Received 29 November 2023, accepted 14 December 2023, date of publication 4 January 2024, date of current version 12 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3349557

APPLIED RESEARCH

The Effects of Weight Quantization on Online Federated Learning for the IoT: A Case Study

NIL LLISTERRI GIMÉNEZ[1], JUNKYU LEE[2], FELIX FREITAG[1], AND HANS VANDIERENDONCK[3], (Senior Member, IEEE)
[1]Department of Computer Architecture, Technical University of Catalonia, 08034 Barcelona, Spain
[2]Institute for Analytics and Data Science, University of Essex, CO4 3SQ Colchester, U.K.
[3]Institute of Electronics, Communications and Information Technology, Queen's University Belfast, BT3 9DT Belfast, U.K.

Corresponding author: Felix Freitag (felix.freitag@upc.edu)

This work was supported in part by the European Union's Horizon 2020 Research and Innovation Program (ASSIST-IoT) under Grant 957258, in part by the Spanish Government (DiPET CHIST-ERA) under Contract PID2019-106774RB-C21 and Contract PCI2019-111850-2, and in part by the Generalitat de Catalunya as Consolidated Research Group under Grant 2021-SGR-01059.

ABSTRACT Many weight quantization approaches were explored to save the communication bandwidth between the clients and the server in federated learning using high-end computing machines. However, there is a lack of weight quantization research for online federated learning using TinyML devices which are restricted by the mini-batch size, the neural network size, and the communication method due to their severe hardware resource constraints and power budgets. We name Tiny Online Federated Learning (TinyOFL) for online federated learning using TinyML devices in the Internet of Things (IoT). This paper performs a comprehensive analysis of the effects of weight quantization in TinyOFL in terms of accuracy, stability, overfitting, communication efficiency, energy consumption, and delivery time, and extracts practical guidelines on how to apply the weight quantization to TinyOFL. Our analysis is supported by a TinyOFL case study with three Arduino Portenta H7 boards running federated learning clients for a keyword spotting task. Our findings include that in TinyOFL, a more aggressive weight quantization can be allowed than in online learning without FL, without affecting the accuracy thanks to TinyOFL's quasi-batch training property. For example, using 7-bit weights achieved the equivalent accuracy to 32-bit floating point weights, while saving communication bandwidth by 4.6×. Overfitting by increasing network width rarely occurs in TinyOFL, but may occur if strong weight quantization is applied. The experiments also showed that there is a design space for TinyOFL applications by compensating for the accuracy loss due to weight quantization with an increase of the neural network size.

INDEX TERMS TinyML, approximate computing, federated learning, IoT.

I. INTRODUCTION

Federated Learning (FL) is a collaborative training approach for a machine learning model with distributed nodes to deal with massive training data efficiently and address data privacy [1]. Specifically, Internet of Things (IoT) devices with sensors are a source of data that can be used for training. On such memory-constrained devices Online Learning (OL) can be used to train the models sequentially as new samples get captured. FL utilizing OL is referred to as online FL [2]. Particularly, we name Tiny Online FL (TinyOFL) for online FL using TinyML devices in the IoT. Likewise, Tiny Online Learning (TinyOL) [3] is referred to as online learning using TinyML devices without any federated learning scheme in this paper. Using machine learning models on IoT devices has been shown by several recent works regarding anomaly detection [4], continuous learning [3], and transfer learning [5].

Different aspects of heterogeneity in the IoT make FL challenging for being used in embedded devices [6]: 1) the computing capacity of the IoT nodes can be different either due to the different hardware, because of concurrent

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Quan.

executions of other tasks at the node, or due to node-specific energy budgets, 2) IoT nodes obtain data from the proper sensors, but the quantity and quality of the local data and their distribution at each node may vary, for instance, due to the local circumstances the nodes may obtain a different number of training samples, and 3) for the communication of the trained model with the aggregator, some nodes may face limitations due to low network bandwidth or lack of energy.

For example, in the case of the first and second challenge, if certain clients are selected more frequently according to node-specific energy budgets and communication bandwidth constraints, their local updates have a stronger influence on the global model. Consequently, the global model may become more biased towards the data distribution of those frequently selected clients. Therefore, it is highly likely that higher accuracy is expected on their specific data patterns, but potentially lower accuracy on the data patterns of less frequently selected clients. The final trained model may reflect the bias in the client selection process in TinyOFL.

This paper addresses the third challenge for enabling FL in the IoT by an understanding of the effects of the weight quantization on TinyOFL in terms of generalization ability, communication efficiency, and energy efficiency. The scenario targeted is IoT devices at remote locations with machine learning applications that use Low Power, Wide Area Network (LPWAN) communication technologies [7]. The wireless links of these LPWANs have very low data rates compared to the Internet. In rural areas such links can cover several kilometers of distance. The communication has a limited power budget since the power supply from batteries or solar power can be unstable.

LoRa is one of the LPWAN for the IoT [8]. Compared to other long range technologies such as NB-IoT and Sigfox, LoRa is operated in the unlicensed band and does not require any license fees or network operator [9]. In comparison with Sigfox, LoRa allows a higher payload size. A LoRa packet can have a size of up to 255B. However, in many countries, the duty cycle of LoRa communication is limited to 1%, which limits the data rate that can be achieved. The node-to-node communication with LoRa between geographically spread IoT nodes is enabled by LoRa mesh networks [10]. However, sending the least number of packets in a LoRa communication is, in general, desirable for reducing packet collisions and latency. The application of the proposed TinyOFL for training Machine Learning (ML) models at such LoRa-interconnected nodes would greatly benefit from the quantized model communication.

FL with IoT devices implies that an ML model is trained on-device, i.e., on the node itself [11]. However, among the options to equip a microcontroller-based IoT device with a trained model, on-device training is nowadays still a niche approach, though recently attracting a significant amount of attention. The dominating method today is training an ML model off-device on a powerful computer or with a cloud service. The advantage of that approach is that enough computing power and data storage is available to train the model with a large dataset up to the required accuracy [12]. Before flashing the model to the IoT device, tools such as Tensorflow Light Micro[1] optimize the model size for fitting into the small memory of a microcontroller board. This external training achieves high performing models, but it does not allow the model to be updated later since its values are stored in the board's flash memory where the weights cannot be changed.

If the IoT device is already deployed, other arrangements of the application code can allow updating a node with a new machine learning model over-the-air, as proposed in [13]. While this approach allows a node to obtain a new model and hence change its function to a different task, the model is still trained off-device and with external data. The sensor data that the node obtains is not exploited for model training but only for inference.

Training a machine learning model on the IoT device is an emerging option to make the embedded learning in the IoT adaptive, e.g., to the concept drift of data or for retraining a model with the data that is generated by the sensors at the device. A recent survey by Rajapakse et al. reviews the growing number of works that studied changing ML models at IoT boards [14]. In on-device training, a model can be trained on a node in an isolated fashion with the local data it collects, without requiring any network interface, or, if a network interface exists to connect to other nodes with the same machine learning task, then the model could principally be trained with FL.

The impact of weight quantization on the accuracy and energy consumption was empirically explored with MNIST using LeNet, SVHN using a shallow CNN, and CIFAR-10 using ALEXnet [15]. However, the generalization ability was not described for these machine learning frameworks. We notice that different behaviors can be expected between conventional ML and tiny ML, given the generalization ability described for well-known ML framework with regards to bias-variance tradeoff, overfitting, and regularization according to the target complexity, model complexity, mini-batch size, and the training data quality and quantity. For instance, in TinyOFL the restricted memory of an IoT board may require training with a mini-batch size of one, while Conventional Federated Learning (CFL) can utilize much larger mini-batch sizes. Due to the inherent limitations in memory resources on TinyML devices, we opt for an online learning approach employing a local mini-batch size of '1'. In this scenario, FedSGD, which averages gradients, necessitates communication between the server and clients for every local sample, resulting in significant training overhead. To overcome this limitation, we choose FedAVG, which averages weight parameters, to minimize the communication cost in proportion to the number of local updates [16].

In this paper, we aim to make a step forward in the applicability of FL in the IoT, specifically for tiny resource-

[1] https://www.tensorflow.org/lite/microcontrollers

constraint IoT nodes with a low communication capacity. For this, we explore the effects of weight quantization on TinyOFL with a flexible model weight quantization communication scheme that allows the client and the server in FL to flexibly choose the number of bits used for communicating the model weights. In flexible model weight quantization communication FL, the clients and server do the local computation with full precision, but the model communication is done with quantized weights. In practice, quantification is implemented as a component of the code that runs on an IoT device. It is active when the nodes do TinyOFL.[2] The client running on the node can choose for every FL round the number of bits for the model transmission to optimize its communication according to its link and energy budget.

The main contributions of this paper include the new findings regarding the effects of weight quantization on TinyOFL as follows:

- TinyOFL allows weight quantization to be more aggressive than TinyOL. E.g., 5-bit weights reach 74% accuracy with FL, but 50% accuracy without FL.
- Overfitting rarely occurs in TinyOFL by weight quantization due to its relatively restricted model capacity.
- In TinyOFL, increasing the model size by leveraging the communication saving from quantization does not generate overfitting but improves the accuracy in general, unlike CFL. There is a sweet spot between the quantization bit-width and the increased model size for saving power and communication bandwidth while keeping the accuracy equivalent to the full-precision model.
- We evaluate the effects of weight quantization on TinyOFL in terms of accuracy, overfitting, power, and transmission time, with a keyword spotting application.

II. DESIGN

A. FEDERATED LEARNING OVERVIEW

FL is a collaborative training method that utilizes multiple clients in parallel to deal with massive training data efficiently in a privacy-preserving manner. First, an ML model is shared among n_K client devices [17]. The training data are split into n_K subsets, and each subset of the training data is assigned to each client. The weights of the model on each client are trained separately using its own local training data, given a mini-batch size and the number of Local Update Round (LUR)s. The trained weights on each device are sent to a centralized component called a server or aggregator. Once this server receives the trained weights of all n_K clients, a new model is created by aggregating the weights using the weighted average. By using this aggregation method the resulting model has better accuracy when the data is unevenly distributed between the clients, by giving more importance to the weights of the models that have went through more training rounds. The weights of the resulting model are sent back to the clients for the next training round. The above procedure is repetitively performed until the accuracy is saturated. FL parameters also include the number of total training samples n_{TR}, the number of local training samples $n_{tr}(= n_{TR}/n_K)$, the number of LURs n_l, the number of global FL updates n_R, and the number of test samples n_{TE}.

[2]Our implementation and the dataset used for the experimentation is available at the git repository https://github.com/NilLlisterri/TinyML-FederatedLearning/tree/mixed-precision

B. WEIGHT QUANTIZATION

We apply uniform quantization to the weights [18]. At each client we obtain for its locally trained Neural Network (NN) model the values of the maximum and minimum weights, w_max and w_min, given as a 4 byte float data type. Our weight quantization transforms the weight values w to the quantized values w' in the range of an int data type between w_max and w_min according to a determined width l of bits. For this width, b is the higher value of the range and a is the lower value of the range that can be represented as int by the number of bits. The transformation from the weights w to w' at the local node applies the equation:

$$w' = round(\frac{(w - w\_min)(b - a)}{w\_max - w\_min}) \tag{1}$$

At the server, w' is transformed back into an approximate w_a by applying the equation:

$$wa = w\_min + \frac{(w' - a)(w\_max - w\_min)}{b - a} \tag{2}$$

The dequanitzed weights w_a have an error p with regards to w, which is determined by the range of w and the width l of w'.

$$p = \frac{1}{2}\frac{(w\_max - w\_min)}{2^l} \tag{3}$$

Figure 1 shows an example of transforming the weights w into w' with 8 bits. In this example, the weights at the local node have a minimum a maximum weight of -0.5 and 0.5, respectively. The integer representation with 8 bits has 256 values for representing w by w' on a range of $a = 0$ and $b = 255$. By the transformation of w into w', there is a loss of precision. In the example, the error with which wa represents w is 0.00195. The approach is similar to the block Floating Point (FP) arithmetic approach [19] in that all quantized weights share an exponent according to their magnitude distribution.

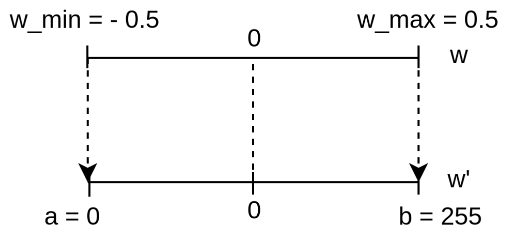

FIGURE 1. Example of applying weight transformation to an 8 bits integer.

Figure 2 illustrates how the quantization is applied in TinyOFL. On the top, N client nodes transform their local weights of 4 byte float datatype into 8 bit integers. Each client sends the metadata about the transformation it used and the converted weights to the server. The metadata is needed by the server to transform back the received weights w' into weights wa of 4 byte floats. This metadata must include the specific w_max and w_min. The width l with which the transformed weights are represented needs to be transmitted as well if it is not statically defined by configuration. FedAVG [16] is applied at the server to obtain the weights wg of the global model. Only for the model communication between clients and server, the weights are quantized. At the server, the dequanitzation is performed by transforming the 8 bit integers received into 4 byte float datatype weights. The full precision, i.e., IEEE single precision, is utilized for the weights wa and wg, albeit the dequantized weights at the server that were transformed have a loss of precision. Once the new global model is calculated, eq. (1) is applied at the server on wg with the corresponding minimum and maxiumum weights to obtain wg', and the clients apply eq. (2) on wg' to obtain wga.

Algorithm 1 shows how to quantize the weights in the FL clients and send them. A client first needs to obtain the number of bits to represent the quantized weights for the communication. The transformation is flexible with regard to the width of the number of bits per weight. This allows an FL client either to use a statically configured value or adapt this parameter to the individual resource availability, such as bandwidth and energy evaluated at every FL round.

Algorithm 1 Algorithm for Sending Quantized Weights From a Client

```
Require: training = false
  $l$ = getWidth()          ▷ e.g. by configuration or dynamically
  $w\_max$ = getMax($weights$)
  $w\_min$ = getMin($weights$)
  $a = 0$
  $b = 2^l - 1$
  sendMetaData($l, w\_max, w\_min$)
  for all weights do
      $w' \leftarrow \frac{(w - w\_min)(b-a)}{w\_max - w\_min}$
      sendQuantizedWeight($w'$)
  end for
```

The server receives from each federated client the metadata that specifies how the weights were quantized. Every client can send its weights with a different number of bits, therefore the server has to obtain the metadata for every client and transform the received weights accordingly. Principally, the server could also send the global model back to the clients with a client-specific quantization. For this, the client could add a flag to the metadata message indicating if the server should maintain the quantization of the client or if it can be changed. Compared to the number of weights of a neural network that are sent, the overhead of sending the metadata is negligible.

III. EFFECTS OF WEIGHT QUANTIZATION ON TINYOFL

The limited compute resources of TinyML devices restrict their applications, model capacity, training setting requirements, and accuracy properties, which is different from CFL. The memory availability on these microcontroller-based boards impedes the storage of data samples and prevents a model from being trained with a mini-batch size greater than one. Such online FL in tiny devices, which we call TinyOFL in this paper to distinguish it from the other FL scenarios, has the specific condition that the data samples obtained from the on-board sensors can be used for training only once, then the sample is deleted and replaced by the next sample from the sensor. If TinyOFL is done in real conditions, the lack of storage of the boards does not allow to work with a training and testing data set as in CFL. Data samples for training in TinyOFL are always new and are never repeated [20], which restricts the model training compared to CFL. Having this mini-batch size of one specific to TinyOFL, we will explore the effects of weight quantization for this situation.

A. WEIGHT QUANTIZATION FOR ONLINE FL

Online learning updates the global weights for every training sample, while online FL updates the global weights for every $n_{tr} \times n_K$ samples. The weight quantization in online FL quantizes the weights for local training on each client according to the mini-batch size assigned to the clients, and the quantized weights trained on each client are sent to the server. Then, the weights received from each client are averaged in the server with full precision arithmetic, and the averaged weights are quantized again and sent back to the clients. Therefore, the weights are updated per the number of training samples (i.e., $n_{tr} \times n_K$ samples) equal to the number of clients multiplied by a local mini-batch size in online FL. On the contrary, the weight quantization without FL utilizes the quantized weights in updating the weights per training sample. The noise effects on the accuracy due to the weight quantization can be stochastically diminished in the weight updates by averaging the weights in FL, while the noise effects are not diminished in the weight updates without FL. Therefore, the weight quantization can be more beneficial with the FL framework.

Eq. (4) represents each local weight update in FL using $n_t r$ training samples.

$$\mathbf{w}_{t,i} = \mathbf{w}_{t-1,i} - \eta \Sigma_{j=1}^{n_{tr}} \nabla e(\mathbf{w}_{t-1,i}^{(j-1)}), \tag{4}$$

where η is a learning rate, $\nabla e(\mathbf{w}_{t,i}^{(j)})$ is an error gradient with respect to the local weights in the i-th client after the j-th local iteration, and:

$$\mathbf{w}_{t-1,i}^{(0)} = \mathbf{w}_{t-1,i}, \mathbf{w}_{t-1,i}^{(j)} = \mathbf{w}_{t-1,i}^{(j-1)} + \eta \nabla e(\mathbf{w}_{t-1,i}^{(j-1)}). \tag{5}$$

Each $\nabla e(\mathbf{w}_{t,i}^{(j)})$ is updated:

$$\nabla e(\mathbf{w}_{t,i}^{(j)}) = \frac{\partial e(\mathbf{w}_{t,i}^{(j)})}{\partial \mathbf{w}_{t,i}^{(j)}} \tag{6}$$

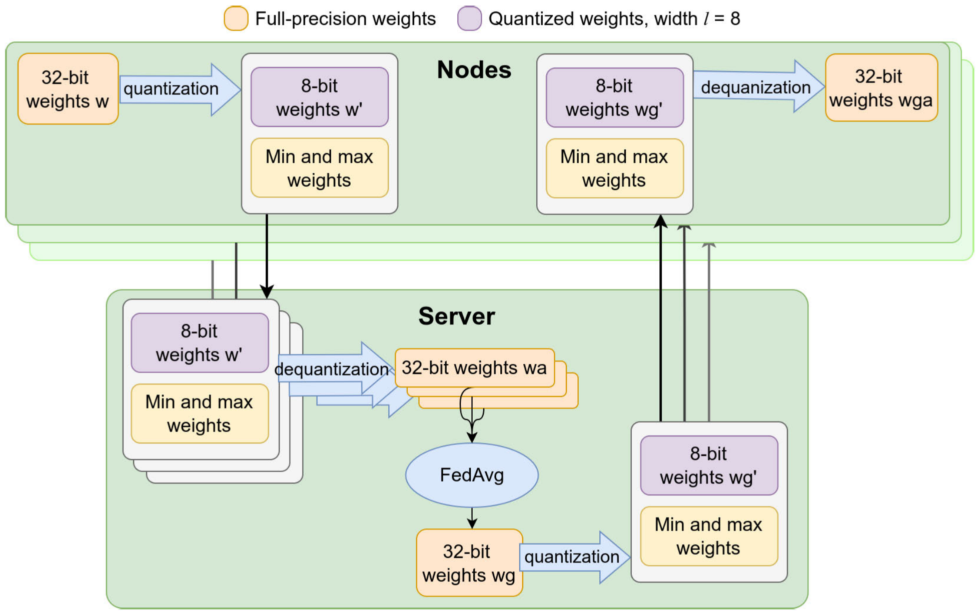

FIGURE 2. Applying quantization and dequantization for model communication in FL at the client and server side.

The FL updates the global weights $\mathbf{w}_t$ by averaging the local weights as follows.

$$\mathbf{w}_t = 1/n_K \Sigma_{i=1}^{n_K} \mathbf{w}_{t,i}. \tag{7}$$

For example, the $n_K = 1$ and $n_{tr} = 1$ for conventional online learning and $n_K \geq 2$ and $n_{tr} \geq 1$ for online FL. Therefore, the global weights for online learning are updated $n_K \times n_{tr}$ times more than online FL.

Based on Eq. (4) to (7), the online FL updates the weights as a *quasi-mini-batch method* as if the mini-batch size equals the number of devices K. Therefore, the stability of the online learning method can be enhanced by online FL, since the gradient direction per each weight update can be more accurate than online learning. The stability can be affected by the error caused by weight quantization. Therefore, the enhanced stability of online FL allows the weights to be quantized more aggressively than in online learning. The TinyOFL Property 1 (TP1) describes the effects of TinyOFL on weight quantization as follows.

TinyOFL Property 1 (Effects of TinyOFL on Weight Quantization): TinyOFL allows weight quantization to be more aggressive than online learning.

The sketch proof of TP1 is as follows. We compare the effects of quantization errors on the accuracy between online learning and online FL. Considering the weight quantization in Eq. (4), online FL applies the weight quantization after using n_{tr} samples, generating the weight quantization noise:

$$\begin{aligned}\tilde{\mathbf{w}}_{t,i} &= \mathbf{w}_{t-1,i} - \eta \Sigma_{j=1}^{n_{tr}} \nabla e(\tilde{\mathbf{w}}_{t-1,i}^{(j-1)} + \boldsymbol{\epsilon}_{t,i}^{(j-1)}) + \Sigma_{j=1}^{n_{tr}} \boldsymbol{\epsilon}_{t,i}^{(j)} \\ &= \mathbf{w}_{t-1,i} - \eta \Sigma_{j=1}^{n_{tr}} \nabla e(\mathbf{w}_{t-1,i}^{(j-1)}) + \Sigma_{j=1}^{n_{tr}} (\boldsymbol{\epsilon}_{t,i}^{(j)} + \boldsymbol{\delta}_{t,i}^{(j-1)}),\end{aligned} \tag{8}$$

where $\tilde{\mathbf{w}}_{t,i}$ represents the quantized weights for $\mathbf{w}_{t,i}$, $\boldsymbol{\epsilon}_{t,i}^{(j)}$ is a quantization error on the j^{th} local weight update and:

$$\boldsymbol{\delta}_{t,i}^{(j-1)} = \nabla e(\tilde{\mathbf{w}}_{t-1,i}^{(j-1)} + \boldsymbol{\epsilon}_{t,i}^{(j-1)}) - \nabla e(\mathbf{w}_{t-1,i}^{(j-1)}). \tag{9}$$

We may consider $\boldsymbol{\delta}_{t,i}^{(j-1)}$ as the deviation of the gradient due to the weight quantization. Considering the weight quantization for Eq. (5) yields:

$$\tilde{\mathbf{w}}_{t-1,i}^{(j)} = \tilde{\mathbf{w}}_{t-1,i}^{(j-1)} + \eta \nabla e(\tilde{\mathbf{w}}_{t-1,i}^{(j-1)}) + \boldsymbol{\epsilon}_{t,i}^{(j)}. \tag{10}$$

Therefore, the quantization errors $\boldsymbol{\epsilon}_{t,i}^{(j)}$ and the gradient deviation due to the weight quantization $\boldsymbol{\delta}_{t,i}^{(j-1)}$ are accumulated in the local updates before a global update. The two accumulated errors can be represented as:

$$\boldsymbol{\epsilon}_{t,i} = \Sigma_{j=1}^{n_{tr}} \boldsymbol{\epsilon}_{t,i}^{(j)}, \; \boldsymbol{\delta}_{t,i} = \Sigma_{j=1}^{n_{tr}} \boldsymbol{\delta}_{t,i}^{(j-1)}. \tag{11}$$

Considering the weight quantization errors for each client i in Eq. (7), the weight update averages the quantization errors across the K devices:

$$\tilde{\mathbf{w}}_t = 1/n_K \Sigma_{i=1}^{n_K} \mathbf{w}_{t,i} + 1/n_K \Sigma_{i=1}^{n_K} (\boldsymbol{\epsilon}_{t,i} + \boldsymbol{\delta}_{t,i}). \tag{12}$$

Based on Eq. (12), the effects of the weight quantization errors and the gradient deviation errors on the accuracy become diminished as the number of devices, n_K, increases thanks to the stochastic error cancellation property. Moreover, the weights proceeded with n_{tr} steps on each client are averaged, leading to more accurate weights. On the contrary, online learning does not average the weights and the quantization errors since it utilizes single client. Hence, without leveraging such stochastic error cancellation property, online learning is more susceptible to the weight quantization errors than online FL. This leads to TP1.

B. WEIGHT QUANTIZATION EFFECTS ON ACCURACY

The accuracy of TinyOFL is often limited due to its restricted model capacity. However, the accuracy is not affected by weight quantization if the quantization noise is relatively lower than the data noise [21].

The feature map using quantized weights, $\tilde{f}_i(\mathbf{x}_i)$, on the input feature $\mathbf{x}_i$ at the i-th layer can be deviated by $\delta\mathbf{y}_i$ due to the weight quantization as follows:

$$\tilde{f}_i(\mathbf{x}_i) = f_i(\mathbf{x}_i) + \delta\mathbf{y}_i, \tag{13}$$

where $f_i(\mathbf{x}_i)$ is the feature map operation on $\mathbf{x}_i$ using non-quantized weights. The $\tilde{f}_i(\mathbf{x}_i)$ generates the quantity $\delta\mathbf{q}_i$ based on the backward error analysis [21], [22]:

$$\tilde{f}_i(\mathbf{x}_i) = f_i(\mathbf{x}_i + \delta\mathbf{q}_i). \tag{14}$$

The $\delta\mathbf{q}_i$ is often referred to as the backward error which makes Eq. (14) hold (i.e., by linearity, $\delta\mathbf{q}_i = f_i^{-1}(\delta\mathbf{y}_i)$.). We refer to the backward error as the quantization noise since the backward error behaves as the noise on the feature $\mathbf{x}_i$ [21].

Next, we explore the effects of quantization on the accuracy mathematically by linking the quantization error with data noise [21]. By the backward error analysis [21], [22], the input neurons multiplied by the quantized weights can be expressed as the perturbed input neurons multiplied by the original weights as follows:

$$x_{i,j} \times (w_{j,k} + \epsilon_{j,k}) = (x_{i,j} + \delta q_{i,j}) \times w_{j,k}, \tag{15}$$

where $x_{i,j}$ is the j^{th} neuron value at the i-th layer, $w_{j,k}$ is the weight connecting between $x_{i,j}$ and $x_{(i+1),k}$ prior to the activation layer, $\epsilon_{j,k}$ is the weight quantization error on the weight $w_{j,k}$, and $\delta q_{i,j}$ is the effect of the quantization on the input noise if $w_{j,k} \neq 0$. If $w_{i,j} = 0$, there is no quantization error. Replacing $x_{i,j}$ with $x^*_{i,j} + \delta d_{i,j}$, where $\delta d_{i,j}$ is the noise in the $x_{i,j}$ and $x^*_{i,j}$ is the noiseless feature map, the right side in Eq. (15) becomes:

$$(x^*_{i,j} + \delta d_{i,j} + \delta q_{i,j}) \times w_{j,k}. \tag{16}$$

Therefore, the quantization error does not affect the accuracy in practice, if:

$$|\delta q_{i,j}| \ll |\delta d_{i,j}|. \tag{17}$$

Based on Eq. (17), TinyOFL Property 2 (TP2) is as follows.

> *TinyOFL Property 2 (Equivalent Accuracy Condition for Weight Quantization in TinyOFL):* The weight quantization can achieve the equivalent accuracy to the baseline model if the condition of (17) is satisfied.

FL exhibits good performance when the data are assigned among clients in an Independent and Identically Distributed (IID) fashion, but the FL performance is degraded when the data distribution follows a non-IID fashion [16]. The data sharing method [23] that shares a portion of the training data among all clients can minimize the accuracy loss for non-IID FL cases. Hence, weight quantization along with the data sharing would be beneficial to minimize the accuracy loss incurred by non-IID data [23], [24].

C. WEIGHT QUANTIZATION EFFECTS ON BIAS-VARIANCE TRADE-OFF

The overfitting events rarely occur on neural networks trained with backpropagation [25]. However, the weight quantization adds noise to the labels of input samples in the feature map operations, increasing the variance part in the bias-variance analysis [26], [27]. Therefore, the effects of the variance part on the accuracy becomes more significant by weight quantization. This implies that a monotonously decreasing error curve from a neural network can be transformed into a double-descent curve as the number of hidden neurons increases due to the weight quantization [27]. Due to the restricted model size for TinyOFL, the overfitting due to the quantization noise rarely occurs in TinyOFL, compared to CFL. TinyOFL Property 3 (TP3) is described as follows.

> *TinyOFL Property 3 (Overfitting in TinyOFL):* Overfitting rarely occurs in TinyOFL due to its relatively restricted model capacity.

The weight quantization allows more parameters fit in the model, increasing the model size. Based on TP3, the increased model capacity would not cause overfitting in TinyOFL. For a high-end computing machine, the quantized networks would require regularization to prevent the overfitting effects caused by quantization. Therefore, unlike CFL, there exists an accuracy trade-off in TinyOFL between the accuracy loss by quantizing the weights and the accuracy gain by increasing model capacity by leveraging additional memory footprint generated by weight quantization. TinyOFL Property 4 (TP4) is described as follows.

> *TinyOFL Property 4 (Accuracy Trade-Off due to Weight Quantization):* There exists an accuracy trade-off in TinyOFL between the level of weight quantization and the model capacity.

D. WEIGHT QUANTIZATION EFFECTS ON TRANSMISSION

Unlike CFL, TinyOFL remote scenarios often require LPWAN due to the location and restricted power budget of the tiny device. Since LoRa [28] is one of the most widely used LPWAN, we adopt LoRa for our case study. LaRa communication in many countries follows a duty cycle regulation. A duty cycle represents the fraction of one period in which a signal is active. For example, for a duty cycle of 1%, sending the next LoRa packet should wait 99× packet sending time. Therefore, sending entire weights of a neural network using LoRa requires the duration T_{LoRa} in Eq. (18).

$$T_{LoRa} = T_{sp} \times \#_p \times 100/DC(\%), \tag{18}$$

where T_{sp} is the time cost to send a single packet of LoRa, $\#_p$ is the number of LoRa packets required for sending entire weights of a neural network, and $DC(\%)$ represents the duty cycle specified with a percentage. The weight quantization reduces $\#_p$, minimizing T_{LoRa} and the energy cost. The $\#_p$ can be estimated by the total storage required for the entire weights divided by the maximum bytes required for a single LoRa packet.

IV. EVALUATION

This section discusses the experimental setup and evaluates the effects of weight quantization on model performance, bandwidth savings, energy savings, and training time.

A. EXPERIMENTAL SETUP

Our case study extends a keyword spotting application [20] with four words. The four words include "Montserrat", "Pedraforca", "Vermell", and "Blau". Our experimental setup is as follows.

1) NEURAL NETWORK STRUCTURE FOR KEYWORD SPOTTING TASKS

Feedforward neural networks are used with a single hidden layer with the sigmoid activation functions (i.e., 650 neurons for the input layer, variable size for the hidden layer, and 4 neurons for the output layer). Each sample consists of 650 values from the 50 windows of 13 Mel Frequency Cepstral Coefficients (MFCC), which is a one-second mono audio recording of the keyword sampled at 16 KHz. The number of neurons in the hidden layer n_h varies throughout the experimentation, ranging from 5 to 25 neurons, with an increment of 5. The output layer consists of 4 neurons, each representing a different keyword. The predicted keyword is given by applying the softmax function with all the output nodes.

2) TRAINING AND TEST DATA FOR FL

The number of clients is: $n_K = 3$, the number of total training samples is: $n_{TR} = 480$, the number of training samples for each client is: $n_{tr} = 160$, and the number of test samples is: $n_{TE} = 60$. We divide the training samples into three 160 training samples (i.e., 40 samples for each class) to allocate them to each local device, as shown in Fig. 3. In order to generate repeatable experimental conditions that allow comparison and performance understanding, the speech samples used for the training were previously recorded. We added to the FL implementation of the server an auxiliary function that allowed to transmit the raw speech samples to the boards, one by one as if they were obtained by the board's microphone, to enable each client to do the local training at the corresponding FL round.

3) HARDWARE AND THE FL IMPLEMENTATION

We use the three Arduino Portenta H7 boards shown in Fig. 4 to run the FL clients. Each board has 1 MB RAM and a dual-core processor consisting of a Cortex M7 core running at 480 MHz and a M4 core running at 240 MHz. For our purpose, only the M7 core is used. The neural network model on each client is trained locally with its training data. Then the node sends its model with quantized weights to a PC, as explained in section II, where the FL server is hosted. We use the serial port with a stop-and-wait protocol to ensure

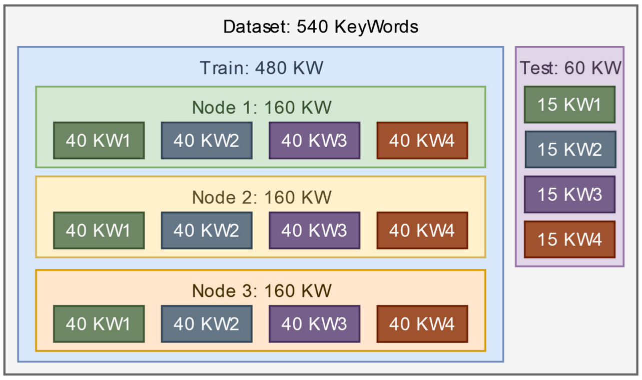

FIGURE 3. Distribution of the dataset samples.

the reliable communication of the model weights. The FL server on the PC generates a new global model from the received local models. For the training at the microcontroller-boards and generating the global models at the PC, the weights are in 4 byte float data types. Quantization is applied for the communication of the weights between clients and the server, for which the 4 byte weights are transformed into the l-bit representation, as depicted in Fig. 2.

4) TRAINING SETTING FOR FL

The mini-batch size is 1 in our experiments based on the memory resource constraint in the Arduino Portenta H7. The number of local updates per FL round is set to 4 ($n_l = 4$). With 160 training samples and 4 local update rounds, the number of FL rounds n_R is 40. All the weights are initialized with a uniform distribution for training.

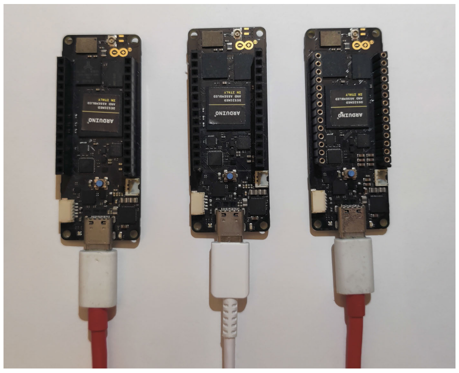

FIGURE 4. The three arduino portenta H7 boards used in the experiments.

The main configuration parameters used in the experimenation are summarized in Table 1.

B. EFFECTS ON GLOBAL MODEL ACCURACY

We evaluate the effect of weight quantization on the accuracy of the global model. The global model is updated at the server after every FL round, and its accuracy is evaluated with the 60 test samples. The accuracy is calculated based on Eq. (19).

$$Acc = n_{TP}/(n_{TP} + n_{FP}), \tag{19}$$

TABLE 1. Experimental setup summary.

Parameter	Value
Neural network type	Feedforward
# Hidden neurons	5 to 25
Number of clients	3
Number of keywords	4
Number of samples	480
Samples per device	160
Samples per device/keyword	40
Local updates per round	4
Total FL rounds	40

where n_{TP} and n_{FP} represent the number of true predictions and the number of false predictions.

1) ACCURACY VARIATION ACCORDING TO QUANTIZATION LEVEL

We explore the equivalent accuracy condition (17) with various weight quantization levels, compared to a baseline model. Fig. 5 shows the accuracy evolution according to $l = [3, 8]$ with a fixed $n_h = 25$. The model using 32-bit FP weights is used as a baseline. The accuracy of the model using 32-bit FP weights is equivalent to the accuracy using 7- or 8-bit weights, ranging between 95 and 98% with an equivalent convergence rate. The equivalent condition of (17) maintains if $l \geq 7$ in our case study, supporting TP2. The accuracy drops occur if $l \leq 6$, since the quantization errors on the locally updated weights affect the accuracy of globally updated weights. Particularly, using 3-bit weights annihilates the model's training ability.

Fig. 6 represents the accuracy using 5-bit weights on a single device without FL. In this figure, 480 training samples are used and the weight quantization is applied every 4 samples. Compared to Fig. 5, 5-bit weights with FL reach 74% accuracy with FL, but 50% accuracy without FL. Therefore, aggressive weight quantization can be allowed with FL, as discussed in TP1.

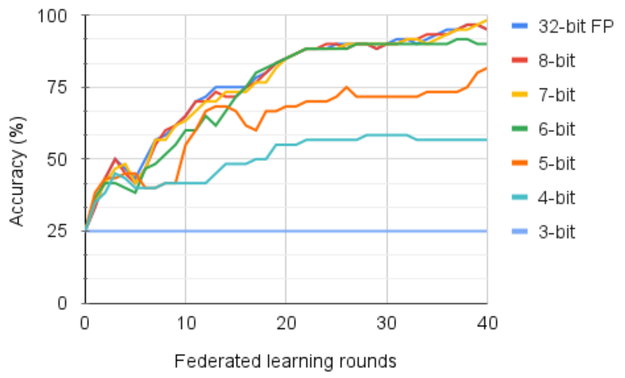

FIGURE 5. Accuracy according to weight quantization with different bit-widths.

2) ACCURACY VARIATION ACCORDING TO THE HIDDEN LAYER SIZE

Fig. 7 analyzes the accuracy concerning $n_h = \{25, 20, 15, 10, 5\}$ for $l = \{32, 16, 8, 7, 6, 5, 4\}$. The accuracy is obtained

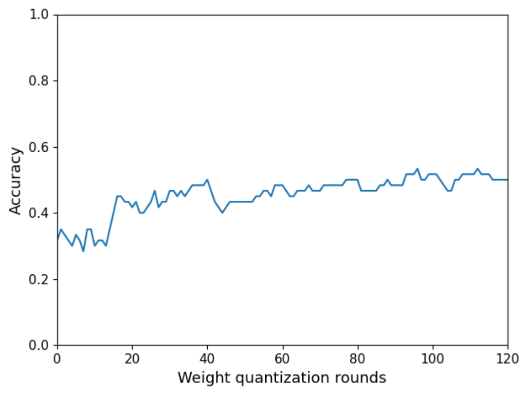

FIGURE 6. Accuracy with 5-bit weight quantization without FL.

after 40 FL rounds, with each client model trained with 160 samples (i.e., 40 samples of each class). The accuracy is improved as the number of neurons in the hidden layer increases, given a fixed number of bits for weights.

We observe that the least sufficient bit width for the weights depends on the network model capacity. For example, the accuracy of the network with full-precision weights is similar to 8-bit weights for all cases of the hidden layer sizes. With 7-bit quantization, a model requires 20 neurons at least to produce an equivalent accuracy to a full precision network. For $n_h \leq 15$, the quantization noise from using 7-bit weights affects the accuracy. I.e., the effect of quantization noise on the accuracy becomes weaker as the number of neurons in the hidden layer increases. In this case, we believe that the average quantization noise effect per weight on the accuracy becomes weaker since more portion of weights could become close to zeros as the size of the hidden layer increases [25]. The neural network having 25 neurons of the hidden layer classified 59 samples out of 60 test samples correctly using 7-bit weights, while 57 samples using 32-bit FP weights - the quantization noise accidentally impacted the accuracy positively.

We observe that TPs 3 and 4 hold in our experiments. Based on Figure 7, some models with a larger hidden layer sent with lower precision perform better than small models with higher precision, such as the case of the model of 5-bit weights with 20 neurons compared to the 7-bit weights with 10 neurons. Therefore, there is a potential to exploit the accuracy tradeoff between using more of an IoT node's communication and more of its local computation resources to reach a certain model accuracy based on TP4. Such accuracy tradeoff exists in TinyOFL thanks to TP3. However, TinyOFL with *aggressive* weight quantization can face overfitting. Fig. 8 represents the accuracy variation as the number of neurons in the hidden layer increases for 32-bit FP and 5-bit weights. The quantization noise larger than a threshold due to employing 5-bit weights degrades the accuracy as the hidden layer size increases (e.g. when the number of neurons is larger than 40). However, neural networks using 32-bit FP weights did not face such overfitting as discussed in TP3.

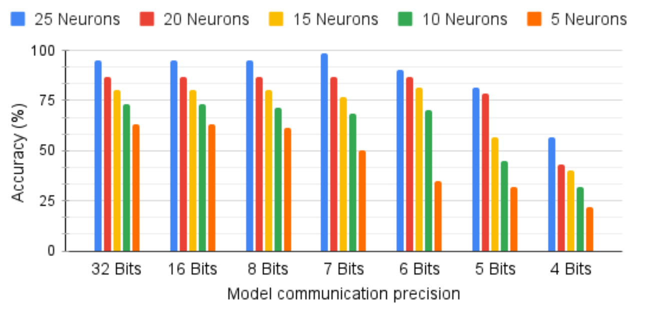

FIGURE 7. Comparison of the effects of the hidden layer size and number of bits of mixed precision on the accuracy.

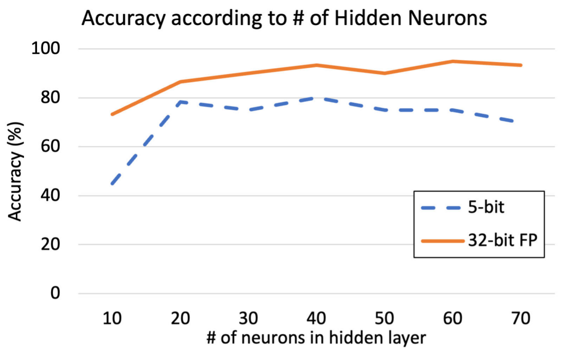

FIGURE 8. Overfitting with 5-bit weights.

3) EFFECTS OF QUANTIZATION ON WEIGHT DISTRIBUTION

The feedforward neural network with 25 hidden neurons has a total number of weights of 16379, consisting of 16250 weights between the input and hidden layer, 100 weights between the hidden and output layer, and 29 weights due to the biases of the neurons in the hidden and output layer. Fig. 9 represents the weight distribution of the quantized weights w' of a local model with 5 bits at a different number of FL rounds ($n_R = \{0, 20, 40\}$). We measure the number of the 5-bit weights within the integer range [0, 31] with step size, $s = 1$. As the training proceeds, the weights tend towards a Gaussian distribution, and the standard deviation becomes smaller (i.e., more weights come closer to zero.). In Fig. 9, the backpropagation property regulating most weights with small values still holds with 5-bit weights in our TinyOFL use case.

C. EFFECTS ON BANDWIDTH SAVINGS

Table 2 represents the communication cost of the model required for a single transfer in terms of *kBytes*. We calculate the cost based on the number of model parameters (weights and bias) according to different bit-width for weights and different numbers of neurons in the hidden layer. Notice that the total communication cost depends on the number of FL rounds, while the number of FL rounds depends on the training data pattern, the accuracy requirements, and on the re-transmission of messages.

Based on Table 2 and Fig. 7, using 7-bit weights with $n_h = 25$ can save bandwidth by 4.6× without losing accuracy,

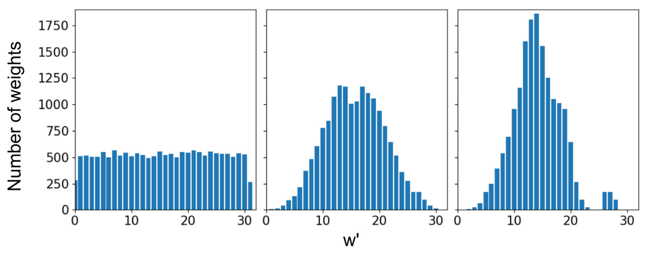

FIGURE 9. Distributions of 5-bit weights at 0, 20, and 40 (from left to right) FL rounds.

compared to 32-bit FP weights. Given 75% accuracy budget, 6-bit weights with $n_h = 15$ can minimize the communication cost, compared to 5-bit weights with $n_h = 20$. Therefore, the optimal bit-width for the weights should consider the accuracy trade-off between the quantization level and the network size. In our experiment, the error due to the quantization noise is compensated by the increased network size. The network size is often limited, so overfitting by increasing the network size hardly occurs in TinyOFL. In such cases, the negative effects of the quantization on the accuracy can be compensated by increasing the network size.

TABLE 2. kBytes of the weights required with respect to the model size and quantization.

	Hidden layer size (number of neurons)				
Quantization (bits)	25	20	15	10	5
32	63.98	51.19	38.39	25.60	12.81
16	31.99	25.59	19.20	12.80	6.40
8	16.00	12.80	9.60	6.40	3.20
7	14.00	11.20	8.40	5.60	2.80
6	12.00	9.60	7.20	4.80	2.40
5	10.00	8.00	6.00	4.00	2.00
4	8.00	6.40	4.80	3.20	1.60

D. EFFECTS ON ENERGY SAVINGS AND MODEL DELIVERY TIME

We analyze the energy cost of the communication for the Arduino Portenta H7 nodes of our FL deployment. The Portenta board supports three wireless network interfaces: Wifi, Bluetooth, and LoRa. We singled out LoRa for our study since LoRa is customized for the energy-efficient communication of remote IoT nodes, which is an important application field for nodes with embedded learning [29].

In LoRa, one of the configuration parameters that determines the energy consumption is the Spreading Factor (SF). A different SF has to be configured depending on the distance between two nodes. The higher the SF, the higher the distance that the link covers. With a higher SF, however, the packet transmission has a longer time-on-air. This longer duration of the LoRa packet transmission then consumes more energy of the IoT device than the quicker transmissions with a lower SF.[3] Therefore, in general, the reduction of the amount of data transmitted with quantized weights will translate into

[3]https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/

less energy consumption, but the concrete savings depend on the application case.

The communication latency is also affected by the proposed quantization approach. The weight quantization can reduce the latency since less data is transmitted. However, also other factors come into play, such as the network topology of the LPWAN communication technology. The latency for a specific application case is different if the nodes of an FL network are, e.g., at a one hop distance or in a mesh network with several hops. Also, the traffic in the network which affects the packet collisions and possible re-transmissions, influences the latency of the specific application case.

We utilize the approach used in [30] to control the transmission of low-level LoRa packets. An SX1276 LoRa Breakout Board [31] is used to measure the energy cost of the LoRa communication. We set a maximum payload to 222B. This is a packet size that is accepted by most of the regulations worldwide.[4] We measure the energy consumption of sending a single LoRa message to estimate the total cost of communication for the number of packets required to send models with different bit widths from one node to another. For the experiment, we use the RadioLib library that supports the SX1276 module.[5] For the LoRa communication setup, we use the default configuration of the library with a TX power of 10dBm and SF 9. Figure 10 shows a snapshot of a measurement.

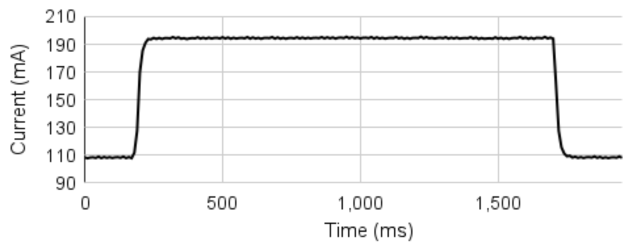

FIGURE 10. **Current measurement in a LoRa packet communication with 222B payload.**

It can be seen that when the node sends a LoRa packet, the current in the idle state is around 87 mA, while 194 mA is required for the active communication during 1520 ms that corresponds to the Time-on-Air of sending of a single LoRa packet with a payload of 222B, given communication settings. Table 3 shows the estimated energy cost based on the estimated number of LoRa packets related to the bit width of the weight quantization, the current given, and the voltage of 5V, and Time-on-Air measurements. It can be observed that the energy cost is linearly proportional to the number of bits of the weights transferred. This result is consistent with the analysis performed on the energy consumption for different data rates and payloads in [32], which shows that higher payload sizes which results in a higher Time-on-Air increase the energy consumption. Applying a reduction of the quantization level which results in a lower number of packets that have to be transmitted, both the energy consumption and delivery time are reduced proportionally to the lower number of bits of the weights.

Notice that the real delivery time of a model should consider a duty cycle regulation, making the delivery time much longer, compared to the Time-on-Air (i.e., the Time-on-Air corresponds to 100% duty cycle in Eq. (18)). If we consider a duty cycle of 1%, sending the model with weights of 32 bit width would take around 100h, and using 8-bit quantized weights can reduce it to 25h. With packet loss, there will be an additional number of packets due to the need for re-sending of packets and the overhead of a reliable message delivery protocol. Since such packet loss does not depend on the weight quantization in LoRa, the bandwidth and energy savings would be in proportion to weight quantization in practice. Given these considerations and the measurements obtained, the weight quantization considering the accuracy trade-off of TP4 can reduce both the energy and the end-to-end delivery time significantly in TinyOFL.

TABLE 3. **Total energy and Time-On-Air of sending the weights with different bit widths of the largest neural network used in the experiment with a hidden layer of 25 neurons.**

No. bits	No. LoRa packets	Energy (Joule)	Time-on-Air (s)
32	2361	801.5	3589
16	1180	400.8	1794
8	590	200.4	897
7	516	175.3	785
6	443	150.3	673
5	369	125.2	561
4	295	100.2	449

V. RELATED WORK

Mixed precision model training methods were mainly investigated for single neural networks. For instance, lower and higher precision arithmetic are employed for the Multiplications and the Accumulations (MACs) in matrix multiplications to accelerate training without losing accuracy [33], [34], [35]. In this regard, various data types have been studied for the lower precision and higher precision arithmetic for MACs, such as IEEE half precision and IEEE single precision data type [34], BFloat16 and IEEE single precision data type [36], and an 8-bit FP and DLFloat data type [37]. The idea behind such mixed precision training was to accelerate training speed by reducing the memory access costs for MAC operations.

It has been a main research topic in FL to explore how to reduce the cost of model communication. One of the main approaches is weight quantization. Probabilistic weight quantization was proposed in [17] as one of the sketching methods for compression. Suresh et al. [38] proposed stochastic rotated quantization to improve the accuracy without decreasing the number of quantization bits, compared to [17]. Yoon et al. [39] proposed a trainable

[4]https://lora-developers.semtech.com/documentation/tech-papers-and-guides/the-book/packet-size-considerations/

[5]https://github.com/jgromes/RadioLib

weight dequantizer used in the central server to reconstructs heterogeneous low-bitwidth weights according to each client environment to appropriate full-precision weights. The work in [40] proposed Quantized Stochastic Gradient Decent (QSGD), in which the gradients of the local model are quantized with a configurable number of bits depending on the network bandwidth. While the above mentioned works transmit in a compressed representation all gradients of the model to the server, in Lazy Aggregated Quantization (LAQ) the client examines the change of each weight gradient and skips the transmission of the less informative quantized gradients [41]. As an extension to LAQ, Adaptive Quantitzed Gradient (AQG) was proposed [42]. In AQG, depending on the amount of update of a gradient, the client quantizes gradients with more or fewer bits, applying thus an adaptive number of bits for the transmission. Quantizing the gradients of the model instead of the weights has an asymmetric performance gain since it reduces only the cost of communication from the client to the server, but not from the server to the clients. Differently, quantizing the model weights benefits the bidirectional communication. It has been pointed out in [43] that realistic environments have non-IID data, for which a performance drop of FL with many existing compression schemes was observed. Different methods for reducing the model communication cost can be combined as in the FedPAQ algorithm [44], which includes optimizing the relation between the amount of local SGD training and communication to the server, considering partial client participation in an FL round and applying parameter value quantization. Mitra et al. [45] proposed an online FL variant that minimizes the difference between the prediction error averaged over all clients and the predication error in the global model in the server after averaging local weights over entire time steps. This method allows clients to communicate with the server infrequently, minimizing the communication cost in FL.

The research for FL on microcontroller-based IoT devices is still a new area with yet only a few works published. Kopparapu et al. proposed TinyFedTL for federated transfer learning [46]. A compressed version of TensorFlow's MobileNet on the device was used to perform an image recognition task on the Arduino Nano 33 BLE. The work [20], [47] analyzed how FL could benefit model performance, compared to training the model with local data only.

Most of the above works are developed from a rather theoretical perspective, and the performance is often evaluated with simulations. Differently, by the prototype developed in our work, we aim to bridge the existing research results with the emerging practitioner community of TinyML.

Several research and practical implementations combine ML on remote IoT nodes with LoRaWAN communication. An example is the mosquito logger system proposed in [29]. The IoT device performs the signal processing and classification. Instead of sending raw data, only the classification result is communicated to the remote gateway, resulting in a huge saving of bandwidth usage of the LoRa link. This kind of application uses a trained model that is flashed on the device before deploying the node in the field. The quantized model weight communication that we studied in this paper can contribute to such types of applications enabling models to be trained by FL over LoRa. Moreover, in this paper, we analyzed the effects of weight quantization on TinyOFL in terms of the stability (TP1), accuracy (TP2), overfitting (TP3), and the accuracy trade-off between the quantization level and the extensible network width (TP4) with theoretical and experimental support.

VI. CONCLUSION

This paper performs a comprehensive analysis of the effects of weight quantization on TinyOFL in terms of the accuracy, stability, overfitting, and accuracy trade-off between quantization level and expandable neural network width, bandwidth savings, energy savings, and delivery time with theoretical and empirical support by a TinyOFL use case on three Arduino Portenta boards. We believe that such foundations on the effects of weight quantization on TinyOFL will provide computing system engineers and machine learning engineers with guidelines on how to practically apply weight quantization for their TinyOFL use cases.

The weight quantization has advantages in TinyOFL. In the case study, the 7-bit weight quantization in TinyOFL saved bandwidth by 4.6× without losing model accuracy, compared to full-precision weights. We also observe that 5-bit weights reach 74% accuracy with FL, but 50% accuracy without FL, implying that TinyOFL allows the weight quantization to be more aggressive, compared to TinyOL. The weight quantization can cause overfitting in TinyOFL. For example, the 5-bit weights degrade the accuracy as the number of neurons in the hidden layer size becomes larger than 40, while 32-bit FP weights did not face such overfitting. It is advantageous to utilize three Arduino Portenta boards since our findings TP1 to TP4 can be supported empirically given our training data. E.g., the main challenges using more clients in our experiments are related to the training dataset size. The training data would have to be partitioned between a lot of devices, and each device would be trained with very few samples. However, it is disadvantageous to utilize the three boards since TinyOFL with more Portenta boards would allow us to explore the effects of weight quantization on the accuracy and the communication bandwidth cost more deeply with respect to the increased number of clients.

In future work, we aim to explore deeper the optimization potential of weight quantization for memory and communication resource needs of the FL server using more Portenta boards. While in our experimental scenario the FL server was hosted on a PC, a LoRa mesh network would allow the clients to communicate with an FL server hosted on an IoT device as well. However, the centralized FL server design requires a significant amount of memory at the server for storing the local models before averaging and also needs bandwidth for the communication of the model exchanges with each client. Applying the proposed weight quantization for a more decentralized FL server design suitable to run on tiny IoT nodes could foster a machine learning intelligence entirely provided by an IoT network.

REFERENCES

[1] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.

[2] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-IID data," in *Proc. IEEE Int. Conf. Big Data (Big Data)*. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2020, pp. 15–24, doi: 10.1109/BigData50022.2020.9378161.

[3] H. Ren, D. Anicic, and T. Runkler, "TinyOL: TinyML with online-learning on microcontrollers," 2021, *arXiv:2103.08295*.

[4] F. De Vita, G. Nocera, D. Bruneo, V. Tomaselli, and M. Falchetto, "On-device training of deep learning models on edge microcontrollers," in *Proc. IEEE Int. Conferences Internet Things (iThings), IEEE Green Comput. Commun. (GreenCom), IEEE Cyber, Phys. Social Comput. (CPSCom), IEEE Smart Data (SmartData), IEEE Congr. Cybermatics (Cybermatics)*, Aug. 2022, pp. 62–69.

[5] S. Disabato and M. Roveri, "Incremental on-device tiny machine learning," in *Proc. 2nd Int. Workshop Challenges Artif. Intell. Mach. Learn. Internet Things*. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 7–13, doi: 10.1145/3417313.3429378.

[6] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for Internet of Things: Recent advances, taxonomy, and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1759–1799, 3rd Quart., 2021.

[7] J. Haxhibeqiri, E. De Poorter, I. Moerman, and J. Hoebeke, "A survey of LoRaWAN for IoT: From technology to application," *Sensors*, vol. 18, no. 11, p. 3995, Nov. 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/11/3995

[8] B. Vejlgaard, M. Lauridsen, H. Nguyen, I. Z. Kovacs, P. Mogensen, and M. Sorensen, "Coverage and capacity analysis of SigFox, LoRa, GPRS, and NB-IoT," in *Proc. IEEE 85th Veh. Technol. Conf. (VTC Spring)*, Jun. 2017, pp. 1–5.

[9] M. A. M. Almuhaya, W. A. Jabbar, N. Sulaiman, and S. Abdulmalek, "A survey on LoRaWAN technology: Recent trends, opportunities, simulation tools and future directions," *Electronics*, vol. 11, no. 1, p. 164, Jan. 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/1/164

[10] J. M. Solé, R. P. Centelles, F. Freitag, and R. Meseguer, "Implementation of a LoRa mesh library," *IEEE Access*, vol. 10, pp. 113158–113171, 2022.

[11] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1622–1658, 3rd Quart., 2021.

[12] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 4, pp. 1595–1623, Apr. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821003335

[13] B. Sudharsan, J. G. Breslin, M. Tahir, M. Intizar Ali, O. Rana, S. Dustdar, and R. Ranjan, "OTA-TinyML: Over the air deployment of TinyML models and execution on IoT devices," *IEEE Internet Comput.*, vol. 26, no. 3, pp. 69–78, May 2022.

[14] V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the extreme edge: A survey on reformable TinyML," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–30, Dec. 2023, doi: 10.1145/3583683.

[15] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1474–1479.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, in Proceedings of Machine Learning Research, vol. 54, A. Singh and J. Zhu, Eds., Apr. 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[17] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2017, *arXiv:1610.05492*.

[18] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in IoT," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5986–5994, Jul. 2020.

[19] J. Wilkinson, *Rounding Errors in Algebraic Processes*. Upper Saddle River, NJ, USA: Prentice-Hall, 1963.

[20] N. L. Giménez, M. M. Grau, R. P. Centelles, and F. Freitag, "On-device training of machine learning models on microcontrollers with federated learning," *Electronics*, vol. 11, no. 4, p. 573, Feb. 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/4/573

[21] J. Lee and H. Vandierendonck, "Towards lower precision adaptive filters: Facts from backward error analysis of RLS," *IEEE Trans. Signal Process.*, vol. 69, pp. 3446–3458, 2021.

[22] J. H. Wilkinson, "Error analysis of floating-point computation," *Numerische Math.*, vol. 2, no. 1, pp. 319–340, Dec. 1960, doi: 10.1007/bf01386233.

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

[24] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on Non-IID data: A survey," *Neurocomputing*, vol. 465, pp. 371–390, Nov. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221013254

[25] R. Caruana, S. Lawrence, and C. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in Neural Information Processing Systems*, vol. 13, T. Leen, T. Dietterich, and V. Tresp, Eds. Cambridge, MA, USA: MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2000/file/059fdcd96baeb75112f09fa1dcc740cc-Paper.pdf

[26] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Comput.*, vol. 4, no. 1, pp. 1–58, Jan. 1992.

[27] Z. Yang, Y. Yu, C. You, J. Steinhardt, and Y. Ma, "Rethinking bias-variance trade-off for generalization of neural networks," in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 1–11.

[28] *LoRa Alliance*. Accessed: Mar. 3, 2022. [Online]. Available: https://lora-alliance.org/

[29] D. Vasconcelos, M. S. Yin, F. Wetjen, A. Herbst, T. Ziemer, A. Förster, T. Barkowsky, N. Nunes, and P. Haddaway, "Counting mosquitoes in the wild: An Internet of Things approach," in *Proc. Conf. Inf. Technol. Social Good (GoodIT)*. New York, NY, USA: Association for Computing Machinery, Sep. 2021, pp. 43–48.

[30] D. L. Pino, F. Freitag, and M. Selimi, "Designing a double LoRa connectivity for the Arduino Portenta H7," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Jul. 2022, pp. 1–2.

[31] (2023). *868MHz SX1276 LoRa Breakout Board with Antenna*. Accessed: Jun. 20, 2023. [Online]. Available: https://paradisetronic.com/en/products/868mhz-sx1276-lora-breakout-board-antenne

[32] L.-T. Tu, A. Bradai, Y. Pousset, and A. I. Aravanis, "Energy efficiency analysis of LoRa networks," *IEEE Wireless Commun. Lett.*, vol. 10, no. 9, pp. 1881–1885, Sep. 2021.

[33] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A study of BFLOAT16 for deep learning training," 2019, *arXiv:1905.12322*.

[34] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. K. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[35] J. Lee, L. Mukhanov, A. S. Molahosseini, U. Minhas, Y. Hua, J. Martinez del Rincon, K. Dichev, C.-H. Hong, and H. Vandierendonck, "Resource-efficient convolutional networks: A survey on model-, arithmetic-, and implementation-level techniques," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–36, Jul. 2023, doi: 10.1145/3587095.

[36] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," 2018, *arXiv:1811.06992*.

[37] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Advances in Neural Information Processing Systems*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2018, pp. 7675–7684. [Online]. Available: http://papers.nips.cc/paper/7994-training-deep-neural-networks-with-8-bit-floating-point-numbers.pdf

[38] A. T. Suresh, F. X. Yu, H. B. McMahan, and S. Kumar, "Distributed mean estimation with limited communication," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2017.

[39] J. Yoon, G. Park, W. Jeong, and S. J. Hwang, "Bitwidth heterogeneous federated learning with progressive weight dequantization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2022.

[40] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst.*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., Long Beach, CA, USA, Dec. 2017, pp. 1709–1720. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/6c340f25839e6acdc73414517203f5f0-Abstract.html

[41] J. Sun, T. Chen, G. B. Giannakis, Q. Yang, and Z. Yang, "Lazily aggregated quantized gradient innovation for communication-efficient federated learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 4, pp. 2031–2044, Apr. 2022.

[42] Y. Mao, Z. Zhao, G. Yan, Y. Liu, T. Lan, L. Song, and W. Ding, "Communication-efficient federated learning with adaptive quantization," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 4, pp. 1–26, Aug. 2022, doi: 10.1145/3510587.

[43] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-i.i.d. data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[44] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization," in *Proc. 23rd Int. Conf. Artif. Intell. Statist.*, in Proceedings of Machine Learning Research, vol. 108, S. Chiappa and R. Calandra, Eds., Aug. 2020, pp. 2021–2031. [Online]. Available: https://proceedings.mlr.press/v108/reisizadeh20a.html

[45] A. Mitra, H. Hassani, and G. J. Pappas, "Online federated learning," in *Proc. 60th IEEE Conf. Decis. Control (CDC)*, Dec. 2021, pp. 4083–4090.

[46] K. Kopparapu, E. Lin, J. G. Breslin, and B. Sudharsan, "TinyFedTL: Federated transfer learning on ubiquitous tiny IoT devices," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Affiliated Events (PerCom Workshops)*, Mar. 2022, pp. 79–81.

[47] M. M. Grau, R. P. Centelles, and F. Freitag, "On-device training of machine learning models on microcontrollers with a look at federated learning," in *Proc. Conf. Inf. Technol. Social Good (GoodIT)*. New York, NY, USA: Association for Computing Machinery, Sep. 2021, pp. 198–203, doi: 10.1145/3462203.3475896.

NIL LLISTERRI GIMÉNEZ received the bachelor's degree in informatics engineering specializing in computing from Universitat Politècnica de Catalunya (UPC), in 2022.

He's been working since then on various research projects, focusing on machine learning and edge computing.

JUNKYU LEE received the Ph.D. degree in computer engineering from The University of Tennessee Knoxville, Knoxville, TN, USA, in 2012. He was a Postdoctoral Researcher with the Joint Institute for Computational Sciences, The University of Tennessee Knoxville–Oak Ridge National Laboratory; the School of Electrical and Information Engineering, The University of Sydney; and the Institute of Electronics, Communications and Information Technology, Queen's University Belfast. He is currently a Research Fellow with the Institute for Analytics and Data Science, University of Essex. His research interests include establishing mathematical and computational foundations for energy-efficient machine learning.

FELIX FREITAG received the Electrical Engineering degree from the Technical University of Munich, Germany, in 1993, and the Ph.D. degree from the Technical University of Catalonia (UPC), Barcelona, Spain, in 1998. He is currently an Associate Professor with the Computer Architecture Department, UPC. His current research interests include edge computing and federated machine learning.

HANS VANDIERENDONCK (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees from Ghent University, Belgium, in 1999 and 2004, respectively. He is currently a Professor of high-performance and data-intensive computing with the School of Electronics, Electrical Engineering and Computer Science, and a fellow of the Institute of Electronics, Communications and Information Technology, Queen's University Belfast, where he directs the Centre on Data Science and Scalable Computing. His research aims to build efficient and scalable computing systems for data-intensive applications.