future internet

MDPI

Article

FL-LoRaMAC: A Novel Framework for Enabling On-Device Learning for LoRa-Based IoT Applications

Shobhit Aggarwal *,† **and Asis Nasipuri** †

Department of Electrical and Computer Engineering, University of North Carolina at Charlotte, 9201 University City Blvd., Charlotte, NC 28223, USA; anasipur@charlotte.edu

* Correspondence: saggarw4@charlotte.edu

† These authors contributed equally to this work.

Abstract: The Internet of Things (IoT) enables us to gain access to a wide range of data from the physical world that can be analyzed for deriving critical state information. In this regard, machine learning (ML) is a valuable tool that can be used to develop models based on observed physical data, leading to efficient analytical decisions, including anomaly detection. In this work, we address some key challenges for applying ML in IoT applications that include maintaining privacy considerations of user data that are needed for developing ML models and minimizing the communication cost for transmitting the data over the IoT network. We consider a representative application of the anomaly detection of ECG signals that are obtained from a set of low-cost wearable sensors and transmitted to a central server using LoRaWAN, which is a popular and emerging low-power wide-area network (LPWAN) technology. We present a novel framework utilizing federated learning (FL) to preserve data privacy and appropriate features for uplink and downlink communications between the end devices and the gateway to optimize the communication cost. Performance results obtained from computer simulations demonstrate that the proposed framework leads to a 98% reduction in the volume of data that is required to achieve the same level of performance as in traditional centralized ML.

Keywords: Internet of Things; artificial intelligence; machine learning; federated learning; LoRa; LoRaWAN

Citation: Aggarwal, S.; Nasipuri, A. FL-LoRaMAC: A Novel Framework for Enabling On-Device Learning for LoRa-Based IoT Applications. *Future Internet* **2023**, *15*, 307. https://doi.org/10.3390/fi15090307

Academic Editors: Panagiotis Papageorgas, Dimitrios Piromalis and Dionisis Kandris

Received: 28 July 2023
Revised: 28 August 2023
Accepted: 5 September 2023
Published: 10 September 2023

1. Introduction

The explosive growth of Internet of Things (IoT) networks has led to an unprecedented increase in the amount of data being generated from a diverse set of networked sensors. With estimates predicting over 55 billion connected devices generating a staggering 80 zettabytes of data by 2025 [1], the potential for artificial intelligence (AI) to unlock insights and improve the management and quality of life is enormous. Machine learning (ML) is the AI tool that provides this ability by developing efficient processes for identifying hidden patterns in the data generated by IoT devices for improved decision making and rapid automated responses. Machine learning tools can immensely help to detect anomalies or project future trends in IoT applications.

In recent years, IoT and ML principles have been used in conjunction in multiple domains, such as agriculture, manufacturing, healthcare, smart cities, etc. [2,3]. Examples include precision agriculture, smart traffic, energy usage monitoring and control, smart home, public safety, weather prediction, process automation, and many others. In healthcare scenarios, the effectiveness and cost benefits of using small low-cost remote health monitoring systems have been encouraging to users [4,5]. With advancements in sensor technology, the sensors used for obtaining data such as the electrocardiogram (ECG), blood sugar, blood pressure, heart rate, SPO2, etc., have become cheaper and smaller to the extent that they can be embedded into wearable devices such as watches, pendants, vests, etc.

Consequently, there is increasing interest in developing ML tools for healthcare applications. A large number of organizations, such as Microsoft [6], Tempus [7], Beta Bionics [8], Insitro [9], etc., are also involved in these efforts.

While multiple solutions are available that can serve as the backhaul technology for IoT devices, low-power wide-area networks (LPWANs) gained significant interest due to their wide wireless coverage, large capacity, low cost, and simple infrastructure. In this work, we focus on LoRa, which is a prominent LPWAN standard, as the backhaul network for IoT connectivity [10–12]. LoRa uses a proprietary chirp spread spectrum technology with the provision to use multiple spreading factors and power levels that provide robustness to interference as well as long-range connectivity with adaptive data rate (ADR) control. The network protocol suite LoRaWAN has been developed to use LoRa technology for implementing an LPWAN. Multiple orthogonal spreading factors (SFs) and adaptive power rates enable end devices in a LoRaWAN network (see Figure 1) to effectively transmit data to the gateways. However, the key concerns with developing ML applications using IoT devices in an LoRaWAN include privacy considerations that must be met for sending data with sensitive personal information to a central server and meeting performance requirements for transmitting large volumes of data over the wireless network.

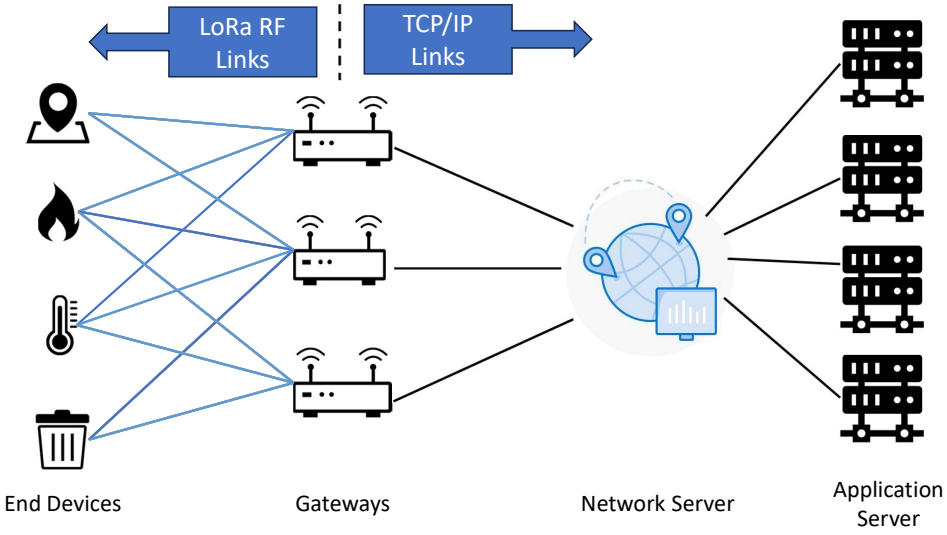


Figure 1. LoRaWAN architecture.

To overcome these challenges, in this paper, we propose to apply a federated learning (FL) framework, which is an effective technique in AI to address privacy concerns. As opposed to traditional ML, where all data are transmitted to a central location for model development and testing, in FL, the model is trained locally, with rounds of updates with a central server. While this avoids any transmission of raw data with potential privacy limitations from the end nodes, using an FL framework in an LPWAN also poses its own challenges. These involve the development of an efficient mechanism for two-way data communication between the end-devices and the server and meeting the communication and computation requirements of FL. To this end, we developed FL-LoRaMAC, which incorporates an asynchronous medium access control (MAC) for downlink communication for LoRaWAN. We introduce a novel SF allocation scheme that was presented in our preliminary work in [13], which, in association with traditional optimization tools such as principal component analysis (PCA) and pruning of the AI model, significantly reduces the communication requirements in the network. The communication requirements of FL using the proposed framework for its implementation in a representative healthcare application using LoRa-based networks are examined numerically.

We present the design considerations and performance evaluations of FL-LoRaMAC by considering the problem of the anomaly detection of ECG signals as a candidate IoT-based

AI application. This is motivated by the fact that, currently, ECG signals may be obtained from low-cost portable devices such as smartwatches that utilize photoplethysmography (PPG) technology [14]. The easy availability of ECG data from FDA-approved sensors integrated into smartwatches by manufacturers such as Apple, Google Pixel, Fitbit, and Samsung has paved the way for AI applications. A trained model with ECG datasets can have the ability to detect or predict heart conditions. However, the model must be able to differentiate between healthy (normal) and unhealthy (abnormal) ECG signals to detect health conditions effectively. It must be noted that the proposed approaches would be useful for a large number of other applications of AI in IoT networks as well.

To summarize, the major contributions presented in this paper are:

- An evaluation and comparison of the performance of ML and FL using a representative healthcare application.
- The development of a novel framework used to implement FL for LoRa-based IoT platforms. The framework includes an MAC protocol that enables two-way half-duplex communication between the central server and the end devices while optimizing power consumption.
- The introduction of a novel SF allocation scheme that assigns SFs for uplink data transmissions to provide differential priorities of the parameters being transmitted.
- An evaluation of the performance of the proposed AI scheme, including accuracy, precision, and recall under channel packet loss rate, PCA sparsification, and number of devices.

The organization of the paper is as follows. In Section 2, we summarize related work. The problem statement and design considerations are presented in Section 3. The proposed framework is described in Section 4. The simulation setup for performance evaluations is described in Section 5 and results are presented in Section 6. Section 7 includes conclusions and future work.

2. Related Work

In this section, we explore real-world applications of machine learning, along with the methods and algorithms employed. Furthermore, we delve into the ongoing research and collaboration between IoT and AI in order to shed light on the existing work in this field.

The utilization of machine learning algorithms for the detection of oil spills in radar images is described in [15], which presents an in-depth examination of the challenges intertwined with this application. The study also proposes novel strategies for effectively tackling these obstacles. To substantiate their claims, the authors conduct comprehensive experimental analyses that specifically focus on two pivotal aspects of machine learning: batched and imbalanced training sets. Within this context, the paper introduces two novel algorithms, namely SHRINK and one-sided selection, that demonstrate remarkable efficacy in controlling the false alarm rate. Hence, this paper not only sheds light on the practical implementation of machine learning techniques for oil spill detection but also provides solutions to overcome the associated challenges.

The application of support vector machines, decision trees, and classification methods for detecting land cover changes and mapping in rural areas is presented in [16]. Three objectives were achieved: identifying suitable bands for classification, comparing the performance of the methods, and detecting changes in land cover. Data preprocessing was performed using ERDAS IMAGINE 9.1 and ENVI 4.5. The results show that the decision tree algorithm outperformed the other methods in terms of accuracy and performance. Failure degradation was also estimated, providing insights into method limitations. The paper highlights the effectiveness of decision trees for land cover change detection in rural areas and offers guidance for future improvements.

The use of support vector machines (SVMs) as a regression technique for estimating soil moisture with remote sensing data was explored in [17]. Ten sites in the western United States were studied. SVMs outperformed multiple linear regression (MLR) and

artificial neural networks (ANNs) in accuracy and performance. The findings contribute to improved soil moisture estimation and management in the studied regions.

Human health benefits much from the advent of IoT and AI tools. In [18], the authors proposed a highly reliable method for measuring blood pressure without the need of putting cuffs. The proposed model uses a combination of information from the ECG signal and PPG to measure blood pressure. The study first establishes a relationship between blood pressure and the ECG. This enables the user with the ECG sensor to potentially measure blood pressure without requiring any additional device.

The authors of [19] proposed a novel solution for predicting heart diseases using the cascaded deep learning model in the fog computing environment. Their solution makes use of data from multiple sensors that provide data about daily activity that are fed to an ensemble classifier. The classifier is hosted in the fog environment and computations are performed in a decentralized manner. The proposed approach achieved a prediction accuracy of 95%.

A transfer learning methodology used to decrease the computational resources needed to train a deep neural network for a reinforcement learning (RL) problem was presented in [16]. The deep neural network is trained on a varied set of meta-environments to acquire broad domain knowledge, which can then be transferred to test environments, and only the last few fully connected layers are trained. The performance of the algorithm is evaluated in terms of mean safe flight, and it was observed that the network's performance is comparable to that of training the network end-to-end, while significantly decreasing the latency and energy consumption by 1.8 and 3.7 times, respectively.

The authors of [20] proposed an ECG classification system that makes use of LoRa and fog computing. The system collects data using the ECG sensors and transmits the collected data to the fog layer using LoRa communication. The fog layer uses deep learning to perform the classification and sends the information over a 4G cellular network to be analyzed by the doctor. Although the article claims that the data rate of LoRa could be acceptable for the transmission of signals such as an ECG, they did not use real-time data transmission in their system. They used an existing dataset [21] for training and testing purposes. At the fog layer, the system uses two models: one to perform the analysis in the time domain and another in the frequency domain. The outputs from both models were merged to obtain more concrete decisions. Their preliminary results show that the proposed system is capable of identifying abnormal heart rhythms.

The majority of studies in the field have focused on improving AI models and preparing data for analysis. However, these studies typically assume that the data are stored on a central server and the model is trained there. After training, the model is sent to other devices for making predictions. However, these studies have overlooked important issues related to network challenges during data collection and privacy concerns with centralized training. The network challenges encompass a wide array of issues, ranging from data transmission efficiency and latency to bandwidth limitations and reliability. Overlooking these challenges can negatively affect the overall performance and scalability of AI systems, hindering their real-world applicability. Additionally, the transfer and storage of substantial amounts of potentially sensitive data in a single location create vulnerabilities and increase the risk of privacy breaches.

3. Problem Statement and Design Considerations

In this section, we present the basic principles of machine learning and the challenges associated with its applications in IoT networks. The proposed approach to overcoming these challenges and the associated design considerations are also explained.

In a typical machine learning scenario, as shown in Figure 2, the data from all the sources (sensors) are sent to the central server via a gateway, where the model resides. The model is trained on the collected data from various sources. Once the model is trained, the test data are sent to the central server where decisions or predictions are made. It can be inferred that, in both the training and testing phases, the data must be collected at the

central server. In the IoT ecosystem, doing so poses some challenges that are discussed in the following subsection.

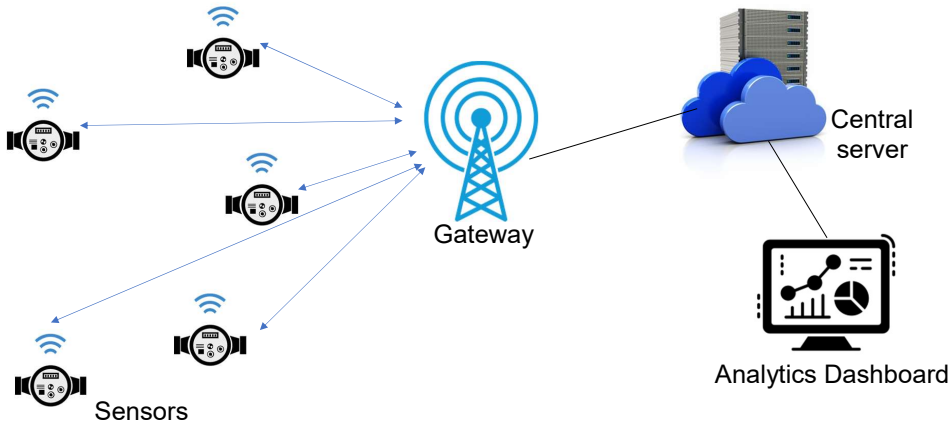


Figure 2. Illustration of a typical machine learning model where all data are transmitted from IoT sensors to a central server for model development and testing.

3.1. Challenges Faced by ML

The key challenges faced by the IoT ecosystem for the implementation of typical machine learning techniques with respect to sending data to the central server include:

- *High data volume:* A sensor node may capture data once every few minutes but the gateway is connected to multiple such sensor nodes. All the data are collected by the gateway and sent to the central server. This can lead to a significant amount of traffic in the network that can result in transmission losses and high latency issues in a real-time environment.
- *Data privacy:* Typically, the machine learning algorithms run on third-party cloud servers. During both the training phase as well as the testing phase, the data need to be sent to the server where the model resides. These data can be private to the individuals. These data can be exploited by third-party service providers; hence, regulatory organizations like California Consumer Privacy Act (CCPA) in the US impose many restrictions in terms of the privacy of sensitive customer data. These may lead to regulatory problems in many applications, such as healthcare.

In IoT applications that involve private user data, the need to send data to a central server for machine learning purposes can be a challenge. This challenge can be addressed by using federated learning, which is an artificial intelligence technique that allows learning to be performed without sending data to the central server.

3.2. Federated Learning

Contrary to the centralized machine learning approach, FL assumes a distributed approach that allows for training AI models across decentralized devices without transferring data to a central server. FL is especially useful in IoT applications that involve private user data since it does not require the data to be sent to a central server, addressing privacy and security concerns. This paper aims to explore the possibility of leveraging federated learning to decrease the amount of data that need to be transmitted while maintaining a comparable performance. The overall architecture of FL is shown in Figure 3.

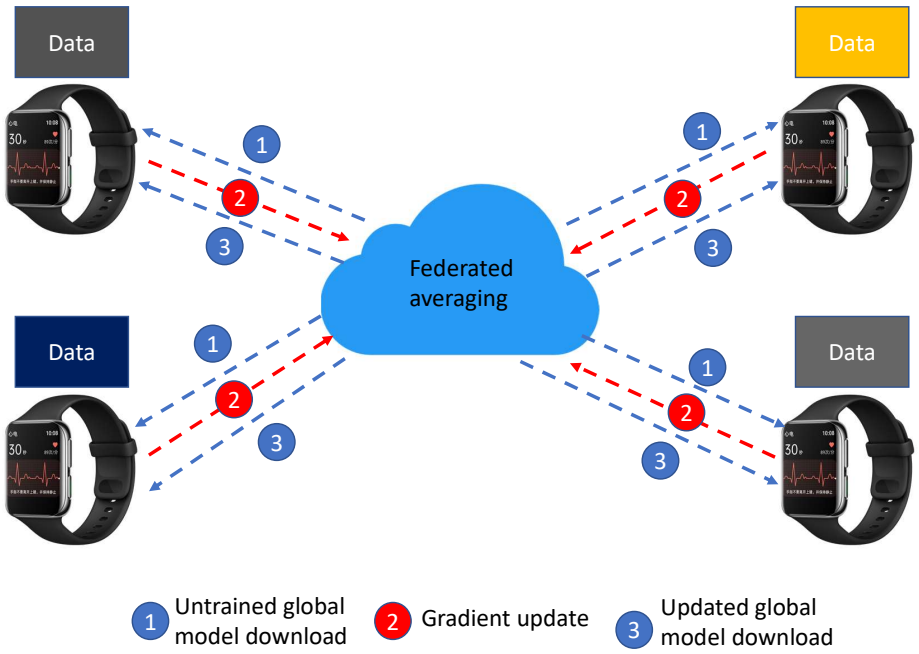


Figure 3. Illustration of a federated learning model, where the initial model is sent to end devices for training using their respective local data. The locally trained models are transmitted to the FL server for aggregation and the process is repeated over multiple rounds of local training and model aggregation.

In federated learning, instead of the end devices transmitting all their data to the central server, the central server sends the global model (initially untrained) to the participating end devices. The end device trains the received model based on its locally collected data. This training is carried out for a certain number of *epochs* similar to the training in the case of machine learning, but it is performed on the edge and based only on the data collected by the end device itself. Once the local training is complete, the model weights, also known as model parameters, are sent to the central server. These local model weights are numerical vectors that do not provide any information about the type or kind of data that was used for training. At no point do the local data leave the end device. Once the central server receives the model parameters from all participating end devices, it performs a federated averaging on all the received model parameters and outputs an improved global model. This entire cycle of sending the global model, local training, sending local updates, and obtaining an updated global model by federated averaging is termed as a *communication round*. Federated learning is performed for a certain number of communication rounds until the desired performance is achieved. Once the training is complete, the global model is sent to all end devices that are fed with test samples, and its performance is evaluated based on the predictions that it makes. In federated learning, under no circumstances do the local data leave the edge device; hence, the data privacy is preserved at all times.

3.3. Problem Statement

To support the federated learning process, the network infrastructure should enable independent two-way communication capability between the end devices and the central server, providing uplink capabilities for sending locally trained weights from the end devices to the central server, also known as the gradient update phase, and the downlink capabilities for transmitting the updated global model from the central server to the end devices. The objective of this work is to develop a framework for meeting both uplink and downlink communication requirements for the FL model.

Figure 4 illustrates the three classes of operation defined under LoRaWAN specifications [22]. These classes provide independent uplink communication from the end devices to the central server but pose some challenges to downlink communication design consid-

erations. The class-A mode of operation prioritizes uplink communications and only opens two short receive windows for downlink after the uplink transmission. If the device does not receive any data during these receive windows, the device goes to sleep and will not open a receive window until after it sends another uplink. The class-B mode of operation opens additional time synchronized receive windows but requires additional resources like a GPS, and real-time clocks (RTCs) for the synchronization. This adds to the complexity of the system. Class-C is capable of fulfilling all the communication requirements for federated learning. However, it is the most power-hungry mode of operation and hence is not advised for battery-powered devices.

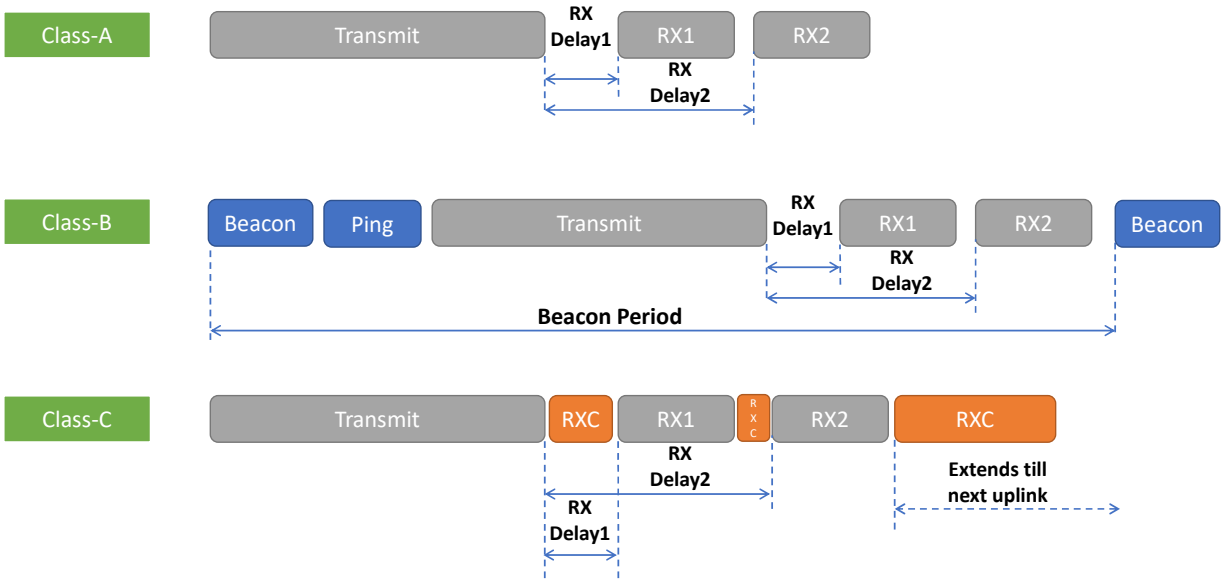


Figure 4. LoRaWAN classes of operation.

Firstly, we observe that none of the above classes of operation fully meet the requirements for FL for energy-constrained IoT devices. The downlink communication modes are either inadequate (Class-A and Class-B) or too energy-hungry (Class-C) for FL operations using low-cost IoT devices. Hence, one of the first objectives of this work is to develop an MAC that would be useful for efficient downlink transmissions of the global models from the gateway to the end devices. We accomplish this by developing an asynchronous transmission scheme using long preambles as described in the next section. Secondly, even the uplink communication faces performance challenges when the volume of traffic exceeds a limit. Since LoRaWAN uses a simple ALOHA-based MAC, performance studies have revealed that, when the number of active devices increases beyond a few hundred on the same channel and the same SF, the packet delivery rate degrades significantly [23–25]. Moreover, the model parameters for FL typically have non-uniform significance in terms of their performance for anomaly detection. This indicates the need for QoS considerations for transmission, which is not included in the legacy LoRaWAN design. However, LoRa provides the flexibility to use alternative MAC implementation as long as their proprietary physical layer protocol is used. The following section presents the proposed framework, referred to as FL-LoRaMAC, which includes an MAC protocol that tries to address the above communication requirements without requiring additional resources or complexity.

4. Proposed Framework: FL-LoRaMAC

This section describes the proposed framework FL-LoRaMAC that satisfies the communication design considerations for the implementation of various steps of federated learning. It includes guidelines for the global model downloads, local gradient updates, and downloading the updated global model. Additionally, it also describes how these

gradients are processed on the end devices as well as on the centralized server and some mechanisms to optimize communication bandwidth.

The working of FL-LoRaMAC is divided into three parts:

1. Network joining;
2. Proposed MAC for gradient updates;
3. Gradient processing.

4.1. Network Joining

In FL-LoRaMAC, all devices taking part in the training need to join the LoRa network first. The joining procedure is similar to that of the legacy LoRaWAN protocol as mentioned in [22] with some additional steps and information in join messages.

Whenever a device is powered ON, it sends a join request 'J_Req' to the network server via an LoRa transceiver. Once the J_Req is sent, the device waits for the join response 'J_Res'. If the end device receives the J_Res, it configures itself according to the J_Res.

On the network server side, a timer termed as the join request timer starts periodically. If the J_Req from the end device is received before this timer is expired, the network server prepares a J_Res. The network server has access to the information regarding the AI model architecture; we termed this information as 'MODEL_INFO'. It also contains information regarding the downlink channels and spreading factor on which the global model parameters will be broadcasted (DL_INFO), the information about the data fragments that will be sent (FRAG_INFO), the information about how frequently the device should perform the listening, and for the duration (LISEN_INFO). The network server will also send MODEL_INFO, DL_INFO, FRAG_INFO, and LISTEN_INFO along with the legacy LoRaWAN J_Res. All this information will be used by the end devices to configure themselves to communicate gradient updates as described in Section 4.2. Once the timer is expired, no J_Res for the end devices will be sent.

4.2. Proposed MAC Layer for Gradient Updates

In the federation construction phase, a patented model (untrained) is present at the central server. This model needs to be transmitted to the end devices for training. In order for end devices to receive this model, they must have their transceivers in listening mode while the server is transmitting. However, this is hard to achieve since the end devices are not synchronized to the network clock and their transceivers must be put to sleep mode in order to conserve power.

In order to achieve the goal mentioned above, we propose an elongated preamble approach. Under this approach, each end device joins the network according to the network joining procedure discussed in Section 4.1. Once the joining procedure is complete, the end device will configure itself according to the information received in the J_Res. It will also generate the local model according to the MODEL_INFO. Once the end device is configured, it will periodically open receive windows for receiving the global model parameters according to the DL_INFO, LISTEN_INFO, and FRAG_INFO as shown in Figure 5. If the end device does not hear any LoRa preamble, it will close the receive window and the transceiver will go to sleep until the next period to conserve power. If the end device finds a preamble, it will continue to listen to the packet.

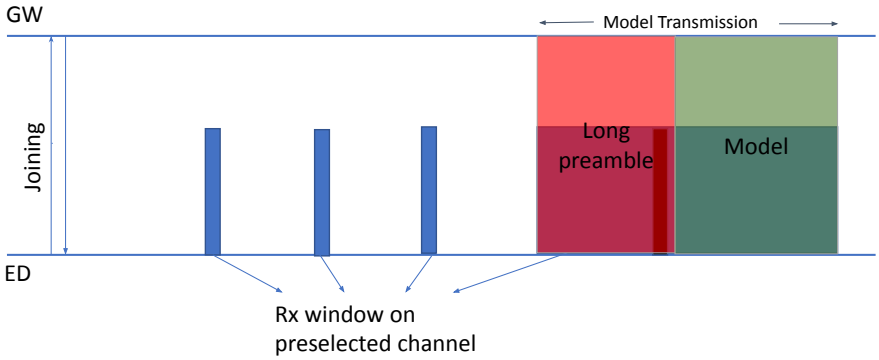


Figure 5. Illustration of the elongated preamble approach for downlink communication.

The gateway will periodically transmit the global model according to the channel and spreading factor in DL_INFO. The transmission will consist of an elongated preamble according to the FRAG_INFO. The length of the preamble is kept slightly longer than the periodicity of the end device at which it is opening the receive windows as shown in Figure 5. This elongated preamble enables the end devices to receive the transmission without being synchronized with the gateway. Once the end device receives all the fragments of the global model, the fragments will be serialized according to the frame numbers. If the end device encounters missing frames while serializing, the retransmission of those frames will be requested; otherwise, it will send an acknowledgment to the gateway indicating that the device received all the global model parameters. After this, the end device will stop opening the receive windows.

For the duration until the model is not received, the end device will keep collecting the data locally, and once the model is received, it will be trained on the data collected by the end device.

The end device trains the model that it received from the network server. Once the local training cycle is complete for a certain number of epochs, the local weights must be sent to the server for averaging. The model parameters must be packed into various data fragments to achieve this. Once the fragments are created, the end device transmits these fragments one by one using the uplink transmissions. The process of the creation of fragments is explained in Section 4.3. After fragment transmission is complete, the end device will follow the elongated preamble approach to receive the updated global model.

4.3. Decentralized Training and Model Aggregation

Section 4.2 provides details about the communication of the model parameters. This section explains how the packets containing model parameters are generated and processed when they arrive at the end device or at the network server. The entire process of FL-LoRaMAC taking place at the end device and the network server is illustrated using flowcharts in Figures 6 and 7.

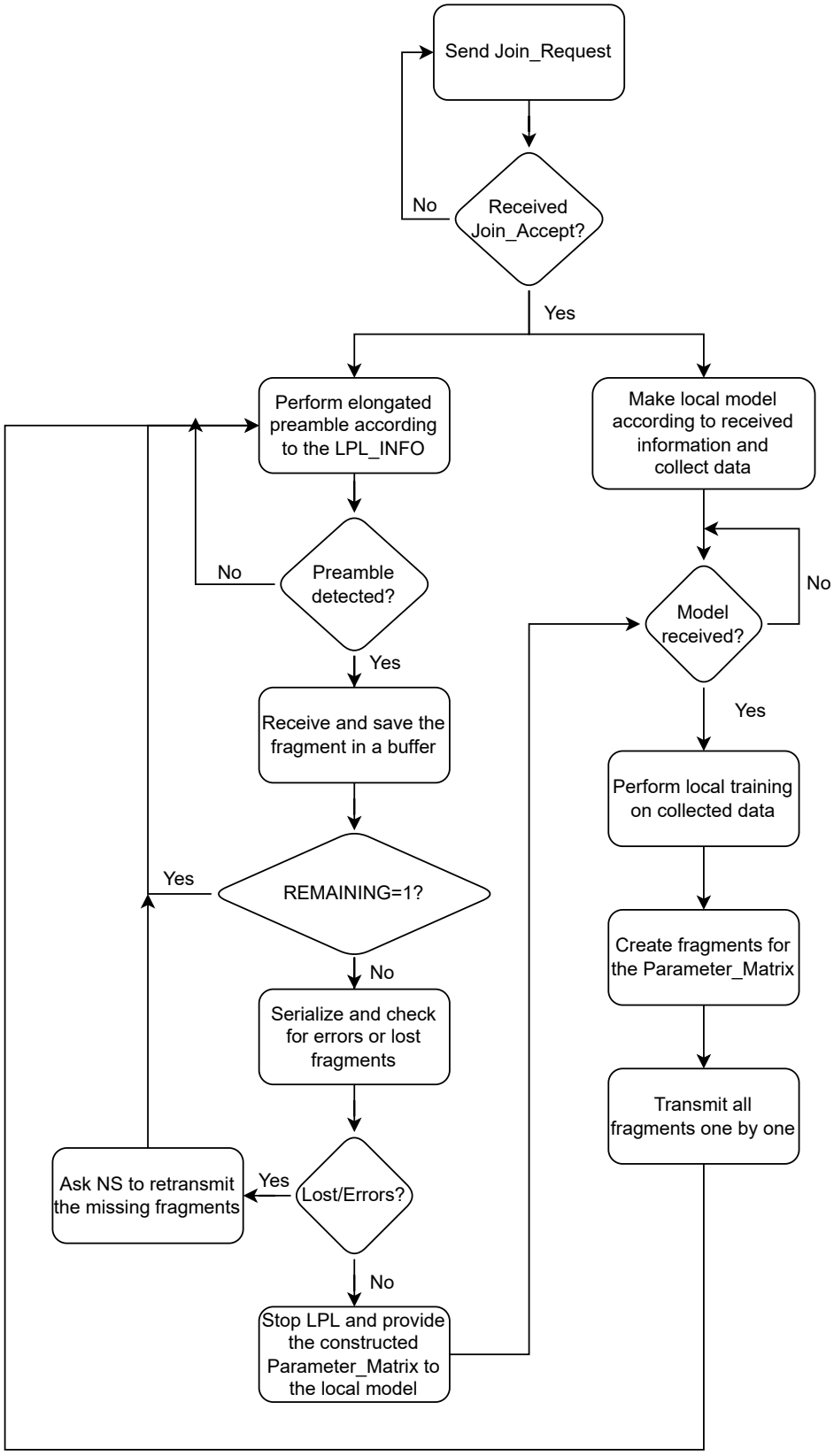


Figure 6. Operations of FL-LoRaMAC at the end device.

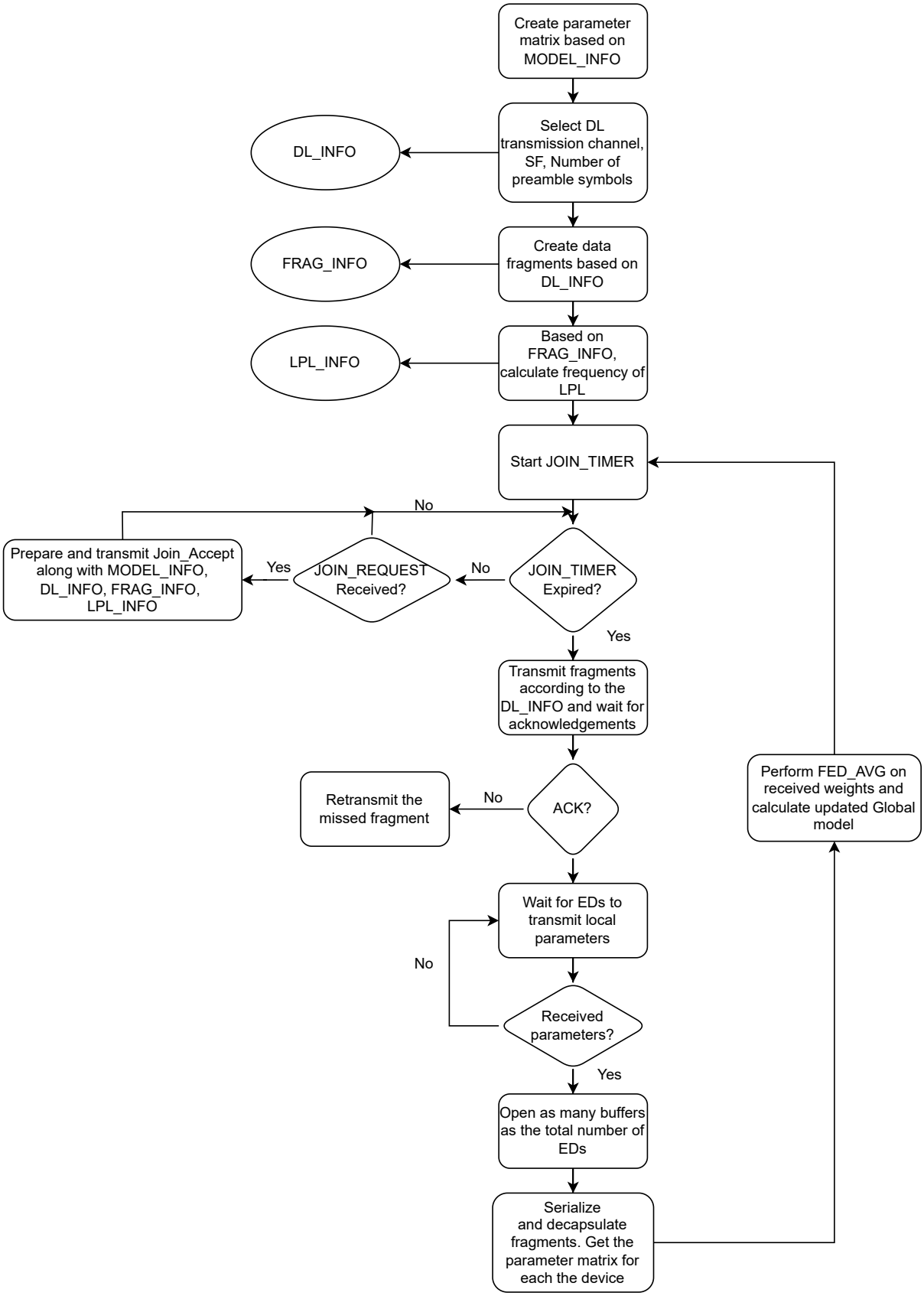


Figure 7. Operations of FL-LoRaMAC at the network server.

First, we will look at the decentralized training that happens at the end device as shown in Figure 6. The end devices participating in the federated learning process will receive the MODEL_INFO. This contains detailed information about the model. For instance, in the case of a neural network, it contains information about the number of neural layers and the number of neurons in each layer that are present in the model. The end device compiles a local model based on this information. As soon as the end device receives all the global model parameters, they are saved into its local model parameters. The global model is not received as a single frame but as a number of data fragments. These fragments are saved in a buffer, serialized, and checked for any missed packets. If there are missing fragments, then the re-transmission of missing fragments is requested.

Otherwise, once the device has enough data to train the model, the model training starts on these local data. Once the training for a certain number of epochs has elapsed, the updated local model parameters are sent to the central server for averaging using LoRa uplink transmissions. As discussed earlier, these model parameters are sent in fragments. These uplink data frames are constructed by flattening the parameter matrix and packing the pre-determined number of parameters in the frame. Also, each frame includes a frame number for identification and serializing purposes.

When the network server starts receiving uplinks from any of the end devices, it reserves a buffer for the end device's local model parameters. Once all the fragments are received from the end device, the fragments are serialized based on the frame numbers. Some of the fragments from the end devices may be lost in the RF environment due to collisions with other devices' communication. Using the frame numbers, the network server can detect the loss of fragments. During serialization, all the information except the model parameters is removed and, if any of the frames are missing, the lost parameters are set to zero. After this serialization, the same flattened matrix (with losses) will be formed from which the uplink frames were created by end devices. From this matrix, the parameter matrix is constructed.

Once the local updates from all the participating end devices have been received, the server performs federated averaging on the constructed parameter matrices. The output of the averaging provides an updated and intuitively better global model. Following the same procedure of flattening and creating fragments for transmitting, this global model is again sent to the end devices for subsequent training rounds using the elongated preamble approach via LoRa downlink transmissions.

4.4. Optimizing Communication Bandwidth

The artificial intelligence models are quite large in terms of the number of parameters in the weight matrix. However, LoRa is a low-bandwidth and low-data-rate communication technology. Hence, the transmission of these model parameters to the server will require a large amount of time and may consume all the network bandwidth. Therefore, the total transmission time and communication bandwidth used need to be optimized without significantly affecting the model's performance. To achieve this, we propose the framework depicted in Figure 8 that uses principal component analysis (PCA) and model pruning in conjunction with a spreading factor allocation scheme that is explained in subsequent sections. PCA is a popular dimensionality reduction method that is widely used in ML to reduce large datasets [26].

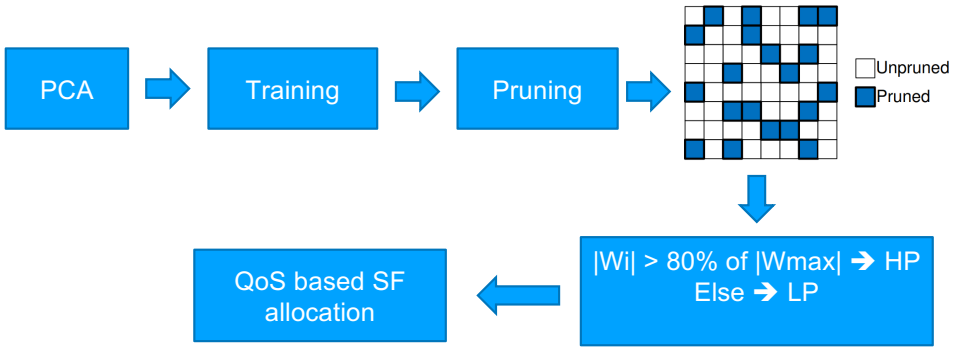


Figure 8. Illustration of the procedure for optimization of communication cost in FL-LoRaMAC.

4.4.1. Model Pruning

In a neural network, not all parameters are equally significant [27]. In other words, not all the connections between neurons of fully connected layers contribute equally to the model's prediction. The pruning technique can be used to reduce the complexity and time of execution by removing the insignificant connections between neurons. This is carried out by modifying the parameter values related to those connections to zero. The output of pruning is a sparsified weight matrix.

Typically, pruning is accompanied by quantization or compression or both to obtain a smaller parameter matrix in terms of bytes, but this also increases the computation cost. In this particular scenario, training is performed on resource-constrained devices. Applying these methods to those devices will result in significant overhead. Instead, we can choose not to communicate these pruned parameters or send them with lower priority.

4.4.2. Qos-Based Assignment of SF for Differential Priorities

The significance of model parameters can be established based on their magnitudes. The pruning algorithm sparsifies the parameter matrix by pruning the parameters closer to zero. Hence, it can be assumed that a higher magnitude means more significance.

If all the parameters are sent using the same priority level, there is an equal chance of data being lost for all. If the high-significance parameters are lost, then either retransmissions or more communication rounds will be required by the system to reach convergence. In order to reduce the loss of significant parameters due to interference, we propose a mechanism that employs a differential SF allocation scheme that changes the collision domain for the packets carrying parameters of high significance. This QoS-based SF distribution is described in detail in our earlier work [13]. For ease of understanding, here, we explain the concept using two levels of priorities for the packets carrying model parameters here as follows.

- *High priority:* High priority is given to the packets carrying any number of significant parameters in the weight matrix. These parameters must be delivered to the receiver with the highest probability of success (say >90%).
- *Low priority:* Low priority is given to the packets carrying the non-significant and less significant parameters in the weight matrix. These parameters can be delivered with the highest possible probability of success that the network can provide.

The proposed QoS-based SF allocation scheme is based on the principle of distribution of transmissions on appropriate SFs to meet their performance expectations. In LoRa, SFs determine the lengths of the chirp symbol and are theoretically orthogonal to each other. Hence, transmissions on different SFs, which range from 7 to 10, are characterized by different times on-air, data rates, and, consequently, different sensitivities. The LoRa SFs available in the United States and their corresponding characteristics are listed in Table 1. Due to varying sensitivities, LoRaWAN traditionally uses lower SFs for devices closer to the gateway and increasing SFs for increasing distances from the gateway. In our earlier work [13,28], we demonstrated that, within a shorter transmission range where

multiple SFs can be used, (a) optimum distribution of transmissions over multiple SFs can increase network capacity, and (b) controlled allocation of devices in specific SFs within the range can essentially provide guaranteed QoS in terms of the expected packet delivery rate (PDR). For instance, if there are a total of p high-priority devices and $p < d_{10}$, where d_i is the maximum number of devices that can be sustained with a given PDR (as defined by the corresponding QoS) in SF-i, $i = 7, 8, 9, 10$, then all p high-priority devices can be allocated to SF-10 and other low-priority devices can be allocated to other SFs, without any guarantees for their PDR. The theoretical limit d_i can be calculated using MAC capacity analysis for ALOHA [13]. Moreover, there exists an optimum distribution of SFs that can maximize the total capacity within any transmission range from the gateway where multiple SFs can be assigned simultaneously (although, traditionally, SFs are allocated in increasing order with increasing distances from the gateway as indicated in Table 1). For instance, in the region where it is possible to use both SFs 9 and 10, the distribution of end devices over SF-9 and SF-10 in the ratio 64:36 maximizes the capacity [28].

Table 1. LoRa spreading factors.

Spreading Factor (UL for 125 KHz)	Physical bit Rate (bits/sec)	Transmission Range (Depends on Terrain)
SF7	5470	2 km
SF8	3125	4 km
SF9	1760	6 km
SF10	980	8 km

Based on this principle, we propose a QoS-based SF allocation scheme that is described as follows (similar to scheme SFA-1 in [13]):

1. If 36% of $p < d_{10}$, then p devices will be distributed according to a 64:36 ratio between SF9 and SF10, respectively, and all remaining devices will be distributed according to a 64:36 ratio between SF7 and SF8, respectively.
2. If 36% of $p > d_{10}$, then d_{10} number of devices will be configured on SF-10 and the remaining $p - d_{10}$ devices will be configured on SF-9, given that $p - d_{10} < d_9$. All other devices will be distributed between SF-7 and SF-8 according to the 64:36 ratio.
3. If $p > d_9 + d_{10}$ but $p < d_8 + d_9 + d_{10}$, then all the low-priority devices will be configured on SF7 and high-priority devices will be configured on SF-8, SF-9, and SF-10.
4. If $p > d_8 + d_9 + d_{10}$, the required PDR x will decrease for HP devices and the values of d_8, d_9, and d_{10} for this new x will be found.
5. All p devices will be configured on SF-8, SF-9, and SF-10 in optimum fraction. The thresholds d_8, d_9 and d_{10} must be maintained at all times.

There can be other ambient devices in the network running some other applications, transmitting packets with higher priorities selected based on some criteria. Using the network configuration, the maximum number of devices that can be supported by each spreading factor can be found and the spreading factors can be allocated to different packets according to the top-down approach [13].

5. Simulation Details and Parameter Selection

To evaluate the performance of the proposed framework and the efficiency of federated learning to make accurate predictions, the proposed approaches were applied to an ECG anomaly detection application. Poisson analysis for the ALOHA-based network was used to evaluate the performance of the networking layers. Federated learning was implemented in Python, which ran on top of the networking layer.

All the codes for evaluation were executed on a machine configured with Intel(R) Core(TM) i5-8250U CPU clocked at 1.60GHz, no GPU, and a single 8 GB DDR4 RAM. The

ECG-5000 dataset was used for training and testing purposes [29]. The original ECG-5000 dataset is a 20 h long ECG downloaded from Physionet. The dataset was then pre-processed to extract each heartbeat and make them equal in length. The dataset contains 5000 labeled samples of ECG data. Each sample consists of 140 data points and a label. The value of the label is either 0 or 1, representing abnormal or normal heartbeats, respectively. A dense neural-network-based autoencoder was trained on the dataset and was used to make the predictions.

5.1. Network Setup and System Model

We assume a network where a large number of end devices are considered to be placed uniformly in a circular LoRa cell with the gateway placed at the center. It is assumed that, at any moment, the gateway has enough resources available to demodulate any number of valid receptions. The network server is connected to the application server. The application server is assumed to contain the untrained global model and it is also responsible for performing the federated averaging on parameters received from the nodes.

Out of all end devices, we assume that five devices are running the ECG anomaly detection application, with other devices potentially used for other applications. All five of those devices will be taking part in the federated learning process. All end devices share the same communication channel and gateway, and hence can interfere with each other's transmissions.

The training dataset consists of 80% of the whole dataset, and the remaining 20% was used for testing purposes. To simulate the real-world scenario, the data samples from the training subset were equally and randomly distributed among the end nodes participating in learning as shown in Figure 9. In a real world scenario, these data will be collected by the devices from local sensing.

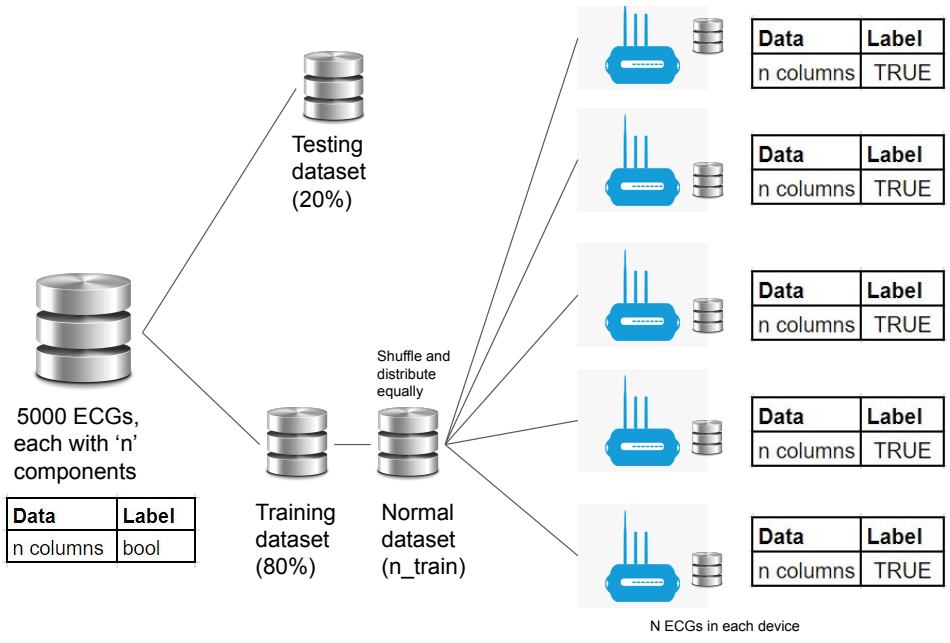


Figure 9. Illustration of dataset division.

5.2. Selection of Hyper-Parameters

The selection of model parameters is a critical issue that must be based on the trained model performance, cost of communicating data and/or parameters of the model, and computation complexity. Wherever pertinent, the model training measures, namely recall, precision, and accuracy, were used for evaluating the model's performance to make predictions.

Recall is defined as the ratio between the number of samples correctly predicted as positive to the total number of positive samples. In the case of ECG anomaly detection, recall can be formulated as

$$Recall = \frac{Number\ of\ samples\ correctly\ predicted\ as\ abnormal}{Total\ abnormal\ samples} \quad (1)$$

Precision is defined as the ratio of positive samples to the total number of positive predictions that the model made. In the case of ECG anomaly detection, precision can be formulated as

$$Precision = \frac{Total\ number\ of\ abnormal\ samples}{Number\ of\ samples\ predicted\ to\ be\ abnormal} \quad (2)$$

Finally, *accuracy* is defined as the ratio of the total number of correct predictions to the total number of predictions. In the case of ECG anomaly detection, *accuracy* can be formulated as

$$Accuracy = \frac{Number\ of\ samples\ correctly\ predicted\ (abnormal\ or\ normal)}{Total\ number\ of\ predictions} \quad (3)$$

The autoencoder applied to the ML model consisted of three fully connected layers with 140 neurons in the input layer, 32 neurons in the hidden layer, and 140 neurons in the output layer. These parameters were chosen experimentally to achieve an acceptable performance. The mean absolute error (MAE) function was used to compute the training losses and ADAM was used as the optimizer to update network weights during training.

For FL, we used the same autoencoder model as described above. The model had a total of 9132 parameters. The dataset was divided equally among five devices. To achieve comparable performance to machine learning, the FL model required three communication rounds with 100 epochs in each communication round. Hence, these parameters were used for all FL simulations.

When the PCA dimensionality reduction algorithm is applied in FL, it essentially reduces the 140 data points in the dataset to a smaller number of PCA components. This reduces the number of neurons in the input and output layers, resulting in the reduction in the number of model parameters. This in turn reduces the communication cost; however, it may reduce the model performance. Hence, we next focus on the selection of the appropriate number of PCA components, which are obtained from simulations. The volume of data required for FL with PCA and the corresponding model performance in terms of recall, precision, and accuracy with different communication rounds and different numbers of PCA components, as determined from simulations, are tabulated in Table 2. It is observed that the trained model performance measures (recall, precision, and accuracy) are consistently above 95% with 100 epochs and 20 PCA components with a sufficient reduction in the volume of data; hence, we select these parameters (indicated in boldface in Table 2.).

Table 2. Traffic volume in kB for federated learning with PCA considered for various numbers of epochs and PCA components with 3 communication rounds.

Epochs	PCA Components	Volume of Data	Recall	Precision	Accuracy
60	10	49.1	0.934	0.953	0.954
	20	95.9	0.963	0.918	0.950
	30	142.7	0.830	0.932	0.905
80	10	49.1	0.866	0.922	0.915
	20	95.9	0.871	0.942	0.925
	30	142.7	0.805	0.954	0.904
100	10	49.1	0.803	0.914	0.888
	20	**95.9**	**0.990**	**0.958**	**0.978**
	30	142.7	0.903	0.959	0.944

In summary, the model parameters used for FL with the PCA dimensionality reduction algorithms are listed in Table 3. Unless otherwise specified, these values are used for all performance evaluations of FL-LoRaMAC.

Table 3. Autoencoder model details.

Parameter	Value
Number of layers	3
Layers type	Dense
Neurons in input layer	20
Neurons in hidden layer and activation	15; ReLU
Neurons in output layer and activation	20; Sigmoid
Optimizer	Adam
Loss function	MAE

6. Results and Discussions

We now present the results obtained from the simulation experiments. These are presented in two parts. First, we present results indicating the comparative performance of the proposed FL model with respect to a centralized ML model in terms of traffic volume, trained model performance, and computation cost. These results demonstrate that the FL model largely reduces the communication cost and computations for achieving a comparable model performance to the centralized ML model. Next, we present an extensive study of the proposed FL-LoRaMAC framework, including the impact of packet loss on the model performance, model training time, and energy consumption, and the quantitative benefits of pruning and the proposed QoS-based SF distribution scheme.

6.1. Performance Comparison between Federated Learning and Machine Learning

This section presents the performance analysis of the AI model trained using federated learning in comparison to machine learning principles in the absence of channel loss. The effects of channel loss are reported in the next section.

6.1.1. Volume of Traffic

We first evaluate and compare the volume of traffic for machine learning, federated learning, and federated learning with PCA.

- *Machine Learning:* With ML, all the data points need to be sent to the central server. Each data sample consists of 140 data points (each of 8 bytes) and a label (each of

4 bytes) resulting in a data sample that is 140 × 8 + 4 = 1124 bytes long. There is a total of 5000 such data samples. Hence, for machine learning, 5000 × 1124 = 4,496,000 bytes or 4.5 MB of data needs to be communicated.

- *Federated Learning:* When using FL, each of five devices will send 9132 × 4 = 36,528 bytes in one communication round. This results in 182,640 bytes of data transmitted for five devices in one communication round. Hence, the total data sent by five devices in three communication rounds equals 547,920 bytes. The server will also send the updated global model after performing federated averaging in each communication round. The total data sent by the server to end devices equals 109,584 bytes. Hence, the total volume of data that needs to be communicated for federated learning is 657,504 bytes or 657.5 KB.
- *Federated learning with PCA:* When PCA is employed with federated learning, each data sample is reduced from 140 data points to 20 data points. Consequently, the number of neurons in the input and output layer also get reduced from 140 to only 20, with 32 neurons in the hidden layer. The resultant model comprises 1332 model parameters. Following the similar calculations performed earlier for federated learning without PCA, the total volume of data traffic communicated among five devices and the server in three communication rounds for federated learning along with PCA equals 95,904 bytes or 95.9 KB.

Figure 10 depicts the relative communication costs of the three methods for a similar level of model performance, which indicates that the volume of data traffic in federated learning is significantly lower in comparison to the typical machine learning approach. The traffic volume is further reduced when the PCA dimensionality reduction algorithm is used. Specifically, with FL using PCA as specified above, the volume of traffic is reduced by 98% relative to an ML model for the same level of model performance.

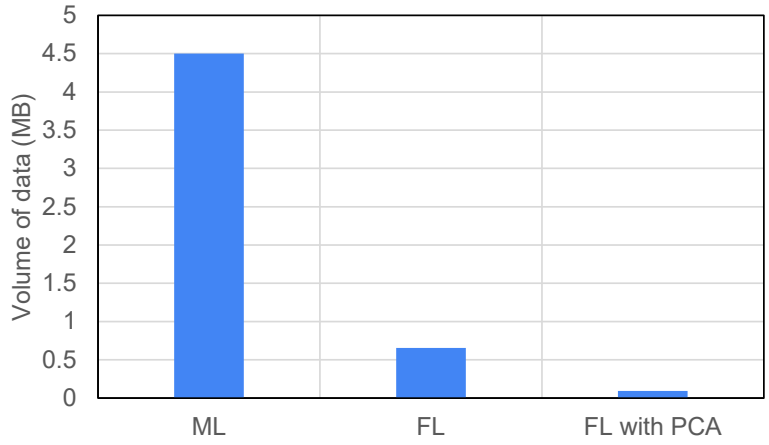


Figure 10. Traffic volume for machine learning (100 epochs), federated learning (3 communication rounds and 100 epochs), and federated learning with PCA (20 components, 3 communication rounds, and 100 epochs) approaches.

6.1.2. Trained Model Performance

The autoencoder was trained individually using typical machine learning as well as using federated learning principles. In the case of machine learning, the model was trained for 100 epochs. In federated learning, the model was trained for three communication rounds and each communication round consisted of 100 epochs. The model was also trained with federated learning with 20 PCA components. The trained model performance was evaluated by testing the model on the testing dataset. The performance of models trained with different methods is tabulated in Table 4, where the numbers of true positives (TPs), true negatives (TNs), false positives (FPs), and false negatives (FNs) are listed. For clarity, the implications of these terms in an ECG anomaly detection scenario are stated below:

TP: The number of samples that are actually normal and are predicted as normal;
TN: The number of samples that are actually abnormal, and are predicted as abnormal;
FP: The number of samples that are actually abnormal but are wrongly predicted as normal;
FN: The number of samples that are actually normal but are wrongly predicted as abnormal.

Table 4. Performance of models trained with different methods.

Method	TN	FP	FN	TP
Machine learning	404	7	10	579
Federated learning	407	4	16	573
Federated learning with PCA	407	4	20	569

It is intuitive that, for ECG anomaly detection, it is desirable that the TN and TP are high and the FP is low. The FN is not of much importance in this scenario as a false alarm is better than a missed anomaly; however, it should also be kept low.

From Table 4, it can be observed that, for the representative application, the model trained with federated learning outperforms the model trained by machine learning. When the PCA algorithm is used along with federated learning, the performance of the model is very similar to that of the model trained without the PCA algorithm, with a slight increase in false alarms.

6.1.3. Computation Time

The computation time for the different approaches was assessed for model training times only. They do not include the data pre-processing time or communication time. All experiments were performed on a CPU-based machine with 8 GB of RAM.

- *Machine Learning:* For machine learning, the training took 100 epochs to converge. The entire training phase took 62 s to be completed.
- *Federated Learning:* For federated learning, all devices performed the training of model parameters in parallel. During the training phase, one communication round took 5 s on average. The federated averaging algorithm running on the server took 0.016 s. Hence, one communication round took a total of 5.016 s for computation. Hence, three communication rounds took 15.048 s for computation.

The results, plotted in Figure 11, indicate that federated learning takes less computational time in comparison to machine learning as it trains models in a decentralized way. Federated learning also provides the benefit of requiring a substantially lower amount of data to be communicated while practically providing the same quality of performance.

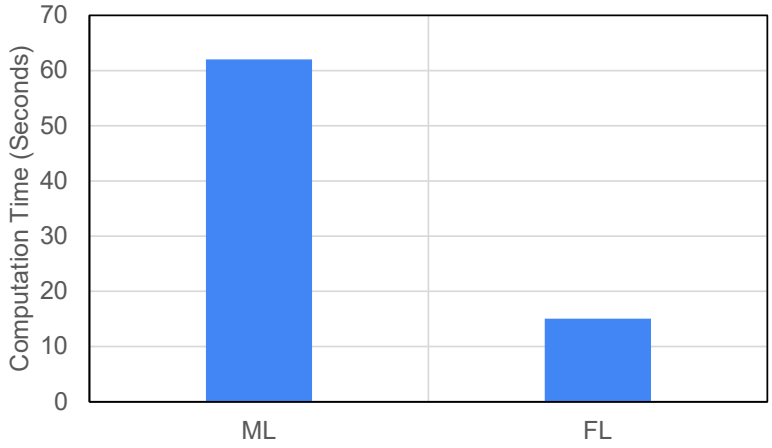


Figure 11. Computation time for machine learning (100 epochs) and federated learning (3 communication rounds and 100 epochs) approach.

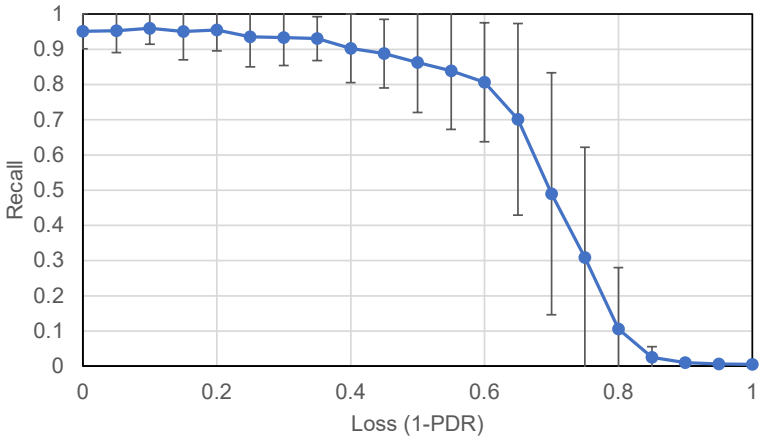
6.2. Performance Study of FL-LoRaMAC

In the following, we present results from simulations on the performance of FL-LoRaMAC to capture its general performance characteristics.

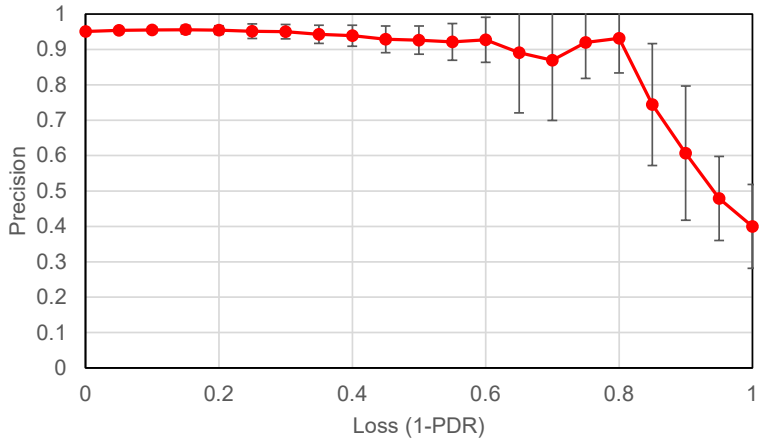
6.2.1. Model Performance of FL-LoRaMAC in the Presence of Channel Loss

We now present the effect of packet loss on the performance of the proposed FL-LoRaMAC framework. We assume that the ambient traffic in the entire network leads to packet losses on uplink transmissions, which depend on the traffic volume. It is assumed that packet loss does not affect network joining and all end devices are successfully configured according to the specifications mentioned in Section 4.2. Also, the end nodes are supplied with the training data (representing locally sensed data in a real scenario). The application server transmits the untrained global model as downlink messages to the end devices taking part in the training. This transmission of the global model is acknowledged by the end devices. Hence, all fragments of the global model are received by all the end devices without loss.

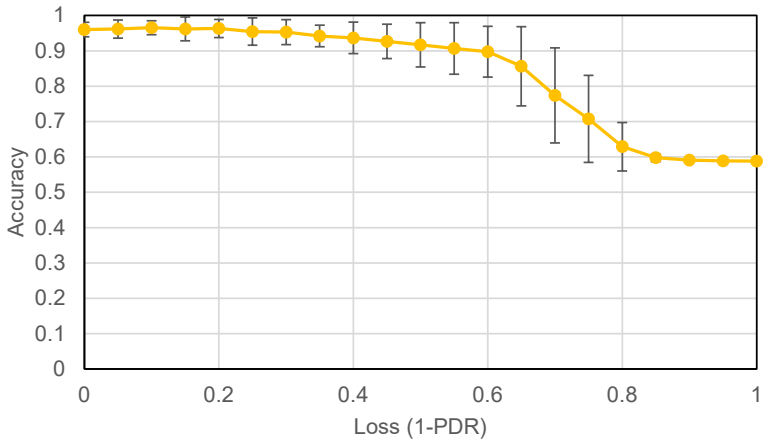
The end nodes train the model based on the local data and the corresponding gradient updates are transmitted to the application server as fragments. Each end device is assumed to transmit one fragment every 60 s. We assume that these fragments are 28 bytes long. Assuming that all devices use SF-7, this leads to the time on air for each packet being 62 millis. During this transmission, fragments may suffer collisions with either data fragments from other participating devices or data from non-participating devices. We use a Poisson analysis for evaluating the probability of successful transmissions under these conditions. The system treats lost fragments from collisions according to the specifications in Section 4.3. Due to this loss, the model performance deteriorates in terms of making predictions on the testing dataset. Evaluation metrics calculated for varying loss rates from the implementation, each averaged from 100 simulation experiments, are depicted in Figure 12.



(**a**) Recall



(**b**) Precision



(**c**) Accuracy

Figure 12. Average performance and corresponding standard deviations with packet loss: (**a**) recall, (**b**) precision, and (**c**) accuracy with the varying loss in network.

In the absence of packet loss, the performance of the model was highest with a mean recall of 95.16%, a mean precision of 95.07%, and a mean accuracy of 95.99%. As expected, the performance of the model degrades with an increasing packet loss. However, this degradation is very slow until the loss rate exceeds 40%, below which the system was able to maintain a mean recall greater than 90.26%, mean precision above 93.9%, and mean accuracy above 93.64%. The federated averaging mechanism was responsible for keeping

the performance stable and hence the performance degrades gracefully. Even if one of the transmissions of a certain model parameter is received by the server, the averaging algorithm preserves the notion of that parameter in the updated global model. Hence, the end devices receive the averaged value instead of obtaining a 'zero' for such cases. This helps the model to converge faster and reap the benefits of federated learning. Once packet loss exceeds a threshold, i.e., not even a single transmission for some parameters is able to reach the server, the model may need more communication rounds to converge or may not converge at all.

6.2.2. Model Training Time with FL-LoRaMAC

The time duration for computing gradient updates during the training phase was measured experimentally and the communication time was calculated mathematically for the training hyperparameters used. With these, the total training time was calculated as follows:

$$T_{Train} = T_{DU} + N * (T_C + T_{Up} + T_{FAv} + T_D) \quad (4)$$

In (4), T_{DU} is the average time taken by the server to transmit all the untrained global model parameters, N is the number of communication rounds used, T_C is the average time taken to compute local model updates, T_{Up} is the average time taken by end devices to transmit local updated model parameters, T_{FAv} is the average time taken by the federated averaging algorithm to compute the updated global model from local updates, and T_D is the average time taken to transmit the updated global model parameters.

All calculations for communication time were made considering a spreading factor of SF-7. The same can be calculated if other spreading factors were to be used. To calculate communication time, the total number of model parameters that need to be communicated was determined. The model has 635 parameters, which equates to 2540 bytes. Each fragment was considered to be 28 bytes long. Hence, there were 91 fragments that need to be communicated from each end device in each communication round. Considering that each end device transmits one packet every 60 s, T_{Up} was calculated to be 5460 s.

The size of the frame was chosen to be 28 bytes based on the fact that LoRa technology in the U.S. has a dwell time restriction of 400 millis. When SF-10 is used, the duration of a 28 bytes packet is 400 millis. Hence, the size of the fragment was chosen to be 28 bytes for the simplicity of implementation with other SFs. Note that when lower spreading factors are used, more data can be sent in one packet. However, a longer packet means a higher probability of collision when there are multiple devices transmitting, which is the case during end devices sending local updates.

When the global model is sent by the server, only one gateway will be transmitting. Hence, the maximum number of bytes that can be packed in a packet can be sent more frequently, with a negligible probability of packet loss due to collisions. Assuming that the gateway transmits using SF7, each packet can have 200 bytes with 100 ms of the preamble. Hence, the gateway can send all 635 parameters with 13 packets, transmitting 1 packet every 10 s. Considering that no packets are lost during transmission, T_{DU} as well as T_D will be 130 s.

The values of T_C and T_{FAv} were measured to be 4.95 s and 0.016 s, respectively. By putting all the values in Equation (4), the total training time in s was calculated to be 16,914.9 s, or 4.7 h:

$$T_{Train} = 130 + (3 * (5460 + 130 + 4.95 + 0.016))\ s \quad (5)$$

Note that the communication of model parameters takes the most time during the training phase. The training time can be reduced if the number of communication rounds is reduced; however, doing so will deteriorate the performance of the model as shown in Table 5. This trade-off can be exploited depending on the nature of the application.

Table 5. Training time and model performance for different numbers of communication rounds.

Comm. Rounds	Training Time (s)	Recall (%)	Precision (%)	Accuracy (%)
2	11,229.9	86.13	93.9	92
3	16,794.85	99.27	95.77	97.9

6.2.3. Energy Consumption of FL-LoRaMAC

As discussed earlier, FL can be implemented using legacy LoRaWAN protocol using the Class-C mode of operation for end devices. However, doing so will incur enormous energy costs. This section compares the energy consumption of the proposed FL-LoRaMAC framework to the Class-C operation of the legacy LoRaWAN protocol.

The uplink transmissions for FL-LoRaMAC are similar to those of the legacy protocol. Hence, the energy consumption for uplink transmissions will be the same for both cases. However, for downlink, the end devices according to FL-LoRaMAC will conserve energy by putting its radios to sleep.

The energy consumed by the LoRa transceiver at the end device for receiving one downlink packet in FL-LoRaMAC can be calculated as follows:

$$E_T = E_S + E_{Un} + E_{Sl} \tag{6}$$

Here, E_S is the energy consumed when the device starts listening and actually receives the downlink packet, E_{Un} is the energy consumed when the device starts listening and does not receive any data, and E_{Sl} is the energy consumed by the transceiver module in sleep mode.

The energy consumption depends on the operating voltage of the transceiver, the current drawn, and the duration for which the current is drawn. The SX1276 LoRa module operates at 3.3V, draws 10.8 mA (I_R) during receive mode, and 0.0002 mA (I_{Sl}) in sleep mode [30]. The gateway transmits downlink frames once every 10 s. In FL-LoRaMAC, the end devices open receive windows every 100 millis, each for 5 millis, and then go to sleep if they do not detect any LoRa preamble. E_S, E_{Un}, and E_{Sl} can be found according to Equations (7)–(9).

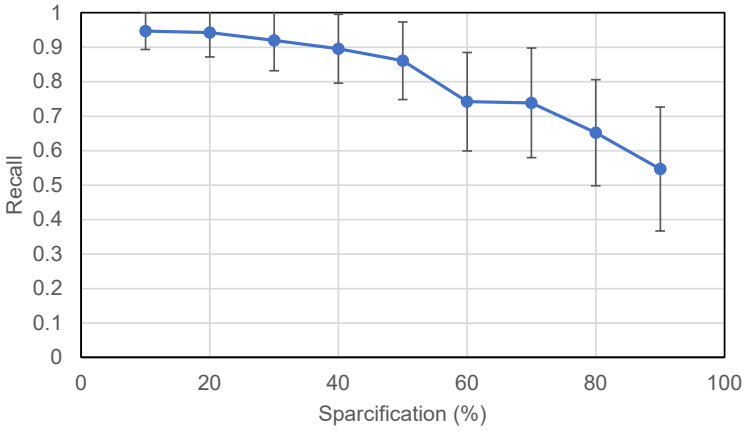
$$\begin{aligned} E_S &= V * I_R * T_R = 3.3V * 10.8mA * 400ms & (7)\\ E_{Un} &= V * I_R * T_R = 3.3V * 10.8mA * 480ms & (8)\\ E_{Sl} &= V * I_{Sl} * T_{Sl} & (9)\\ &= 3.3V * 0.0002mA * (1000 - 480 - 400)ms \end{aligned}$$

Here, T_R and T_{Sl} are time durations of reception and sleep, respectively. Using Equation (6), the energy consumption of the LoRa transceiver for one downlink packet is 0.0087 mWH. Using SF-7, 13 packets need to be sent in the downlink for transmitting all model parameters in one communication round. Hence, the energy consumed by the transceiver to receive all the parameters in one communication round is 0.1131 mAH.

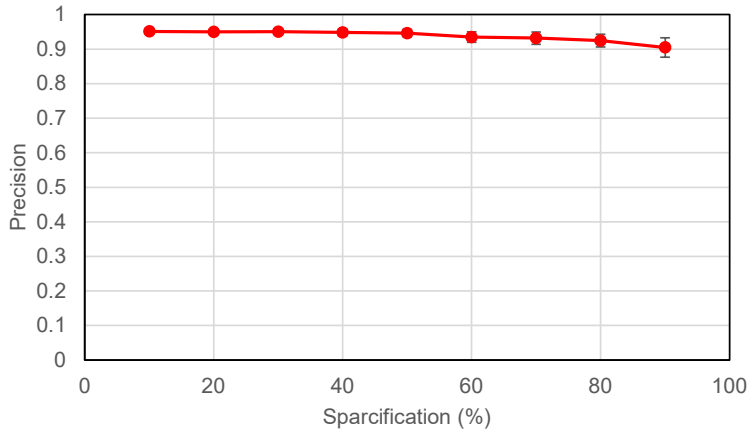
In the case of legacy Class-C operation, the transceiver module always stays in receive mode. It will use the standard eight-symbol preamble, not the elongated preamble, so a data packet of SF7 can have 255 bytes. In this scenario, 10 packets will be sufficient for transmitting all 635 model parameters. The energy consumed by the transceiver to receive all model parameters will be 10 × (3.3V × 10.8 mA × 10,000 s) = 0.99 mAH. These calculations indicate that the legacy Class-C devices will consume approximately nine times more energy than the proposed FL-LoRaMAC for receiving model parameters.

6.2.4. Performance with Pruning

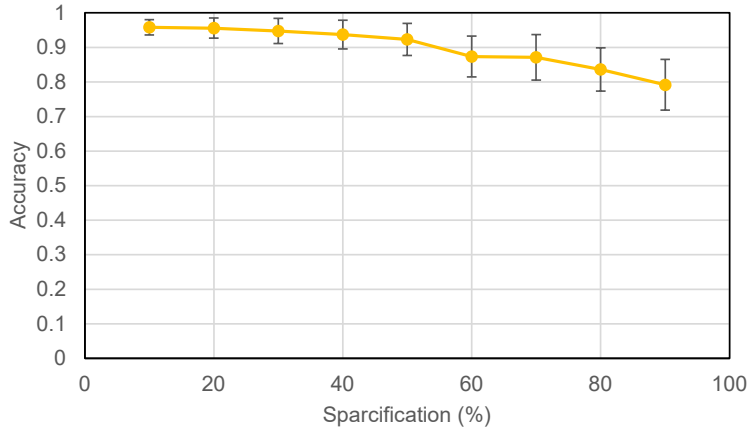
While performing model pruning, a sparsification parameter specifies the percentage of the model parameters that are pruned. The performance of the model was evaluated for varying sparsification percentages and plotted in Figure 13. These results were obtained by training the model 100 times and the plots show the average and standard deviation for the performance metrics.



(**a**) Recall



(**b**) Precision



(**c**) Accuracy

Figure 13. Performance with various levels of sparsification: (**a**) recall, (**b**) precision, and (**c**) accuracy with the varying sparsification percentages.

It is observed that the model performance degrades as the parameter matrix gets more and more sparse. Sparsity is a hyper-parameter and it can be observed that, for the representative application, the model performance is not substantially affected for up to 50% sparsity. Hence, if 50% of the model parameters are pruned, the model still performs well. If the framework chooses not to send those pruned parameters, this will result in at least 50% savings in bandwidth with minimal compromise to the performance of the system.

6.2.5. Performance with QoS-based SF Distribution

Finally, we study the performance of the proposed QoS-based SF distribution principles. As stated earlier, we assume that the packet transmissions in FL-LoRaMAC are affected by ambient transmissions of packets in the IoT network such as from devices in the network that are not performing ECG anomaly detection. Some of these ambient packets may also be interfering with FL-LoRaMAC's high-priority packets. We assume a requirement of a 90% probability of success for high-priority packets. Also, it is assumed that all end-devices, including those transmitting ambient packets, are transmitting data every 60 s, similar to the devices running the representative application. However, we assume that at any time the network has only 10% high-priority packets out of the total traffic.

Using this network configuration, the maximum number of devices that can be supported on each spreading factor was computed using Poisson analysis, as tabulated in Table 6. The spreading factors were then allocated to different packets without exceeding the maximum allowable limit, according to the SFA-1 approach. The packet delivery ratios for both high-priority and low-priority devices were then evaluated using Poisson's distribution for a varying number of total devices in the network. These results are plotted in Figure 14.

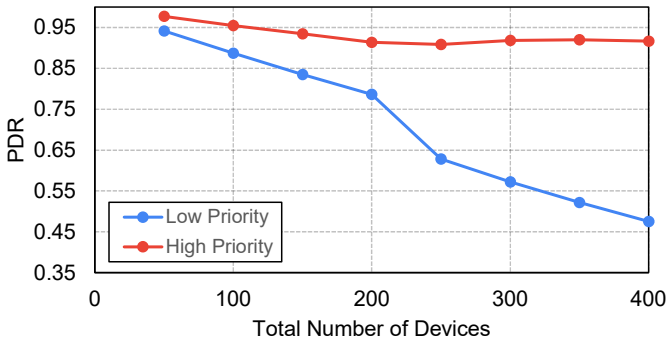


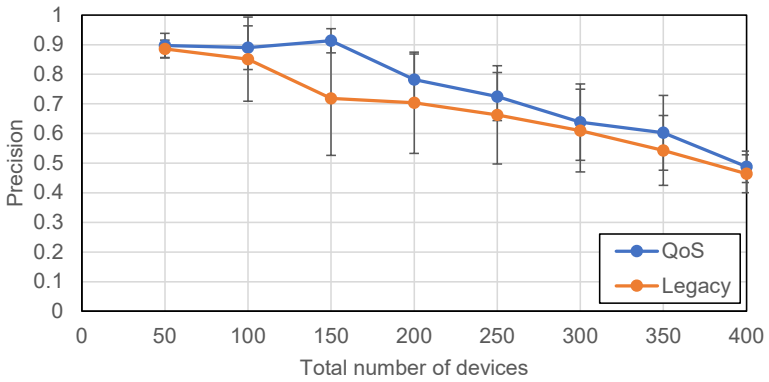
Figure 14. PDR for high- and low-priority devices for varying total number of devices.

Table 6. Maximum number of devices for various spreading factors with 90% or greater probability of success.

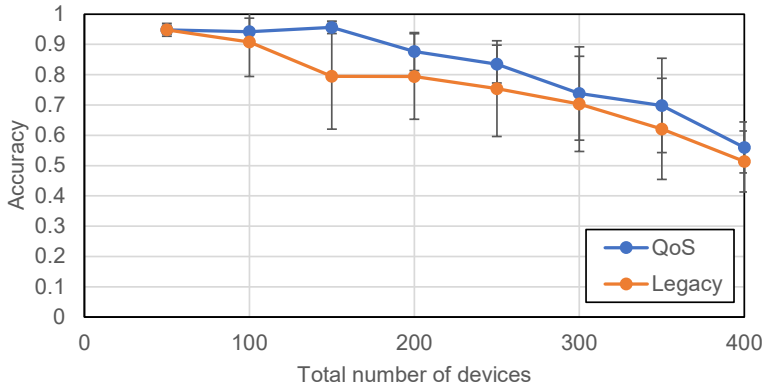
Spreading Factors	Max. Number of Devices (D)
SF7	50
SF8	27
SF9	15
SF10	8

To study the effects of the packet losses in the channel, the ECG anomaly detection application was trained and tested using the PDR values for the varying total number of devices illustrated in Figure 14. The significance criteria were chosen to be 80% of the maximum and minimum values. In other words, if the parameter's value is greater than 80% of the maximum value or less than 80% of the minimum value, that particular model parameter is considered significant. The performance of the model trained by

applying the pruning technique along with QoS-based distribution and without applying QoS distribution is plotted in Figure 15.



(**a**) Precision



(**b**) Accuracy

Figure 15. Performance comparison of the trained model with QoS and legacy: (**a**) precision and (**b**) accuracy with the varying total number of devices in network.

For comparison, the value of recall was kept fixed at 99.02%, and the mean as well as the standard deviation for precision and accuracy were determined against this recall value. This was achieved by varying the threshold for the prediction. It can be observed that, with an increasing number of devices in the network, the performance of the model degrades. However, the model trained by considering a differential priority of packets (i.e., the proposed QoS-based SF distribution scheme) provides a better performance than the legacy scheme that applies the same priority for all packets. Providing better success rates to high-priority packets that contain more significant parameters helps the model to converge faster when compared to the legacy approach.

7. Conclusions

This paper introduces artificial intelligence tools for IoT applications and a novel framework to enable on-device learning for LoRa-based devices. Machine learning principles can prove to be very beneficial for IoT applications but, due to the challenges imposed by IoT and LoRaWAN ecosystems such as data privacy, low bandwidth, etc., traditional machine learning techniques cannot be efficiently applied. However, a decentralized learning technique known as federated learning can be used while maintaining data privacy. The performance of models trained via traditional machine learning and federated learning principles was analyzed. This analysis proves that, for the representative application, federated learning can be viably employed in place of machine learning. However, federated learning requires independent bidirectional communication. The legacy LoRaWAN protocol fails to satisfy the communication requirements of federated learning; hence, FL-LoRaMAC

has been proposed to fill this gap. The performance of the framework was evaluated by implementing the framework on a representative application. The results show that the framework was able to successfully support all the design requirements of federated learning, even with communication losses in the system. The proposed framework also incorporates bandwidth optimization to enable efficient resource utilization.

The proposed framework considers all devices running the application taking part in the learning process. Typically, a subset of these devices possesses a substantial amount of information and holds the capacity to effectively train the learning model. To mitigate potential escalations in computational and communication expenses, a nuanced approach is necessary, wherein the training process is applied to to only a small subset of devices without regard for their information contribution. This will further reduce the communication cost.

This also leads to an intriguing avenue for further investigation. The development of a selection mechanism can lead to identifying devices based on the richness of their information and the quality of their channel conditions. This aspect remains underexplored at present but bears substantial significance, particularly within the context of federated-learning-driven applications for smart cities. Hence, delving into this aspect could yield valuable insights and enhancements for federated learning in such contexts.

Author Contributions: Conceptualization, S.A. and A.N.; methodology, S.A. and A.N.; software, S.A.; validation, S.A. and A.N.; formal analysis, S.A.; writing—original draft preparation, S.A.; writing—review and editing, A.N.; supervision, A.N.; project administration, A.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Future of Industry Ecosystems: Shared Data and Insights. Available online: https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/#:~:text=IDC%20estimates%20there%20will%20be,the%20importance%20of%20expanding%20their (accessed on 8 September 2023).
2. Majumdar, N.; Shukla, S.; Bhatnagar, A. Survey On Applications Of Internet Of Things Using Machine Learning. In Proceedings of the 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India 10–11 January 2019; pp. 562–566. https://doi.org/10.1109/CONFLUENCE.2019.8776951.
3. Shinde, P.P.; Shah, S. A Review of Machine Learning and Deep Learning Applications. In Proceedings of the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 16–18 August 2018; pp. 1–6. https://doi.org/10.1109/ICCUBEA.2018.8697857.
4. Chen, M.; Hao, Y.; Hwang, K.; Wang, L.; Wang, L. Disease Prediction by Machine Learning Over Big Data From Healthcare Communities. *IEEE Access* **2017**, *5*, 8869–8879. https://doi.org/10.1109/ACCESS.2017.2694446.
5. Shailaja, K.; Seetharamulu, B.; Jabbar, M.A. Machine Learning in Healthcare: A Review. In Proceedings of the 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 29–31 March 2018; pp. 910–914. https://doi.org/10.1109/ICECA.2018.8474918.
6. Microsoft: AI for Health. Available online: https://www.microsoft.com/en-us/ai/ai-for-health (accessed on 8 September 2023).
7. Tempus: Data-Driven Precision Medicine. Available online: https://www.tempus.com/ (accessed on 8 September 2023).
8. Beta Bionics. Available online: https://www.betabionics.com/ (accessed on 8 September 2023).
9. Insitro. Available online: https://insitro.com/ (accessed on 8 September 2023).
10. LoRa and LoRaWAN: Technical Overview. Available online: https://lora-developers.semtech.com/library/tech-papers-and-guides/lora-and-lorawan/ (accessed on 8 September 2023).
11. Vangelista, L.; Zanella, A.; Zorzi, M. Long-Range IoT Technologies: The Dawn of LoRa™ . In *Future Access Enablers for Ubiquitous and Intelligent Infrastructures*; Atanasovski, V., Leon-Garcia, A., Eds.; Springer: Cham, Switzerland, 2015; pp. 51–58.
12. Zourmand, A.; Kun Hing, A.L.; Wai Hung, C.; AbdulRehman, M. Internet of Things (IoT) using LoRa technology. In Proceedings of the 2019 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS), Selangor, Malaysia, 29 June 2019; pp. 324–330. https://doi.org/10.1109/I2CACIS.2019.8825008.
13. Aggarwal, S.; Nasipuri, A. QoS Based Spreading Factor Assignment for LoRaWAN Networks for IoT Applications. In Proceedings of the SoutheastCon 2022, Mobile, AL, USA, 26 March–3 April 2022; pp. 46–53.

14. Can a Smartwatch with ECG Capability Really Warn You About an Irregular Heartbeat? Available online: https://www.houstonmethodist.org/blog/articles/2022/jan/can-a-smartwatch-with-ecg-capability-really-warn-you-about-an-irregular-heartbeat/#:~:text=The%20ECG%20technology%20in%20a,sense%20of%20your%20heart's%20rhythm (accessed on 8 September 2023).
15. Kubat, M.; Holte, R.C.; Matwin, S. Machine Learning for the Detection of Oil Spills in Satellite Radar Images. *Mach. Learn.* **1998**, *30*, 195–215. https://doi.org/10.1023/A:1007452223027.
16. Otukei, J.; Blaschke, T. Land cover change assessment using decision trees, support vector machines and maximum likelihood classification algorithms. *Int. J. Appl. Earth Obs. Geoinf.* **2010**, *12*, S27–S31. https://doi.org/10.1016/j.jag.2009.11.002.
17. Ahmad, S.; Kalra, A.; Stephen, H. Estimating soil moisture using remote sensing data: A machine learning approach. *Adv. Water Resour.* **2010**, *33*, 69–80. https://doi.org/10.1016/j.advwatres.2009.10.008.
18. Simjanoska M, Gjoreski M, Gams M, Madevska Bogdanova A. Non-Invasive Blood Pressure Estimation from ECG Using Machine Learning Techniques. *Sensors* **2018**, *18*, 1160. https://doi.org/10.3390/s18041160 .
19. Raju, K.B.; Dara, S.; Vidyarthi, A.; Gupta, V.M.; Khan, B. Smart Heart Disease Prediction System with IoT and Fog Computing Sectors Enabled by Cascaded Deep Learning Model. *Comput. Intell. Neurosci.* **2022**, *2022*, 1070697. https://doi.org/10.1155/2022/1070697 .
20. Rincon, J.A.; Guerra-Ojeda, S.; Carrascosa, C.; Julian, V. An IoT and Fog Computing-Based Monitoring System for Cardiovascular Patients with Automatic ECG Classification Using Deep Neural Networks. *Sensors* **2020**, *20*, 7353. https://doi.org/10.3390/s20247353.
21. Clifford GD, Liu C, Moody B, Li-wei HL, Silva I, Li Q, Johnson AE, Mark RG. AF classification from a short single lead ECG recording: The PhysioNet/computing in cardiology challenge 2017. In Proceedings of the 2017 Computing in Cardiology (CinC), Rennes, France, 24 September 2017; pp. 1–4. https://doi.org/10.22489/CinC.2017.065-469.
22. *LoRaWAN Specification v1.0*; LoRa Alliance, Inc.: San Ramon, CA, USA, 2015.
23. Mikhaylov, K.; Petaejaejaervi, J.; Haenninen, T. Analysis of Capacity and Scalability of the LoRa Low Power Wide Area Network Technology. In Proceedings of the European Wireless 2016, 22th European Wireless Conference, Oulu, Finland, 18–20 May 2016; pp. 1–6.
24. Vejlgaard, B.; Lauridsen, M.; Nguyen, H.; Kovacs, I.Z.; Mogensen, P.; Sorensen, M. Coverage and Capacity Analysis of Sigfox, LoRa, GPRS, and NB-IoT. In Proceedings of the 2017 IEEE 85th Vehicular Technology Conference (VTC Spring), Sydney, Australia, 4–7 June 2017; pp. 1–5. https://doi.org/10.1109/VTCSpring.2017.8108666.
25. Aggarwal, S.; Nasipuri, A. Survey and Performance Study of Emerging LPWAN Technologies for IoT Applications. In Proceedings of the 2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT IoT and AI (HONET-ICT), Charlotte, NC, USA, 6–9 October 2019; pp. 69–73. https://doi.org/10.1109/HONET.2019.8908117.
26. Kurita, T. Principal component analysis (PCA). In *Computer Vision: A Reference Guide*; Springer: Cham, Switzerland, 2019; pp. 1–4.
27. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning Both Weights and Connections for Efficient Neural Networks. In Proceedings of the Proceedings of the 28th International Conference on Neural Information Processing Systems–Volume 1, Cambridge, MA, USA, 7–12 December 2015; NIPS'15, p. 1135–1143.
28. Aggarwal, S.; Nasipuri, A. Improving Scalability of LoRaWAN Networks by Spreading Factor Distribution. In Proceedings of the SoutheastCon 2021, Atlanta, Georgia, USA, 10–13 March 2021; pp. 1–7. https://doi.org/10.1109/SoutheastCon45413.2021.9401855.
29. ECG-5000 Dataset. Available online: http://www.timeseriesclassification.com/description.php?Dataset=ECG5000 (accessed on 11 August 2022).
30. SX1276 Datasheet. Available online: https://www.mouser.com/datasheet/2/761/sx1276-1278113.pdf (accessed on 8 September 2023).