] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[27] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *Proc. PMLR*, pages 2021–2031, 2020.

[28] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Computing Surveys (CSUR)*, 51(6):1–29, 2019.

[29] Manlio Bacco, Marco Colucci, and Alberto Gotta. Application protocols enabling internet of remote

things via random access satellite channels. In *Proc. ICC*, pages 1–6, 2017.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[31] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proc. NIPS*, pages 3123–3131, 2015.

[32] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proc. ECCV*, 2016.

[33] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *Proc. NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[34] Dan Alistarh, Demjan Grubic, Jerry Z. Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Proc. NIPS*, page 1707–1718, 2017.

[35] Geoffrey Davis, Stephane G. Mallet, and Marco Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, 1997.

[36] Yiwen Guo, Anbang Yao, Hao Zhao, and Yurong Chen. Network sketching: Exploiting binary structure in deep cnns. In *Proc. CVPR*, pages 4040–4048, 2017.

[37] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[38] Jackson A. Killian, Kevin M. Passino, Arnab Nandi, Danielle R. Madden, John D. Clapp, Nirmalie Wiratunga, Frans Coenen, and Sadiq Sani. Learning to detect heavy drinking episodes using smartphone accelerometer data. In *KHD@ IJCAI*, pages 35–42, 2019.