Pervasive and Mobile Computing 93 (2023) 101819

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Embedded federated learning over a LoRa mesh network

Nil Llisterri Giménez, Joan Miquel Solé, Felix Freitag [*]

Universitat Politecnica de Catalunya, Barcelona, 08034, Spain

ARTICLE INFO

Article history:
Received 15 July 2022
Received in revised form 6 June 2023
Accepted 19 June 2023
Available online 22 June 2023

Keywords:
Embedded machine learning
Federated learning
LoRa
IoT

ABSTRACT

In on-device training of machine learning models on microcontrollers a neural network is trained on the device. A specific approach for collaborative on-device training is federated learning. In this paper, we propose embedded federated learning on microcontroller boards using the communication capacity of a LoRa mesh network. We apply a dual board design: The machine learning application that contains a neural network is trained for a keyword spotting task on the Arduino Portenta H7. For the networking of the federated learning process, the Portenta is connected to a TTGO LORA32 board that operates as a router within a LoRa mesh network. We experiment the federated learning application on the LoRa mesh network and analyze the network, system, and application level performance. The results from our experimentation suggest the feasibility of the proposed system and exemplify an implementation of a distributed application with re-trainable compute nodes, interconnected over LoRa, entirely deployed at the tiny edge.

1. Introduction

Machine learning models are nowadays deployed in ever smaller computing devices including microcontrollers. In such devices, the sensor data capture, signal processing, and machine learning tasks are performed directly on the device. The increased microcontroller computing capacities and the availability of open tools such as Tensorflow Lite have generated considerable interest in embedded machine learning on microcontrollers [1].

Embedded machine learning opens an opportunity for affordable smart compute nodes since microcontroller boards are typically cheap compared to higher-end computing devices. Furthermore, the microcontroller's material and energy consumption is low, which is positive from an environmental point of view. However, the computing capacity of an embedded compute node as a single device is limited. But leveraging the interconnection of devices through the networking interfaces of the boards could increase their computing capacity. Interconnected machine learning training with networked microcontroller boards could benefit from integrating a larger dataset leveraging the local data of the distributed nodes. Furthermore, the computing needs for machine learning model training could be shared among the network of nodes. While such concepts have already been successfully exploited in machine learning on higher-end devices, there is still a lack of understanding and experience in applying these ideas to embedded systems [2].

To equip a microcontroller with a neural network for embedded machine learning, there are two main approaches: off-device training and on-device training of the model. Off-device training is a very popular approach and training is typically done on a powerful computer, where enough computing capacity is available to store the complete dataset and train the model up to the required accuracy [3]. Once the model is trained off-device, it is flashed to the microcontroller and used for inference.

* Corresponding author.
E-mail address: felix.freitag@upc.edu (F. Freitag).

https://doi.org/10.1016/j.pmcj.2023.101819

In on-device training, the machine learning model is trained on the microcontroller itself. While this approach has started receiving increasing interest from the research community, it is still a niche approach in comparison to off-device training [4]. On-device training has limitations due to the restricted microcontroller hardware. Typically, the flash memory of the microcontroller is very small and does not allow the storage of the full training data set. Training with few samples, however, influences the accuracy of the model. Another issue is the reduced computing capacity of the microcontroller, which results in that the training process on the microcontroller being much slower than when performed with a GPU or CPU of personal computers. Despite these performance limitations, on-device training on microcontrollers has the advantage of allowing models of machine learning applications to adapt to new data. For IoT applications that integrate smart distributed and remote devices, where external updates with new models are not feasible, the capacity for on-device training has an interesting potential.

Federated learning has raised the interest of the research community as a technique for model training that does not require the sharing of a node's local data, and thus allows training with privacy-sensitive data [5]. In federated learning, instead of training a single model with a centralized dataset, a machine learning model is trained at each device with its local dataset and then merged into a global model. In [6] a comprehensive survey reviews the application of federated learning in edge devices. The multiple federated learning rounds, however, can be communication intensive. In each round bandwidth is consumed for sending the machine learning model from the training nodes to a central aggregator where the new global model is computed, and sending it back to each node. While this communication need is not an issue for those nodes connected to high bandwidth links, it can be an issue for distant tiny IoT nodes in which sometimes a low bandwidth communication technology such as LoRa is the only available network connectivity.

LoRa is a widely used wireless communication technology in IoT applications. It provides long range communication, low power consumption, and low data rates to meet the requirements of remote IoT nodes which from time to time send small amounts of sensor data. A LoRa link between two nodes can cover several kilometers of distance, thus making LoRa suitable for the communication of nodes in geographically-spread IoT applications [7].

Most IoT applications that use LoRa apply the LoRaWAN architecture [8]. This architecture builds a star topology that establishes a single hop between an end node (sensor node) and the gateway. The gateway has Internet connectivity and forwards the data messages received through the LoRa packets to higher layers of the IoT application. The LoRaWAN architecture, however, does not directly interconnect the end nodes between themselves. Therefore, LoRaWAN is not suitable for building a horizontally interconnected network of IoT devices.

LoRa mesh networks have been proposed for diverse scenarios which cannot be addressed well by the LoRaWAN architecture [9]. For geographically widely spread IoT nodes and low gateway density, it is not always possible for an end node to reach the gateway in a single hop. However, it has been shown that LoRaWAN applications can be extended by multi-hop LoRa, where intermediate nodes operate as repeaters that broadcast traffic to other LoRa nodes to finally reach a gateway [10]. Another reason for LoRa mesh networks is to enable IoT applications that do not use the centralized cloud-based LoRaWAN stack. The Meshtastic application, for instance, is a gateway-less IoT application where decentralized LoRa nodes communicate with each other over a mesh network [11].

While both embedded machine learning and LoRa-based IoT applications have made progress separately, the confluence of machine learning in microcontrollers and networked LoRa connectivity is not yet well understood. In this paper, we propose embedded federated learning on microcontroller boards that uses the communication capacity of a LoRa mesh network. We design a tiny networked distributed computing infrastructure with nodes that consist of the Arduino Portenta H7, a recent microcontroller board equipped with embedded sensors suitable for diverse machine learning tasks, and the TTGO LORA32 microcontroller board, acting as an embedded router to transmit packets over LoRa-based links. Using real nodes we show the implementation feasibility. We evaluate the system performance with regard to machine learning accuracy, power consumption, and bandwidth usage. Our results envision smart distributed applications deployed at the tiny edge, for which embedded federated learning provides the model adaptivity.

The novelty consists in enabling the embedded federated machine learning entirely in the IoT layer, where the tiny compute nodes interact over a LoRa mesh network. This involves a decentralized federated learning protocol and support services to form federated learning networks. Fig. 1 illustrates the system. In the mesh network, some of these nodes host the federated learning application for updating and retraining their machine learning model. We consider that IoT nodes can run dynamic federated learning networks without a central server in a peer-to-peer relationship, such that any node can operate as a server or client. The LoRa mesh network provides support services for the network and application level interconnection of these tiny nodes which run the federated learning application.

The main contributions are:

- We propose smart retrainable embedded compute nodes interconnected over a LoRa mesh network as a system architecture for a new kind of distributed applications at the tiny edge.
- We analyze the performance of a distributed machine learning application when the embedded machine learning model is trained on the microcontroller board with the federated learning technique over the LoRa mesh network.
- Our experiments are conducted with real microcontroller boards and the used software implementation has been made publicly available, making thus the results reproducible and facilitating their practical application.

We develop the software implementation of the proposed system and build a prototype. The experimentation is done in a real environment with three Arduino Portenta H7 boards that run a keyword spotting (KWS) application trained by

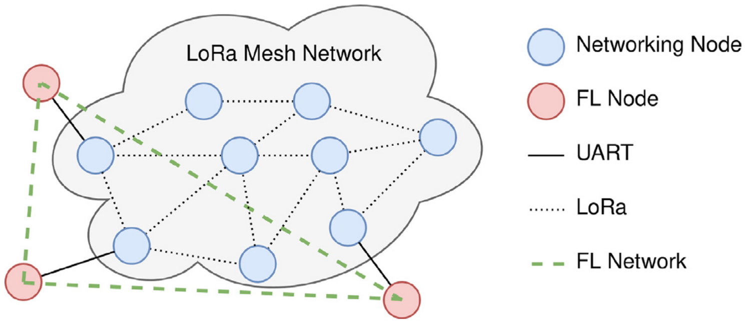

Fig. 1. Federated learning in IoT nodes interconnected over a LoRaMesh network.

federated learning, and three TTGO LORA32 which establish the LoRa mesh network for the application. The dataset of the speech samples used for the training and the software has been made available in publicly accessible repositories.[1,2]

2. Background and related work

2.1. Federated learning with edge devices

Federated learning is a machine learning approach in which machine learning models are trained at edge nodes instead of using a centralized cloud-based infrastructure. Federated learning avoids the local data having to leave the node, allowing thus to exploit data for training which otherwise due to privacy constraints may not be available. Federated learning, however, requires the availability of sufficient local computing capacities at the node to conduct the training, as well as to have a network link with communication capabilities that allow transmitting machine learning models between the participating nodes. While powerful edge nodes such as that of 5G systems do have such computing and communication capacities to support federated learning, Single-Board Computer (SBC) and tiny embedded IoT edge nodes can face both computation and communication challenges [12].

Core challenges of federated learning include the communication efficiency, the heterogeneity of systems, the heterogeneity of training data at the different nodes, and the privacy of the data [13]. A recent survey presented in [14] puts these challenges in the light of the IoT, where the resource constraints of the devices exacerbate some of these challenges. Within several open research challenges highlighted, sparsification is an approach to address the heterogeneity IoT, by selecting only a subset of devices for conducting a federated learning process, where the selection criteria can be tailored to the specific IoT needs of the application case. Another survey by Nguyen et al. [15] looks more closely into the resource management and deployment aspects of federated learning in IoT devices. Both topics are part of the identified open research challenges. None of both surveys, however, include works on federated learning performed over Low Power Wide Area Network (LPWAN) communication technologies such as LoRa.

Training with federated learning in SBC was proposed in several works, for instance in the Flower framework, where Android phones, Raspberry Pi, and NVIDIA Jetson are used [16]. Given the computing capacity of these devices, the federated learning client for the Android phones was implemented in Java applying specific TensorFlow Lite Model Personalization support for Android Studio. The federated learning client for the Raspberry Pi and NVIDIA Jetson was implemented in Python. Model training was performed with federated learning using TensorFlow Lite on the boards. While federated learning in SBC faces resource constraints, such as shown in [17] for wireless networks, the computing and communication resources of such systems are still magnitudes higher than that of the embedded IoT devices which we target in this paper.

Machine learning with embedded devices has raised the interest of a fast growing community of researchers and practitioners, e.g. TinyML.[3] Popular machine learning (ML) frameworks such as Tensorflow have specific versions that can be applied to embedded systems. Easy-to-use workflows that train machine learning models in an off-device training approach at a personal computer or with cloud-based services,[4] prune and quantize the model to reduce its size [18], and finally flash the optimized model on the microcontroller board have become available.

Many popular microcontroller boards are supported by TensorFlowLite.[5] These boards are equipped with several sensors, targeting being used for machine learning applications. In [19] two of the Arduino boards are compared, the

1 https://github.com/NilLlisterri/TTGO-LoRaMesher
2 https://github.com/NilLlisterri/FL_LoRaMesher
3 https://www.tinyml.org/
4 e.g. https://www.edgeimpulse.com/
5 https://www.tensorflow.org/lite/microcontrollers

Arduino Nano 33 BLE Sense, which integrates, among other sensors, a motion detection unit and a microphone, and the Arduino Portenta H7. While the Nano 33 BLE Sense is a versatile board for which the community has demonstrated a large number of machine learning applications, the Portenta H7 is a dual core high-end industry grade board that can be adapted to specific needs through extension shields. Given the potential of the Portenta H7, in this paper, we use the Arduino Portenta H7 as compute node that hosts a machine learning application for doing federated learning over a LoRa mesh network.

Federated transfer learning in embedded devices was proposed in the work of Kopparapu et al. [20]. In their system called TinyFedTL, the popular Arduino Nano 33 BLE Sense board is used for an image recognition task. Only the output layer of a neural network is trained with backpropagation, while the previous layers are obtained from a compressed version of TensorFlow's MobileNet. The advantage of transfer learning is that only a subset of the neural network is trained, which saves computing resources of the resource-constraint microcontroller board. Transfer learning, however, addresses machine learning applications that aim to adapt a pre-trained model to a specific need. This approach does not target the training of a machine learning model from scratch.

To provide adaptability for trained machine learning models, Disabato and Roveri [21] proposed incremental transfer learning for a *k*-nearest neighbor classifier. The authors used two types of devices, a Raspberry Pi 3B+ and an STM32F7 microcontroller with 512 kB of RAM and 2 MB of flash memory. With image and audio benchmarks the authors showed the feasibility of these two hardware platforms for doing on-device incremental transfer learning. The model training was done on-device on a single node, focusing on the performance comparison of the two boards, but the work did not apply the training with federated learning.

In TinyOL [22] incremental one-device training on streamed data is proposed. The target hardware is the Arduino Nano 33 BLE Sense where an autoencoder neural network is trained. The motivation for doing on-device training was to obtain flexible IoT applications which can adapt to different scenarios. The work proposes generic building blocks for on-device training. The system is evaluated with an anomaly recognition task on streaming data. The work focused on showing on-device training as a means for a neural network on a microcontroller to adapt over time. On-device training was performed on isolated nodes without federated learning. It was observed that the incremental training, due to the limited capacity for storing the data set at the embedded device, improved slowly compared to off-device training, but improves its performance and adapts over time.

In [23] the authors argue for on-device training of deep learning models in order to facilitate that smart boards can detect new patterns that may emerge over time. The system was evaluated for anomaly detection in DC motor operation with an autoencoder network. The network was first trained with normal behavior patterns of the motor and then an anomaly was injected. The system was not trained with federated learning, but the application case provides a continuous stream of training samples to the network, allowing a single network to be trained sufficiently well for distinguishing between normal and abnormal motor operations.

In our work in [24], we developed a centralized federated learning application. It was evaluated with three training nodes that used the Arduino Nano 33 BLE Sense boards. The federated learning server, which aggregates the local models into a new global model, was run on a PC. The boards were connected over USB to the PC. Thus, the federated learning workers communicate with the server over serial ports. In the evaluation of the machine learning performance, it was observed that the machine learning performance was higher when the boards are trained with federated learning, leveraging also the higher total number of training samples available at the participating nodes. Furthermore, we analyzed the resource usage of the machine learning application run on the microcontroller board. It was shown that on-device training requires additional memory compared to off-device training, such as new data structures to store the gradients from the backpropagation algorithm.

Decentralized federated learning architectures were proposed to avoid the need for the centralized aggregation of the federated learning model. For instance, in BrainTorrent [25], a peer-to-peer approach is introduced in which the nodes of the federated learning training directly communicate with each other, assuming both the role of server and client. In that work, the decentralized architecture of federated learning is motivated by the scenario of multiple independent medical centers in which the component of a centralized aggregator does not exist. In the work of Savazzi et al. [26] another decentralized serverless federated learning approach is proposed for the IoT. The evaluation is done in an Industrial Internet of Things (IIoT) scenario of an industrial factory plant where the devices have a dense communication network. Other decentralized federated learning approaches often refer to the decentralized management of the federated learning process [27].

2.2. LoRa communication in the IoT

LoRa is a popular LPWAN communication technology for the IoT [7]. Compared to other long range IoT communication technologies such as NB-IoT and Sigfox, LoRa is operated in the unlicensed band and does not require any license fees or network operator. This property eases the deployment of LoRa-based networks and enables organic growth, such as crowd-sourced LoRa networks as in Helium.[6] The fact that a LoRa network can be established by multiple parties could also be leveraged for building different types of distributed applications within such a network. In comparison with

[6] https://www.helium.com/

Sigfox, a LoRa packet has a higher payload size, which makes LoRa suitable for a larger range of IoT applications [28]. Many professional IoT applications with LoRa use the LoRaWAN standard developed by the LoRa Alliance.[7] Active maker communities use the community edition of cloud-based LoRaWAN stacks such as that provided by The Things Network[8] for building LoRaWAN-based applications.

The LoRaWAN architecture requires a certain density of gateways to assure that any LoRa end node can reach a LoRaWAN gateway. This density of gateways, however, cannot always be guaranteed. To address this issue, multi-hop LoRa mesh networks were proposed to enable a larger geographic spread of the IoT nodes [10]. Real field deployments of multi-hop LoRa networks, such as in the work of Ebi et al. [29], who proposed a synchronous LoRa mesh network, have shown the feasibility of this approach.

Some IoT applications combine embedded machine learning and LoRaWAN, e.g. the mosquito logger system proposed in [30]. In that work, the signal processing and machine learning tasks are performed directly at the IoT device. A lightweight machine learning model is used to fit into the resource constraints of the sensor node. Instead of sending raw data, only the classification result is communicated to the remote gateway. The sending of the classification has the advantage of saving the bandwidth of the communication link.

LoRa IoT applications do not necessarily need a gateway that is connected to the Internet. In some applications, the nodes within a LoRa mesh network communicate with each other without applying the LoRaWAN architecture. A prominent example of such a case is Meshtastic [11], a real operational application that only communicates data within a LoRa mesh network without any gateway. Each user of Meshtastic is equipped with a smartphone (without mobile network coverage) and an embedded system board with a LoRa transceiver. Bluetooth is used for communication between both devices. In terms of system design the smartphone hosts the user application, while the embedded system board takes care of the networking over LoRa with the other nodes. Our system design, described in Section 3.1, applies a similar approach.

In LoRa mesh networks with a diameter of more than one hop, LoRa packets have to be forwarded by intermediate nodes. The forwarding of LoRa packets to the destination can be done by flooding, such as currently done in Meshtastic,[9] or be based on routing. Flooding has the risk of congesting the network if no additional measures such as hop counters are activated. Differently, when routing protocols are used, the forwarding of messages is determined by the routing protocol. However, there is also a cost in the latter approach since the routing tables must be maintained at each node. The work in [31] presents an initial study of a LoRa mesh network based on RadioHead library. In [32] a proactive distance-vector routing protocol was designed for a LoRa mesh network. In our work, we apply this design as described in Section 3.3.

A few works proposed LoRa networks as a complementary communication infrastructure for distributed applications. In [33] the LoRaX system was proposed for extending through LoRa the device's connectivity to the Internet to underserved regions. Network access was achieved through a pervasive low data rate LoRa network. In [34] a messaging system was implemented as a case study in which clients connect to LoRa nodes for reaching other participants either within the LoRa network or through hubs on the Internet. The authors remarked that the presented architecture could generalize to other application cases.

In light of the works reviewed in Sections 2.1 and 2.2, federated learning has been brought into higher-end edge devices, but the usage of federated learning in embedded systems is yet very little with only very few published works. There is however a recent growth of works reporting on on-device training for embedded systems, showing the interest of the community in moving from off-device training to approaches that enable the adaptivity of the neural network models on smart nodes. For application cases when the number of training samples at a node is low, leading to underperforming models of isolated nodes, federated learning may be a technique to improve the model performance. IoT communication technology has recently presented LoRa mesh networks to enable the interconnection of IoT nodes. Initial applications have been presented by lightweight text messaging over a LoRa communication substrate. It was shown that using specific hub nodes enables the possibility of cross-network applications with the Internet.

In this paper, we integrate embedded learning and LoRa communication. We develop and evaluate a distributed machine learning application that conducts federated learning over a LoRa mesh network, representing an application with smart embedded nodes in the IoT layer that can adapt and improve their model over time. To experiment with the application in real nodes, we integrate the LoRaMesher library which we have developed [35].

3. System design and implementation

3.1. System overview

Our goal is to create a distributed machine learning application that can be trained with a variable number of tiny networked embedded nodes. Each node should run an instance of the ML application and train a neural network on-device, using the samples it captures locally. Each node should be capable of performing Federated Learning (FL) with

[7] https://lora-alliance.org/
[8] https://www.thethingsnetwork.org/
[9] https://meshtastic.org/docs/developers/Firmware/mesh-alg

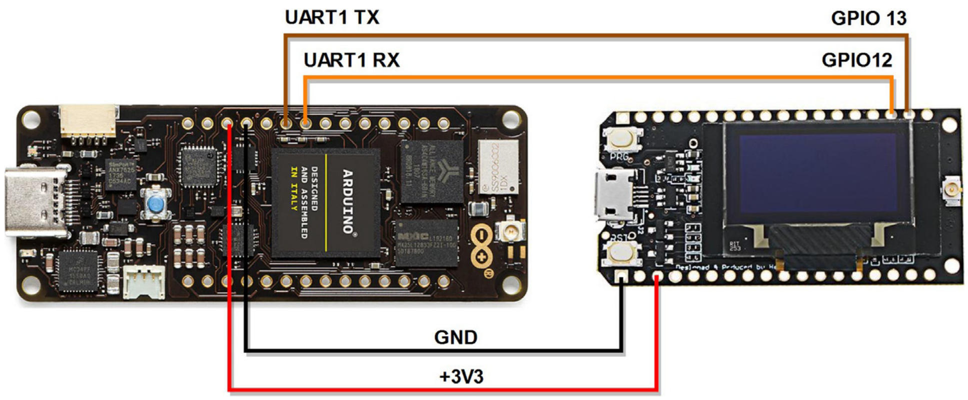

Fig. 2. UART connection between the Arduino Portenta H7 and the TTGO-LORA32.

other nodes over a LoRa network. These nodes will not be coordinated nor monitored by any central server, and construct a decentralized architecture.

The question arises as to how the targeted computing infrastructure will be built with hardware. One option to consider is using a single embedded system board to handle both the ML application and also the networking operations. This implies that one board can manage the networking while also performing the CPU-heavy operations of machine learning. While threading is available in most RTOS, the amount of high-priority threads which would be needed to spawn the tasks and to maintain low latencies on both the ML and networking aspects could be significant.

Another option is to apply the *Separation of Concerns* principle and split the networking and application tasks into two boards, which is the approach we followed. By doing so, each board will be able to allocate all of its resources to one of the two tasks. Specifically, for our system implementation of the networking task, we use the TTGO-LORA32 board, applying it as an embedded router by making use of the LoRaMesher library for sending LoRa messages among the nodes [35]. All the machine learning application logic is put into the Arduino Portenta H7 board. We mention that this separation of the application stack into two boards also eases software updates, such that when a new version of the network or application components is available, only one of the boards needs to be updated. For the interconnection of the two boards, UART is used as shown in Fig. 2.

3.2. Machine learning application

We use the keyword spotting application that was initially developed in [24]. The application trains a three layer feedforward neural network to detect three keywords. Hence the output layer consists of three neurons, the input layer of 650 neurons for the mel-frequency cepstral coefficients (MFCCs) corresponding to a one second speech signal, and the hidden layer size is a parameter that can be changed in the experimentation.

To organize the application on the Arduino Portenta H7 dual-core architecture, we leverage the lessons learned from [19]. Therefore, we use the less-powerful M4 core to handle the communication with the router for the model exchange in the federated learning process. The M7 core handles the audio recording, pre-processing, and training of the neural network.

The size of the neural network and the data type used to store the weights directly affects the amount of data that has to be transmitted during the federated learning process. In the context of the LoRa data transmission that we apply in this work, the data size is especially critical, since in several regions of the world LoRa duty cycle limitations are in place, which limits the number of bytes that a LoRa node is allowed to transmit within a certain time. Furthermore, the maximum size of a LoRa packet is limited to 256B. While in [24] the neural network weights were represented by 4-byte float variables, in the work we present here we also experiment with 2-byte integers and 1-byte integers.

3.3. Lora mesh network library

As introduced in Section 3.1, our scenario consists of distributed machine learning nodes interconnected over LoRa communication. For the development of our application, we use the LoRaMesher library,[10] which implements a distance-vector routing protocol for communicating messages among LoRa nodes [35]. For the interaction with the LoRa radio chip, the LoRaMesher library uses RadioLib, a versatile communication library that supports the SX127X LoRa series module available on the TTGO-LORA32 board.

10 https://github.com/LoRaMesher/LoRaMesher

The application level code that uses the LoRaMesher library, in our case the machine learning application, should have at least two different RTOS tasks, one for the reception of packets and one for sending of packets. The *receiver packet algorithm* for the reception task, shown in Listing 1, is designed with two different functions. There is an RTOS notification wait, which allows the LoRaMesher library to notify when a new packet has arrived, and a function that gets the next packet. After having received and processed a packet, this packet is deleted from the queue.

Algorithm 1 Algorithm for packet reception

```
loop                                              ▷ forever
    waitRecivedPacketNotification()
    packet = getNextUserPacket()
    processPacket(packet)
    deletePacket(packet)
end loop
```

Listing 2 provides the *send packet algorithm* for the sending task. The application code needs to call a function specifying the destination address and the payload to be sent. In our case, the address corresponds to the identifier of a node in the LoRa mesh network with which federated learning will be performed. The payload may correspond, for instance, to a subset of weights of the neural network model.

Algorithm 2 Algorithm for sending packets

```
function SendPacket(Destination, Payload)
    createPacketAndSend(Destination, Payload)
end function
```

Besides offering the communication interface to the application code, there are some configurations of the LoRaMesher library that the developer can modify to adapt to specific needs. For example, it is possible to configure the periodicity of the routing table updates, the periodicity of the route timeouts and LoRa-related parameters such as the radio frequency, bandwidth, Spreading Factor (SF), and adding the payload Cyclic Redundancy Check (CRC).

3.4. Application integration and federated learning process

The application starts once the node's boards are turned on. The LoRaMesher library [35] installed on the TTGO-LORA32 boards, maintains and regularly updates a routing table that contains all known nodes in the LoRa mesh network. The machine learning application can query this routing table from the LoRaMesher library. From inspecting the entries in the routing table, the application identifies the other nodes present in the given LoRa mesh network. In the case of our implementation, the machine learning application exploits this feature of the LoRaMesher library, which avoids the need for a search function for the discovery of other nodes.

During the training process with federated learning, a training node pools all the nodes in the network at a certain point to know the number of epochs each node went through in its local training. This information is used to help the node decide whether to start a federated learning round. The criteria we implemented is the number of epochs since the last time the nodes performed federated learning. If this amount is greater than a specified threshold, the federated learning process between the pooling node and the selected peer begins. We note that the implemented trigger for initializing a federated learning round in our experimentation is only one option, other more sophisticated triggers can be designed to meet different requirements.

Fig. 3 shows the lightweight communication protocol which we have designed and implemented to send and receive data between the application on the Arduino Portenta H7 and LoRaMesher on the TTGO-LORA32. The sending and receiving of messages are related to the exchange of machine learning models in the federated learning process. The routing table request is used for the discovery of other nodes in the LoRa mesh network.

3.5. Decentralized federated learning

In centralized federated learning, a server is connected with all the training nodes in the network and orchestrates when and how the nodes should combine their models in the federated learning process. These nodes send their model and other metadata to the central server, which aggregates them and sends the new model back to the nodes. This centralized design implies that all the nodes have to be able to reach the central server at all times. We argue that for remote tiny IoT nodes as targeted in our work, this centralized design can be difficult to be fulfilled in some scenarios.

We, therefore, implement a decentralized federated learning approach that works without a central server. The training nodes are enabled to make decisions about the federated learning process. As explained in Section 3.3, through the LoRaMesher library, the application nodes can communicate with all the nodes in the network. Thus, each node can gather information about all the other nodes, such as the number of epochs performed in the local training and the distance (in the number of hops between nodes). Such information and additional metrics, like the last federated learning interaction

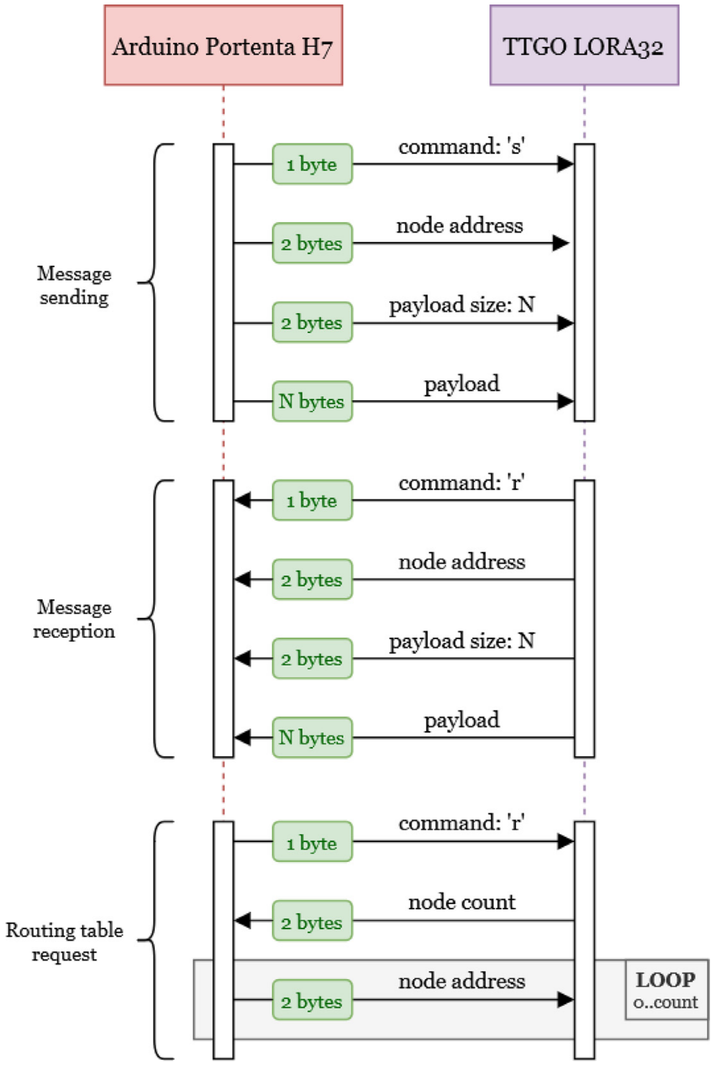

Fig. 3. Communication protocol between the Arduino Portenta H7 and the TTGO-LORA32.

with each node, could be taken into account by a node to autonomously decide whether to start a federated learning process.

Listing 3 describes one of the possible algorithms for a node to control the federated learning process. The *shouldPerformFL* function uses the node state and metrics of other nodes to decide whether federated learning should be performed with a specific node. In our experimentation, the node performs federated learning if the number of new epochs of another node is greater than a configured value, but other metrics like battery level, last federated learning time, accuracy, etc. could also be used.

The communication between the Arduino Portenta H7's M4 core, the M7 core, and the other nodes can be seen in Fig. 4. The M7 core begins the federated learning process by instructing the M4 core via RPC to start communication with the other nodes. The M4 core, which handles all outgoing and incoming traffic through the UART connection to the TTGO-LORA32 board, will request all nodes in the network to send the number of epochs they registered. It will then evaluate these metrics and decide which node (if any) is best to perform federated learning. Then it will request the neural network weights of the selected node, broken into small batches that can fit in a LoRa payload. Finally, with all the weights stored in a memory region shared by both cores, the M4 will notify the M7 core. The M7 core will use those weights to calculate the weighted average and update its neural network model.

In order to guarantee the delivery of all the messages in the FL process, the M4 core implements a method for detecting when a packet was not successfully transmitted and resending it, transparent to the M7 core. After forwarding the message to the TTGO LORA32 router for transmission, it will pool the modem for a receipt confirmation packet. If the confirmation is not received after a predefined amount of time, the original message will be sent again and the process will begin again.

Algorithm 3 Algorithm for decentralized federated learning

```
 % The node_epochs map contains the epochs of the node in the last FL
for node in getNetworkNodes() do
    metrics = getNodeMetrics(node)
    if (shouldPerformFL(metrics)) then
        weights = getNodeWeights(node)
        elapsed_epochs = metrics.epochs - node_epochs[node]
        new_weights = weightedAvg(weights, elapsed_epochs)
        updateLocalWeights(new_weights)
        node_epochs[node] = metrics.epochs
    end if
end for
```

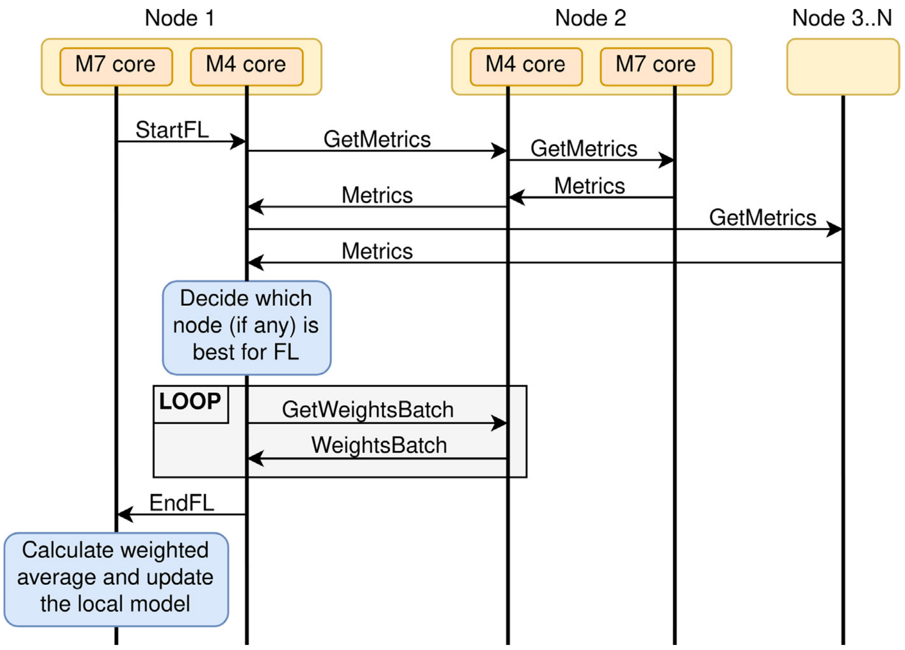

Fig. 4. Interaction between machine learning nodes during the FL process.

4. Evaluation

4.1. Experimental environment

We describe in the following the choices related to the experiments.

Dataset: We use a dataset of 480 samples of 3 different voiced keywords which is publicly available.[11] The advantage of using a pre-recorded dataset instead of recording each sample during the experiment is that when performing different experiments, the samples used by the machine learning training are always the same, so the obtained results can be compared and are reproducible. The voice samples available are not pre-processed, so they have the same format as if they were recorded at this instant by the microphone on the board.

Application hosting: As introduced previously in Section 3.1, for hosting the machine learning application we selected the Arduino Portenta H7 board. It contains two cores, a Cortex M7 running at 480 MHz and a Cortex M4 running at 240 MHz. This dual-core architecture allows for real parallelism in the two main application tasks, by assigning communication to the M4 core, and the training of the neural network to the M7. The application requires having, at times, as much as 3 times the size of the weights of the neural network in memory. The first instance will be used by the M7 core only and will be used for inference and backpropagation. The second instance will be created from the original weights when another node starts the federated learning process. This ensures that while the communication is active, the weights will

[11] https://github.com/NilLlisterri/FL_LoRaMesher/tree/main/datasets

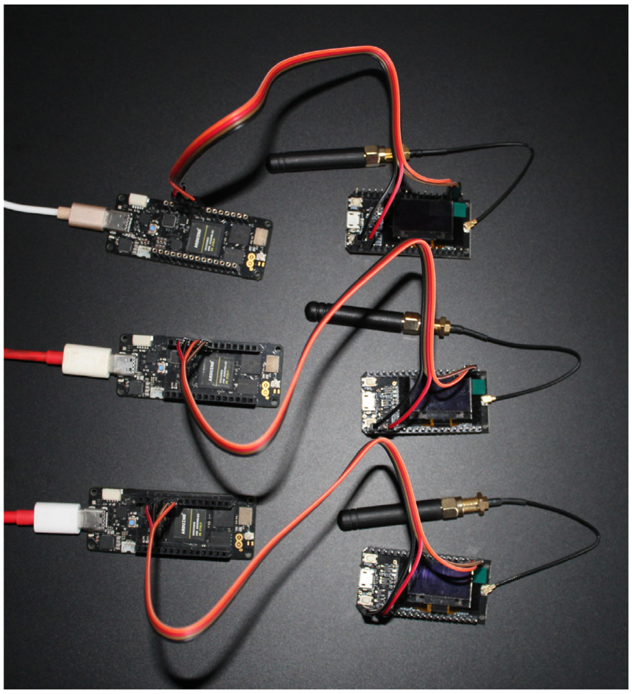
Fig. 5. Hardware used: Three Arduino Portenta H7 and three TTGO-LORA32 boards.

not be updated mid-sending. Finally, the third instance will be used to store the weights received from another device, before merging them with the current neural network. The Arduino Portenta assigns 523 kB to the Cortex M7 core and 295 kB to the Cortex M4. The Vision Shield, which includes a microphone, is added to the Portenta board.

Networking device: To have LoRa connectivity between the application running on the Portenta boards, we chose the TTGO-LORA32 board. It is based on the ESP32 microcontroller, running two cores at 240MHz, with an SX1276 transceiver that provides LoRa connectivity. The board also has a small built-in OLED display. The main reason to use this board is that the LoRaMesher library was available for this device. The TTGO-LORA32 can also be easily interfaced with another board using the UART communication protocol that we used. For the experimentation the OLED display has shown to be useful since it allows to have brief information about the state of the LoRa mesh network, avoiding thus the need to monitor the state of the network via the device's serial port. For the experimentation, we use six boards forming three nodes as can be seen in Fig. 5.

Experimental environment: The experiments are performed in a single location, a room where all the boards are located close to each other, at a distance of about 20 cm. The SF of the LoRa messages is fixed at SF7. For the collection of experimental data, each Arduino Portenta H7 is connected via a serial port to a PC, from where the experiments are managed. With this setup, the experiments can be programmed and automated, for instance, the sending of the speech training samples as if they were captured by the nodes themselves at determined instants. The debug information and traces are also obtained and aggregated on the connected PC.

4.2. Experimentation

4.2.1. Quantization impact

To be able to perform federated learning over LoRa efficiently, the amount of data to be transmitted has to be optimized. The payload consists mainly of the weights of the neural network. Those were initially stored using 4-byte floats, which had a great amount of precision, but a LoRaMesher packet, limited by the size of a LoRa packet, could only contain up to 50 weights at a time.

We experiment with weights represented by three different data types (see Section 3.2) to understand the trade-off between the size of the neural network and the model accuracy. The size of the hidden layer is 25 neurons. The first experiment uses 4-byte floats, the second reduced the size of the weights to 2-byte shorts, and the third uses single bytes. The training was done with 120 data samples. Then inference with the trained model was done for another 60 data samples. Federated learning was not used in this experiment. After each new sample, the loss is calculated using the MSE and stored for inspection. Fig. 6 shows the results after all samples are sent. It can be seen that the accuracy obtained in inference mode is practically the same for the three data types of weights.

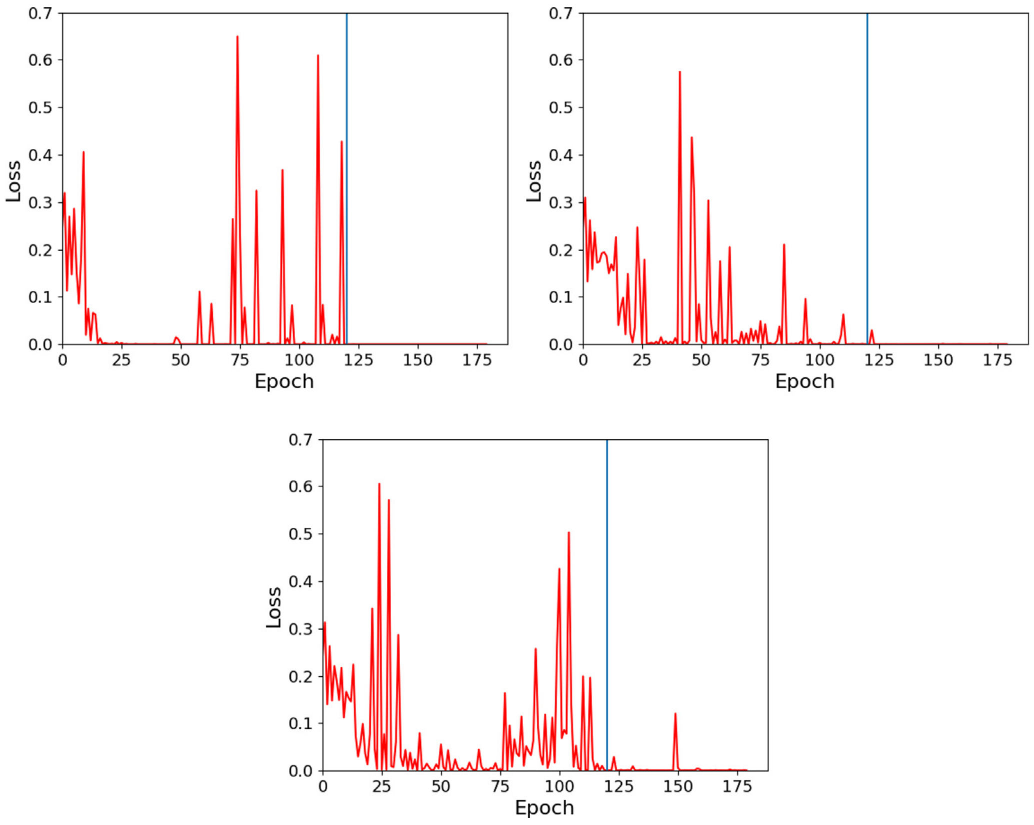

Fig. 6. Loss vs. epochs during training and testing using different data types to store the weights. From left to right and top to bottom: 4-byte floating point numbers, 2-byte integers, and 1-byte integers.

With such little impact on the accuracy of the neural network model and the need for small payloads to be transmitted via LoRa, we decide to use in the following experiments the tiniest datatype to store the weights, a signed 8-byte integer.

The memory cost of each neuron is given by the formula:

$$neuron\_cost = input\_layer\_size \cdot datatype\_bytes \cdot 2$$

Therefore, reducing the number of bytes of the data type from floats to just one byte makes the memory cost fourfold lower.

When measuring the cost of the quantization, there is a slight, but necessary, time overhead. The computations in the inference and backpropagation process are performed using floating point arithmetic operations, so a previous mapping from the stored one-byte weight to the required float is performed. In our experimentation, we observed that this added operation increased the total inference time by about 0.6 ms. This represents a 10% slowdown in the operation, from 6 ms to 6.6 ms.

4.2.2. Accuracy

We conduct an experiment to compare how decentralized federated learning can affect the model accuracy. In this experiment, three nodes (A, B, and C) are trained with a total of 120 samples. Then, 40 new samples are used to test the performance of the resulting model. The metric used to assess the model performance is the loss, computed for each new sample as the MSE of the neural network before training it. In the training process, for every 30 samples, node A performs federated learning with both other nodes, B and C, applying algorithm 3.

The total amount of transmitted data can be calculated using the following formula:

$$bytes\_transmitted = neuron\_cost * hidden\_layer\_size * fl\_rounds$$

In this experiment, the size of the hidden layer is set to 15. Node A performs federated learning 4 times with node B and 4 times with node C, at epochs 30, 60, 90, and 120. With this configuration, the amount of training data transmitted within the LoRa network is 156 kB.

The resulting evolution of the model performance at the three nodes can be seen in Fig. 7. We can observe how node A, the one that requests the weights (pictured in red) to the other nodes (B in green and C in blue) substantially reduces

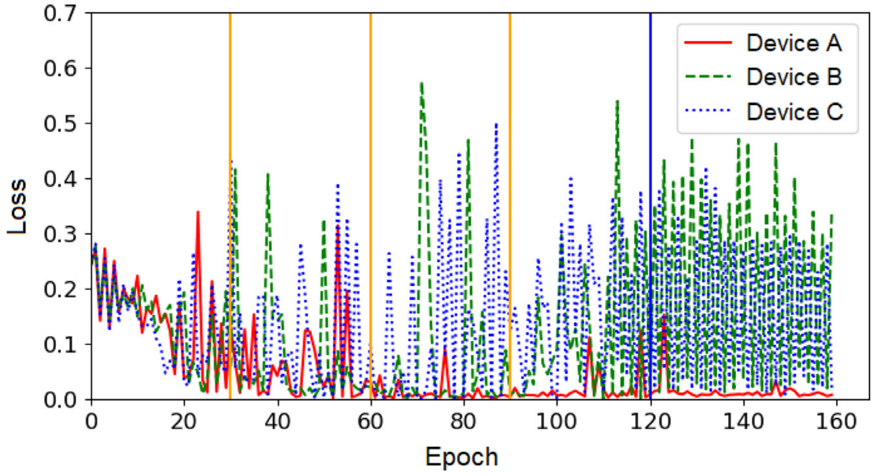

Fig. 7. Loss vs epochs evolution throughout the training and testing process with a network of 15 hidden neurons. Federated learning of node A at epochs 30, 60, 90, and 120.

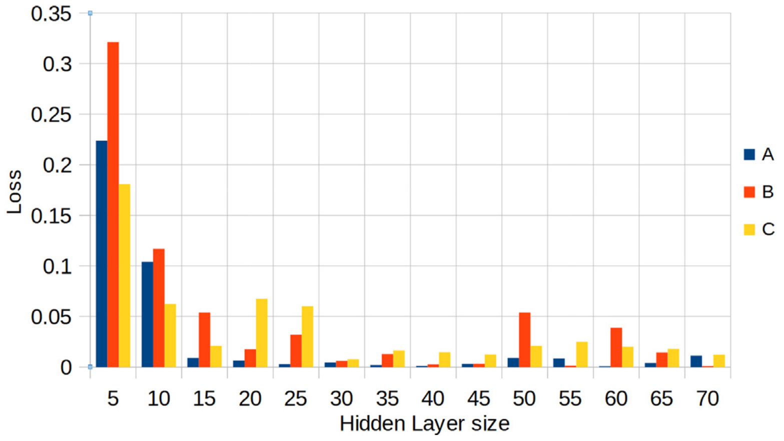

Fig. 8. Mean MSE vs hidden layer size for nodes A, B and C. Federated learning was done by node A at epochs 30, 60, 90, and 120.

its loss as the training process advances and achieves this reduction sooner than the other two. Analyzing the inference done with the last 40 test samples we could observe that node A, which merged its model with the other nodes, obtained an accuracy of 100% in the KWS task, while the other two nodes scored 62.5% (B) and 50% (C).

The same experiment was performed with different sizes of the hidden layer. The benefits of the federated learning process were evidenced across most hidden layer sizes, obtaining node A better model performance and with fewer epochs than the other two nodes. For hidden layer sizes over 20 neurons, however, all nodes started to report accuracies close to or at 100% in the test samples, due to the relatively small complexity of the problem. To obtain a more detailed understanding of each model's performance, the mean of the loss in the test samples was analyzed. As can be seen in Fig. 8, node A reports lower mean loss values than the other two nodes for most hidden layer sizes while being the only node that trained its model with federated learning at rounds in epochs 30, 60, 90, and 120.

The experiment was repeated multiple times, each time changing the seed that was fed to the random function that shuffled the data samples. Having a relatively small dataset consisting of a total of 120 training samples for each node, the order in which the samples are sent to the devices can have a relevant effect on the training process. On a few occasions, the training results for a specific hidden layer size were affected and led to a better or worse model performance than expected. Such a case can be seen for example for the hidden layer size 55, where node B reports a minor loss than node A.

We also experimented with more and less frequent federated learning rounds. In the case of decentralized federated learning, the number of rounds can be a means for a node to compensate for its lack of local training data at the cost of bandwidth consumption. If a node has few training samples, by doing more frequent federated learning rounds a node can seek to improve its local model by merging it more often with the models from other nodes which were trained with a larger number of samples. Differently, if a large number of local training data was used for the local model training compared to that trained at other nodes, the potential performance gain of the model resulting from a federated learning round may not be worth the incurring cost of communication.

4.2.3. Energy consumption

Energy consumption can play a major role in the feasibility of an IoT application. LoRa enables long-distance communication between nodes, allowing the deployment of IoT devices at remote locations. In some of these scenarios, nodes can be hard to be reached for being maintained. If these nodes are battery-powered, the energy consumption must be taken into account or even be sustained by, for example, a solar cell.

The requirements of the type of IoT applications has an influence on the importance of the device's energy consumption and how the node should be powered. Smart IoT devices that perform continuous machine learning operations on sensor input may require the device to be active most of the time, and have rechargeable batteries, solar power or other forms of permanent energy supply. Differently, tiny nodes that just infrequently transmit sensor data have a stricter energy budget and the energy consumption can be optimized through a sleep mode to last for a long time. It was shown in [30] that even for specific learning applications, the energy consumption of the node depends on the actual sensor input since it triggers the device's activity. For instance, a higher number of classifications may imply a higher number of writes to microSD cards, and an increased number of LoRa messages, all influencing the device's total power consumption for a given time frame.

Both boards of our system implementation operate at 3.3 V. In our experiments the Arduino Portenta H7 board is powered from the computer's USB port at 5 V. The board has a buck converter which transforms the voltage to 3.3 V. We use the +3V3 and ground pins to power the TTGO-LORA32 board (Fig. 2).

We measure the power consumption of each board at different stages of the application lifecycle. The TTGO-LORA32 board had 4 levels of consumption: When the device is awake, waiting to receive or send any messages, the consumption is 52 mA. When the device is processing messages or interacting with the serial interface, the consumption rises to 76 mA. While receiving a message, the board consumes 84 mA, and while sending it the maximum consumption is achieved, 141 mA. In our experiments, we used a SF of 7, the minimum of LoRa. When increasing the SF to the maximum, i.e. the SF of 12, the longer transmission time increases the energy consumption of sending a LoRa packet, but does not increase the instantaneous current consumption.

The TTGO-LORA32 board has a sleep mode which can be exploited in some scenarios. For example, when the federated learning happens only at a certain time, the board could be put to sleep until that time, while the Arduino Portenta H7 keeps capturing new samples and training the local model. In this sleep mode, activated by the command *esp_deep_sleep_start()*, the current consumed by the device was 4 mA.

The Arduino Portenta H7 board has 2 cores. When both of the cores are idle, the consumption is 128 mA. When the board is receiving a message from the TTGO-LORA32 board via UART or trains the neural network, the consumption rises to 165 mA. This consumption is high for a battery-powered device, but the board can put both cores to sleep until an interrupt occurs to save energy. To wake up the M4 core when a new message is received from the routing device, an additional connection between the TTGO-LORA32 and the Portenta board could be added. This connection could be used to trigger an interrupt in the Portenta board that wakes up the M4 core to start reading the serial data.

4.2.4. Federated learning over lora

The duration of the federated learning process is directly affected by many factors, being one of them the bandwidth of the communication link. As explained in Section 2.2, LoRa communications links have low bandwidth. One of the reasons relate to legal restrictions. In Europe for instance, LoRa transmissions must obey the 1% duty cycle limit, forcing a data rate below the technical limitations of the LoRa technology. Another reason for reduced throughput is the possibility of collisions of LoRa packets, which result in packet losses.

The number of weights of the neural network to be transmitted, which depends on the number of neurons, affects the duration of training. For the specific case of the neural network we used, each neuron in the hidden layer increases the size of the federated learning payload by 1.3 kB. As a consequence, each additional neuron results in longer training times.

We conduct an experiment with the LoRa parameters configured in the LoRaMesher library shown in Table 1. We measure the time it takes to transmit the neural network weights for different hidden layer sizes over the LoRa link and the number of LoRa packets, in order to gain insight into this behavior (Table 2). While the exact values of the training times depend on our specific configuration, the magnitudes indicate that a federated learning round over LoRa links can take hours to be accomplished. It can be stated that such performance is not feasible for applications that have fast federated learning requirements. However, considering that the training process can be done on-device, this training time might not be a critical requirement for many applications, while the approach offers other advantages over off-device training such as the capacity for model adaptation.

Throughout the experiments, we experienced some collisions between LoRa messages. The percentage of collisions we observed in the experiments with a network consisting of 3 nodes was very low, between 0 and 1%. Nevertheless, as more LoRa packets are present in the network, more collisions are expected to happen. Because of this, applications in higher traffic LoRa networks may require an additional algorithm that controls the delivery of messages. In the work presenting the LoRaMesher library the reduction of the Packet Delivery Rate (PDR) caused by collisions is analyzed [36]. To handle packet losses, LoRaMesher offers a stop-and-wait communication, which can be activated by the application developer to obtain reliable messaging over LoRa.

Table 1
LoRa parameter setting in the LoRaMesher library.

Parameter	Values
Spreading Factor	7
Bandwidth and band	125 kHz. EU863-870.
Preamble length	8
Transmission power	+10 dBm
Coding rate	4/7
CRC checking	Header & Payload
Max. packet size (bytes)	222
Max. payload size (bytes)	211
Max. packet delay (s)	48

Table 2
Hidden layer size vs. LoRaMesher transmission.

Hidden layer size	Transmission time (min)	Num. packets
5	25 m	31
15	75 m	93
25	124 m	155
40	198 m	247
70	346 m	432

5. Discussion

One of the concerns about the two-board design can be the time it takes to transmit the data from one board to the other. To maximize the UART's communication protocol speed, we conducted an experiment where the baud rate was gradually increased. The highest stable baud rate was 74880 bps. As to our empirical findings, when it was further increased, the communication was prone to errors. With the UART communication's baud rate set to 74880 bps and a batch of weights consisting of 200 bytes or fewer, the transmission took 21 ms. This speed showed to be suitable and fast compared to the low data rate that is provided by the LoRa communication.

Transmitting a neural network model in a federated learning process over LoRa can take a long time, depending on the model size (Table 2). Methods for model quantization are important to reduce the bandwidth usage in low-capacity links, but also the number of federated learning rounds influences the time of the federated learning process. In our application, the federated learning process was designed to run a round once a certain amount of epochs on the other devices within the LoRa network had elapsed. For other applications, different rules and trade-offs may be more suitable, for instance pre-configuring to perform federated learning daily at a fixed time or making federated learning performance dependent, such as based on model quality.

In on-device training, the samples for the models to be trained are obtained from the sensors on the microcontroller board. Depending on the specific application, data may be obtained more regularly, for instance as a stream of data, or infrequently. Since there is no large storage capacity on IoT boards, once the sample is used for the training of the model, the sample is dropped. From this perspective, the amount of data for training does not affect the performance of the device in terms of memory used, since only that sample with which the model is currently trained is temporarily stored. However, as long as the model was only trained with a small amount of data, it may be less performing. Applying the decentralized federated learning approach which we presented in Section 3.5 can allow these nodes with few training samples to enhance their model while taking into account the trade-off between model performance improvement and communication cost, as described in Section 4.2.3.

In LoRa networks, collisions of packets can happen which lead to packet losses, as observed in Section 4.2.4. Many factors of the LoRa configuration, such as SF and payload size, can influence packet losses. Some factors can be configured at the proper nodes of the federated learning network, while others are external, such as the LoRa traffic that other nearby networks may produce. In order to detect collisions and the loss of packets, a CRC should be applied on the packet, which allows to validate that the received header and payload are correct. Furthermore, reliable messaging, such as explained in Section 3.5, either provided at the application or network level, should be used. With these means, it can be ensured for different types of network conditions that all the data has been received correctly by the destination node. The transmission time of a federated learning round increases with reliable messaging and also it depends on the network conditions. Congested networks lead to higher packet losses and hence re-transmissions, and larger diameters of networks are another factor that contributes to an increased transmission time.

Different from centralized federated learning, the design of serverless decentralized federated learning, which we applied in this work, is not yet a well-researched topic. We observed that depending on the algorithm used, suboptimal weights combination can happen in certain conditions. For example, when node A merges its model with node B, and then node C merges its model with nodes A and B, the resulting model will be more similar to that of the B node. The algorithm we introduced in Listing 3 does not take into account this information and could be extended to become aware of such a situation.

A larger federated learning network with a higher number of nodes will provide more opportunities for a node to obtain better performing models. The decentralized federated learning approach described in 3.5 allows each node to choose its participation in a federated learning round. Since each node can tailor this decision to its local conditions, a larger number of nodes does not automatically affect the operation of a node.

The memory consumption of a node that is a peer in a federated learning network is higher than that of a node that is only a client or a server, as we describe in Section 4.1. For being a server, the memory of an IoT board may limit a node to run a federated learning round with a large number of clients, where this number also depends on the size of the model that is exchanged. To address the memory limitations of this case, a node acting as a server could select only a subset of clients that fit its memory availability.

IoT applications with embedded machine learning on remote nodes and LoRaWAN communication have become deployable nowadays [37]. For these types of applications, the limit is the model update and the networking of the node through LoRaWAN. Federated learning over LoRa mesh networks is a step towards overcoming both limitations, where collaborative training addresses obtaining more performant machine learning models, and mesh networking capabilities to enable the communication of the nodes within the IoT network layer.

The computing capacity available in our distributed IoT application is minimalistic from the computing and communication resource usage point of view, consisting only of embedded systems boards interconnected over LoRa and without Internet access. While with such an infrastructure, the absolute machine learning capacity will always be limited compared to higher-end devices, the ratio between machine learning capacity and resource consumption, both in terms of energy consumption and material cost, might be very favorable in the proposed design and may fit well to low-energy AI designs of future societal needs.

6. Conclusions and future work

This paper proposed federated machine learning over LoRa communication, performed by embedded devices at the IoT layer. The work developed a system implementation by integrating a machine learning application with a LoRa mesh library. The code was deployed on two boards, one board taking care of the machine learning application and the other being used as a router within a LoRa mesh network.

The work evaluated several parameters related to model performance, bandwidth usage, and energy consumption. We analyzed the representation of the neural network weights with different data types to determine its potential for bandwidth saving in the LoRa communication link. Decentralized federated learning over LoRa showed to be beneficial for the accuracy of the local model at the active node but incurs a cost for the model communication and longer training times. While it was found that the approach is not suitable to deliver fast model training, requiring rather long training times due to the slow LoRa data transmission, it enables on-device training by federated learning within devices interconnected only by LoRa in the IoT layer.

This work has integrated diverse components in order to demonstrate the feasibility of the system implementation and the opportunity for distributed machine learning applications on LoRa-interconnected tiny devices. It opens directions for future work for analyzing in more detail several of the design choices. Decentralized federated learning could become a relevant technique to enable remote smart IoT devices to adapt or update the node's machine learning model. More research is needed about how the node can decide for updating while taking into account the trade-off between the expected benefit in model accuracy and the cost of the training process. Furthermore, additional machine learning applications should be experimented with to identify the common and specific design parameters of the system architecture.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported by the Spanish Government under contracts PID2019-106774RB-C21, PCI2019-111851-2 (LeadingEdge CHIST-ERA), PCI2019-111850-2 (DiPET CHIST-ERA).

References

[1] P.P. Ray, A review on TinyML: State-of-the-art and prospects, J. King Saud Univ. - Comput. Inf. Sci. 34 (4) (2022) 1595–1623, http://dx.doi.org/10.1016/j.jksuci.2021.11.019, URL https://www.sciencedirect.com/science/article/pii/S1319157821003335.

[2] V. Rajapakse, I. Karunanayake, N. Ahmed, Intelligence at the extreme edge: A survey on reformable TinyML, ACM Comput. Surv. (2023) (in press), https://doi.org/10.1145/3583683.

[3] F. Sakr, F. Bellotti, R. Berta, A. De Gloria, Machine learning on mainstream microcontrollers, Sensors 20 (9) (2020) http://dx.doi.org/10.3390/s20092638, URL https://www.mdpi.com/1424-8220/20/9/2638.

[4] Q. Zhou, Z. Qu, S. Guo, B. Luo, J. Guo, Z. Xu, R. Akerkar, On-device learning systems for edge intelligence: A software and hardware synergy perspective, IEEE Internet Things J. 8 (15) (2021) 11916–11934, http://dx.doi.org/10.1109/JIOT.2021.3063147.

[5] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) http://dx.doi.org/10.1145/3298981.

[6] H.G. Abreha, M. Hayajneh, M.A. Serhani, Federated learning in edge computing: A systematic survey, Sensors 22 (2) (2022) http://dx.doi.org/10.3390/s22020450, URL https://www.mdpi.com/1424-8220/22/2/450.

[7] J. Haxhibeqiri, E. De Poorter, I. Moerman, J. Hoebeke, A survey of LoRaWAN for IoT: From technology to application, Sensors 18 (11) (2018) http://dx.doi.org/10.3390/s18113995, URL https://www.mdpi.com/1424-8220/18/11/3995.

[8] M.A.M. Almuhaya, W.A. Jabbar, N. Sulaiman, S. Abdulmalek, A survey on LoRaWAN technology: Recent trends, opportunities, simulation tools and future directions, Electronics 11 (1) (2022) http://dx.doi.org/10.3390/electronics11010164, URL https://www.mdpi.com/2079-9292/11/1/164.

[9] R. Pueyo Centelles, F. Freitag, R. Meseguer, L. Navarro, Beyond the star of stars: An introduction to multihop and mesh for LoRa and LoRaWAN, IEEE Pervasive Comput. 20 (2) (2021) 63–72, http://dx.doi.org/10.1109/MPRV.2021.3063443.

[10] J.R. Cotrim, J.H. Kleinschmidt, Lorawan mesh networks: A review and classification of multihop communication, Sensors 20 (15) (2020) http://dx.doi.org/10.3390/s20154273, URL https://www.mdpi.com/1424-8220/20/15/4273.

[11] Meshtastic: Open source hiking, pilot, skiing and secure GPS mesh communicator, 2023, https://meshtastic.org/. (Accessed 6 March 2023).

[12] S. Niknam, H.S. Dhillon, J.H. Reed, Federated learning for wireless communications: Motivation, opportunities, and challenges, IEEE Commun. Mag. 58 (6) (2020) 46–51, http://dx.doi.org/10.1109/MCOM.001.1900461.

[13] T. Li, A.K. Sahu, A. Talwalkar, V. Smith, Federated learning: Challenges, methods, and future directions, IEEE Signal Process. Mag. 37 (3) (2020) 50–60, http://dx.doi.org/10.1109/MSP.2020.2975749.

[14] L.U. Khan, W. Saad, Z. Han, E. Hossain, C.S. Hong, Federated learning for internet of things: Recent advances, taxonomy, and open challenges, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1759–1799, http://dx.doi.org/10.1109/COMST.2021.3090430.

[15] D.C. Nguyen, M. Ding, P.N. Pathirana, A. Seneviratne, J. Li, H.V. Poor, Federated learning for internet of things: A comprehensive survey, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1622–1658, http://dx.doi.org/10.1109/COMST.2021.3075439.

[16] A. Mathur, D.J. Beutel, P.P.B. de Gusmão, J. Fernandez-Marques, T. Topal, X. Qiu, T. Parcollet, Y. Gao, N.D. Lane, On-device federated learning with flower, 2021, arXiv:2104.03042.

[17] F. Freitag, P. Vilchez, L. Wei, C.-H. Liu, M. Selimi, Performance evaluation of federated learning over wireless mesh networks with low-capacity devices, in: A. Rocha, C. Ferrás, A. Méndez Porras, E. Jimenez Delgado (Eds.), Information Technology and Systems, Springer International Publishing, Cham, 2022, pp. 635–645.

[18] J. Choi, Z. Wang, S. Venkataramani, P.I.-J. Chuang, V. Srinivasan, K. Gopalakrishnan, PACT: Parameterized clipping activation for quantized neural networks, 2018, arXiv:1805.06085.

[19] N.L. Giménez, F. Freitag, J. Lee, H. Vandierendonck, Comparison of two microcontroller boards for on-device model training in a keyword spotting task, in: 2022 11th Mediterranean Conference on Embedded Computing, MECO, 2022, pp. 1–4, http://dx.doi.org/10.1109/MECO55406.2022.9797171.

[20] K. Kopparapu, E. Lin, J.G. Breslin, B. Sudharsan, TinyFedTL: Federated transfer learning on ubiquitous tiny IoT devices, in: 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events, PerCom Workshops, 2022, pp. 79–81, http://dx.doi.org/10.1109/PerComWorkshops53856.2022.9767250.

[21] S. Disabato, M. Roveri, Incremental on-device tiny machine learning, in: Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things, AIChallengeIoT '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 7–13, http://dx.doi.org/10.1145/3417313.3429378.

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

[23] F. De Vita, G. Nocera, D. Bruneo, V. Tomaselli, M. Falchetto, On-device training of deep learning models on edge microcontrollers, in: 2022 IEEE International Conferences on Internet of Things (IThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics, Cybermatics, 2022, pp. 62–69, http://dx.doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics55523.2022.00018.

[24] N. Llisterri Giménez, M. Monfort Grau, R. Pueyo Centelles, F. Freitag, On-device training of machine learning models on microcontrollers with federated learning, Electronics 11 (4) (2022) http://dx.doi.org/10.3390/electronics11040573, URL https://www.mdpi.com/2079-9292/11/4/573.

[25] A.G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, C. Wachinger, BrainTorrent: A peer-to-peer environment for decentralized federated learning, 2019, arXiv:1905.06731.

[26] S. Savazzi, M. Nicoli, V. Rampa, Federated learning with cooperating devices: A consensus approach for massive IoT networks, IEEE Internet Things J. 7 (5) (2020) 4641–4654, http://dx.doi.org/10.1109/JIOT.2020.2964162.

[27] L. Witt, M. Heyer, K. Toyoda, W. Samek, D. Li, Decentral and incentivized federated learning frameworks: A systematic literature review, IEEE Internet Things J. 10 (4) (2023) 3642–3663, http://dx.doi.org/10.1109/JIOT.2022.3231363.

[28] B. Vejlgaard, M. Lauridsen, H. Nguyen, I.Z. Kovacs, P. Mogensen, M. Sorensen, Coverage and capacity analysis of Sigfox, LoRa, GPRS, and NB-IoT, in: 2017 IEEE 85th Vehicular Technology Conference, VTC Spring, 2017, pp. 1–5, http://dx.doi.org/10.1109/VTCSpring.2017.8108666.

[29] C. Ebi, F. Schaltegger, A. Rüst, F. Blumensaat, Synchronous LoRa mesh network to monitor processes in underground infrastructure, IEEE Access 7 (2019) 57663–57677, http://dx.doi.org/10.1109/ACCESS.2019.2913985.

[30] D. Vasconcelos, M.S. Yin, F. Wetjen, A. Herbst, T. Ziemer, A. Förster, T. Barkowsky, N. Nunes, P. Haddawy, Counting mosquitoes in the wild: An internet of things approach, in: Proceedings of the Conference on Information Technology for Social Good, GoodIT '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 43–48, http://dx.doi.org/10.1145/3462203.3475914.

[31] R. Berto, P. Napoletano, M. Savi, A lora-based mesh network for peer-to-peer long-range communication, Sensors 21 (13) (2021) http://dx.doi.org/10.3390/s21134314, URL https://www.mdpi.com/1424-8220/21/13/4314.

[32] R.P. Centelles, Towards LoRa Mesh Networks for the IoT (Ph.D. thesis), Universitat Politècnica de Catalunya, 2021.

[33] M. Vigil-Hayes, M.N. Hossain, A.K. Elliott, E.M. Belding, E. Zegura, Lorax: Repurposing LoRa as a low data rate messaging system to extend internet boundaries, in: ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies, COMPASS '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 195–213, http://dx.doi.org/10.1145/3530190.3534807, URL https://doi-org.recursos.biblioteca.upc.edu/10.1145/3530190.3534807.

[34] A.M. Cardenas, M.K.N. Pinto, E. Pietrosemoli, M. Zennaro, M. Rainone, P. Manzoni, A low-cost and low-power messaging system based on the LoRa wireless technology, Mob. Networks Appl. 25 (3) (2020) 961–968, http://dx.doi.org/10.1007/s11036-019-01235-5.
[35] J.M. Solé, S. Miralles Nogués, R.P. Centelles, F. Freitag, Demonstration of a library prototype to build LoRa mesh networks for the IoT, in: 2022 IEEE 42nd International Conference on Distributed Computing Systems, ICDCS, 2022, pp. 1328–1331, http://dx.doi.org/10.1109/ICDCS54860.2022.00150.
[36] J.M. Solé, R.P. Centelles, F. Freitag, R. Meseguer, Implementation of a LoRa mesh library, IEEE Access 10 (2022) 113158–113171, http://dx.doi.org/10.1109/ACCESS.2022.3217215.
[37] M. Altayeb, M. Zennaro, M. Rovai, Classifying mosquito wingbeat sound using tinyml, in: Proceedings of the 2022 ACM Conference on Information Technology for Social Good, GoodIT '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 132–137, http://dx.doi.org/10.1145/3524458.3547258, URL https://doi-org.recursos.biblioteca.upc.edu/10.1145/3524458.3547258.