Big Data Research 40 (2025) 100510

Contents lists available at ScienceDirect

Big Data Research

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

Efficient training: Federated learning cost analysis

Rafael Teixeira [a,b,*], Leonardo Almeida [a], Mário Antunes [a,b], Diogo Gomes [a,b], Rui L. Aguiar [a,b]

[a] Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, 3810–193, Aveiro, Portugal
[b] Departamento de Eletrónica Telecomunicações e Informática, Universidade de Aveiro, Aveiro, 3810–193, Aveiro, Portugal

ARTICLE INFO

Keywords:
Federated learning
Network slicing
Intrusion detection system
5G
6G

ABSTRACT

With the rapid development of 6G, Artificial Intelligence (AI) is expected to play a pivotal role in network management, resource optimization, and intrusion detection. However, deploying AI models in 6G networks faces several challenges, such as the lack of dedicated hardware for AI tasks and the need to protect user privacy. To address these challenges, Federated Learning (FL) emerges as a promising solution for distributed AI training without the need to move data from users' devices. This paper investigates the performance and costs of different FL approaches regarding training time, communication overhead, and energy consumption. The results show that FL can significantly accelerate the training process while reducing the data transferred across the network. However, the effectiveness of FL depends on the specific FL approach and the network conditions.

1. Introduction

Although the Sixth-generation Network (6G) is still in early development, various works already discuss which technologies should be used, what Key Performance Indicators (KPIs) need to be fulfilled, and which use cases should be addressed [1–5]. One central technology addressed in this discussion is Artificial Intelligence (AI) and how it should be leveraged.

In 6G, AI is expected to replace traditional methods in most network functions, solving tasks such as designing and optimizing the wireless architecture [4,5], network management and orchestration [3,5], intrusion detection [3], and automated network slicing [2,3,5]. This is a consequence of the increased complexity and the stricter latency requirements, which are already felt in 5G, that make traditional management models too slow or incapable of dealing with the complexity [6]. This reliance on AI makes it a pivotal technology in transitioning from the currently connected people and things to connected intelligence.

However, one prominent problem must be addressed to implement this vision successfully: where and how is the model training/inference performed? Considering the lack of specialized hardware for AI tasks in currently deployed edge nodes, the solution to this problem can be twofold. Either equip edge nodes with dedicated hardware for AI or leverage the number of available edge nodes and perform distributed training/inference. Since the first approach is inherently expensive, especially when considering that some recent models require multiple dedicated computing units to train in a reasonable time (1 day [7]), the second one is the solution supported in most 6G whitepapers [2,4,5].

Although distributed learning was developed to reduce training times, where model training leverages various computing units to spread the training, the scenario in which it will be applied in 6G is considerably different. Currently, the efficacy of distributed learning is achieved in High-Performance Computing (HPC) clusters, where the various dedicated computing units are connected via high-speed cable links. In 6G and advanced infrastructures for 5G-advanced, like the 5GAINER [8] and IMAGINE-B5G infrstructures,[1] the objective is to use the network nodes to perform the training, which do not have dedicated hardware and are connected between them with slower links than the ones in an HPC. This, in turn, can invalidate some of the proposed advantages of edge training, like reduced latency and reduced training times. Furthermore, edge training can also become an issue when considering sensitive data, given the privacy-preserving laws, like the General Data Protection Regulation (GDPR), as users need to consent to their data being transferred to the edge node for training.

The data sharing/ownership problem is tackled by Federated Learning (FL) [9]. This distributed training approach allows models to be

* Corresponding author.
E-mail addresses: rafaelgteixeira@av.it.pt (R. Teixeira), leonardoalmeida@av.it.pt (L. Almeida), mario.antunes@ua.pt (M. Antunes), dgomes@ua.pt (D. Gomes), ruilaa@ua.pt (R.L. Aguiar).

[1] https://imagineb5g.eu/.

https://doi.org/10.1016/j.bdr.2025.100510
Received 7 December 2023; Received in revised form 21 November 2024; Accepted 11 February 2025
Available online 20 February 2025

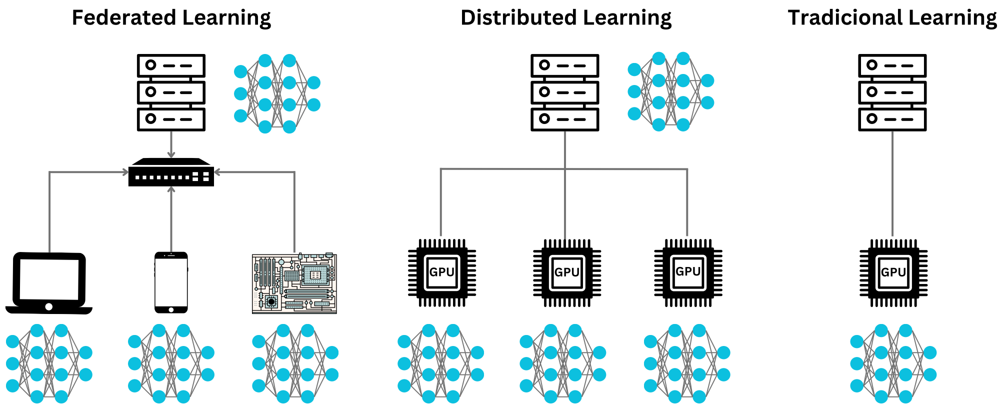


Fig. 1. Summary of the different training scenarios and the expected hardware for model training.

trained with the data from the various devices without the data leaving the devices in which it was collected. Since the data does not leave the device, the concerns about storing it securely and anonymizing it are no longer present, and privacy is kept. To train the models without taking the data from the devices, FL uses a distributed learning algorithm where each device trains a local model for one or more epochs and, after, shares the model weights with a centralized unit that combines them and sends the updated weights back to the devices. This cycle repeats until the global model achieves the desired performance.

Although this solution solves the data privacy problem, it has considerable costs. Unlike distributed learning, FL uses the collection device for model training, meaning it can use mobile phones, Internet of Things (IoT) devices, etc. Since the devices are not used exclusively to train the model, the percentage of device computation used for training is concerning, especially if the device is battery-powered, as no user wants their device's battery drained to train a model. Furthermore, since there is no control over the devices used for training, disconnections mid-training can negatively impact the final model's performance.

Considering the divergent requirements that will be found in telecommunications, understanding which model training approach should be used can be a challenging task, and further work needs to be developed to highlight each training approach's benefits and costs in different scenarios.

The main contributions of this work are the following:

- Insights into the costs different FL approaches have on data from common network tasks;
- The cost analysis from different FL approaches;
- Time characterization FL training approach;
- Network communication analysis for the different FL approaches.

The remainder of this document is organized as follows. In Section 2, a brief explanation of the different training approaches and their known costs is provided. Section 3, highlights recent work developed in FL for Fifth-generation Network (5G) and Beyond Fifth-generation Network (B5G). The experiments details, such as datasets used or models considered, are provided in Section 5. Section 6 presents the results obtained, and Section 7 their discussion. In Section 8, the conclusions and future work are drawn.

2. Background

In this section, we will overview model training from the perspective of how distributed resources can be leveraged to train the models and how the training cost can be reduced. Consequently, the overview will not cover different machine learning models or algorithms that optimize model parameters during training.

That said, even though most Machine Learning (ML) models can be trained with commodity hardware nowadays, bigger Deep Neural Networks (DNNs) usually require multiple dedicated computing units to be trained promptly. Since traditional learning cannot leverage numerous computing units to train a model, Distributed Learning was proposed. Recently, model training has experienced a new challenge related to ownership of data and its collection. Federated learning was presented as a training procedure that respects data privacy and avoids data centralization by training the model on the device where the data is collected and sending the resulting weights/gradients instead of the actual data to the central server.

Given that each training approach has advantages and disadvantages, this section details each process, highlighting their best scenarios. Fig. 1 summarises how each primary training approach leverages its hardware to train a model.

2.1. *Traditional learning*

Traditional learning is the simplest form of training. It trains a model using a single computing unit and the complete training set. Since it only uses a single computing unit, it is limited to vertical scaling, which can become very expensive. Currently, it cannot train the larger DNNs promptly, taking in some cases months to finish the training.

Despite that, this approach does not create any communication overhead, as there is no communication between computing units, and it is the one with less initialization overhead since there are no devices to sync with.

Ideally, if the model can be trained using this paradigm in a timely manner, then this is the best approach to training the model.

2.2. *Distributed learning*

Distributed learning builds upon traditional learning to enable it to be horizontally scaled. This is achieved by leveraging multiple computing units in the same computer or distributed over various computers connected through high-speed links [7].

Although most models can leverage multiple computing units to reduce training time, for example, decision trees [10], random forests [11] or DNNs [7]. Since shallow models usually have shorter training times even when using commodity hardware, model training distribution is only applied to DNNs. Consequently, we will focus on the DNNs' distributed and federated learning applications.

Regardless of the approach, it is assumed that a central server, called a parameter server, commands the training and stores the entire dataset. Once the training starts, the server provides the dataset subsets needed for each computing unit. The main difference between distributed training algorithms is how and what they communicate, as reducing the amount of communication between the workers and the parameter server while extracting valuable learning directions is the best way to reduce training times.

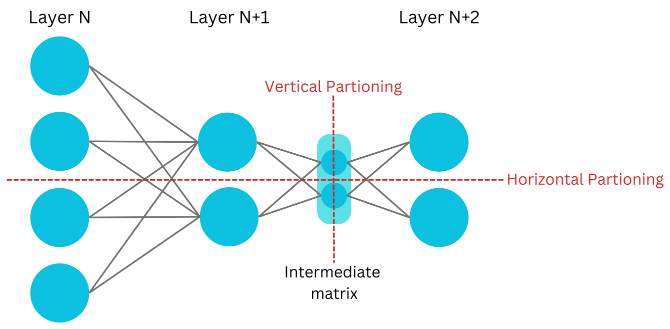


Fig. 2. Horizontal and vertical partitioning example.

With this in mind, several approaches to distribute the training were proposed that can be grouped under two main categories: model parallelism and data parallelism.

2.2.1. *Model parallelism*

Model parallelism can be achieved by dividing the model vertically (dividing by layers) or horizontally (dividing the layers), as demonstrated in Fig. 2. This approach is ideal when the computing unit cannot perform the entire model training alone. However, it is communication intensive as computing units must propagate/receive the intermediate values for each example twice (one for the forward propagation and one for the backward propagation).

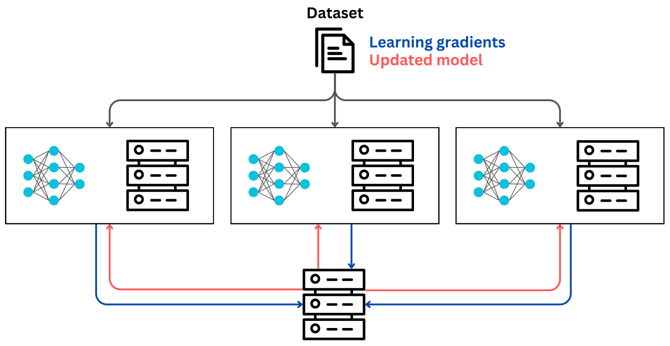
The most significant difficulty in implementing a model parallelism approach is finding the ideal model split, as the slowest route through the model determines the training/inference time. This, in practice, requires sophisticated algorithms, as even changing the mini-batch size can create a new optimal configuration. Given this difficulty and that most models fit in commodity hardware, there has been a shift towards data parallelism approaches in recent years.

2.2.2. *Data parallelism*

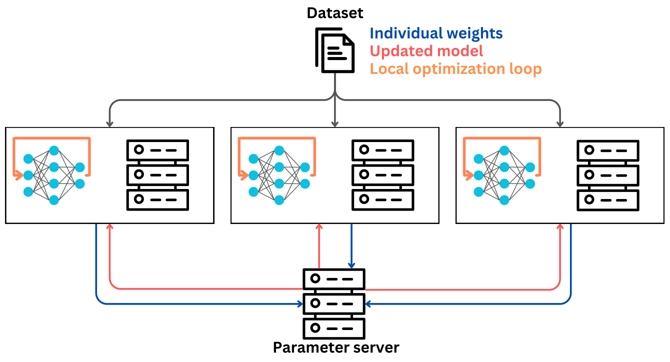
As the name states, data parallelism divides the training data over multiple computing units. The idea is that by having various computing units processing different samples simultaneously, the sample processing throughput increases, reducing the time spent per batch. The two main approaches to data parallelism are centralized and decentralized optimization, which differ on where the model parameters are updated as presented in Fig. 3.

In centralized optimization, only one node updates the model parameters, usually the parameter server. To update the model, it receives the gradients from every computing unit in the cluster, combines them, and performs backpropagation. After the weights are updated, the parameter server distributes the model through the remaining computing units in the cluster. Decentralized optimization, on the other hand, allows every node to update its model parameters. Instead of requiring computing units to share gradients, it requires them to share the current model weights. The remaining process is identical to a centralized optimization, as the parameter server receives the model weights, combines them, and distributes a new updated model with the computing units.

Regardless of the approach, the training can be performed synchronously or asynchronously. The methods described previously consider the approaches synchronous version—the asynchronous versions trade-off update consistency by worker efficiency. For example, in asynchronous decentralized learning, whenever a worker needs to synchronize with the parameter server, it does not need to wait for every worker to present their current model. Instead, it pulls the current version of the global model, combines it with its local version, and uploads the outcome to the parameter server. The problem with these approaches is that slower workers can end up with considerably outdated models. Whenever they combine their model with the parameter server, they can produce incorrect models. This is avoided by implementing safeguards to ensure the updated model goes in the correct direction. The downside of the safeguards is that they also reduce the impact of accurate updates.



(a) Example of a centralized learning approach.



(b) Example of a decentralized learning approach.

Fig. 3. Example of centralized and decentralized learning.

2.3. *Federated learning*

Every approach presented until now considered that there was a dataset that could be centralized if necessary. However, this is not always possible, especially when working with sensitive data like medical records, GPS location, and personally identifiable information. Since to use this information, the data owners need to provide consent, and the data must be anonymized, collecting a centralized dataset when possible is expensive.

Federated learning aims to solve this issue by using the data without centralizing it, removing any concerns regarding possible data leaks of the centralized dataset. Although in a completely different scenario, this approach can use any of the distributed learning approaches we mentioned before. The scenario has three main differences: first, the computing power since the computing units are usually cell phones or embedded devices that collect data with varied computing power. Second, the data distribution, since we cannot move data between devices and each of them might have a noticeable imbalance towards a specific pattern. Third is the device connection, since in federated learning, the devices train the model in the background, and there are no connectivity guarantees. The worker devices might disconnect mid-training, taking away their data, and since most will be connected via wireless links, data transmission can take considerably longer.

Although this approach has become popular in Next-Generation Network (NGN) applications, including in 5G and B5G, its costs and efficiency compared with the other approaches are still widely unknown and require a more in-depth study.

3. Related work

In this section, we overview the state-of-the-art in FL on 5G and B5G, identifying common FL methods and machine learning models used.

For example, in [12], the authors proposed a blockchain-empowered FL framework and presented some possible use cases in 5G. The FL architecture proposed relies on an asynchronous decentralized update scheme, as the authors identify that synchronous updates would consume too many resources and cause communication delays. Some proposed use cases for 5G are intelligent transportation, model training in mobile networks or IoT scenarios, and network data analysis.

In another perspective, in [13], the authors test both traditional and FL approaches in various models for predictive virtual network function autoscaling. The FL approaches are decentralized, and in one use case, the model is updated sequentially by ensuring that only one worker trains at a time with the latest model. On the other, the authors use the typical decentralized synchronous training where the parameter server averages the updated weights of all workers simultaneously. The models tested were all DNNs focused towards time-series processing. The results show that the best-performing model was trained using traditional learning, followed by the first FL approach presented.

Similarly, the authors of [14] propose a FL approach to solve a typical 5G problem. However, this work focuses on the physical layer, proposing a FL framework for channel estimation. The FL framework uses as a base the FedAvg algorithm, which implements a centralized synchronous approach for model updating. The model optimized in this work is a DNN capable of obtaining a normalized mean squared error of less than 0.001 while creating approximately 16 times less communication overhead when compared with data centralization.

Still focusing on the physical layer, in [15], the objective is to provide the FL workers/parameter server with enough bandwidth to perform the FL task, while at the same time keeping the process energy-efficient. The paper analyzes centralized synchronous and asynchronous approaches and shows a decrease of 31% and 14% in the total energy consumption of one FL iteration for each scheme, respectively.

In [16], the authors focus on another critical task in 5G/B5G, the energy-efficient resource allocation, leveraging FL to remove some of the computational load from the base stations. The authors proposed a specific reinforcement learning focused FL framework, the FRL_{suc}, a decentralized synchronous framework. The results show that the proposed framework outperforms other benchmark decentralized approaches and reduces the base station's computational load and communication cost compared to traditional learning.

Regarding the tasks we will address, [17] proposes a two-level FL framework to train a slice selection algorithm, in the first level, the authors opted for a decentralized synchronous approach to train the model for devices with the same features. Then, a vertical model aggregation was performed to improve models with devices that only share part of the features. The results show an improvement in communication efficiency of around 25% compared with other state-of-the-art approaches.

As for Intrusion Detection System (IDS), [18] presents a decentralized synchronous FL framework for the Internet of Medical Things where the authors leverage the devices that receive the traffic to train their models collaboratively without sharing their data. The results show that FL improves the overall model performance compared to traditional learning on the isolated devices.

Table 1 summarises the related work analysis, highlighting the FL approach and ML model used. Where the most common approach is decentralized synchronous FL applied to the training of DNNs.

Table 1
Summary of the related work.

REF	FL approach	Model
[12]	Decent. Async.	–
[13]	Decent. Async./Sync.	DNN
[14]	Cent. Sync.	DNN
[15]	Cent. Sync. /Async.	DNN
[16]	Decent. Sync.	DNN
[17]	Decent. Sync.	DNN
[18]	Decent. Sync.	DNN

4. Analyzed tasks

To provide the reader with some background on the tasks considered in this experiment, this section provides an overview of the tasks addressed and some typical ML models applied to solve them.

4.1. Network slicing

First introduced by the Next Generation Mobile Network Alliance [19], network slicing leverages network function virtualization and software-defined networks to create multiple independent virtual networks over the same physical one [20]. These networks are called slices and can be configured to offer specific network capabilities and characteristics, such as bandwidth, latency, and reliability. This is necessary to ensure a cost-effective way to meet the different consumers' heterogeneous requirements, as developing a network capable of achieving every requirement would be unfeasible [21].

For 5G, 3GPP defined three main Quality of Service (QoS) profiles for network slicing: the enhanced Mobile Broadband (eMBB), the Ultra Reliable Low Latency Communications (URLLC), and the massive Machine Type Communications (mMTC). The eMBB enables high throughput applications, such as high-definition video streaming or mobile TV, by creating a network with considerable bandwidth. The trade-off in this profile is that the applications are expected to be less latency-sensitive. In cases where a massive number of devices need to be connected (e.g. IoT), the profile to use is mMTC. Here, the devices are not expected to require high bandwidth or low latency, just the connection between them. Lastly, when consistent low latency and high reliability are mandatory, the URLLC profile should be used [22].

In network slicing, ML models are used for two primary use cases: slice management, where the models manage the resources attributed to each slice and how many slices of each type are needed, or end-user slice attribution, where the models decide to which slice the end-user requests go to.

Researchers in [23] delved into the issue of end-user admission, considering two distinct user categories: eMBB users and vehicular users. They employed a reinforcement learning agent using a DNN to tackle this challenge. Their primary goal was to optimize the reward, encompassing content caching efficiency in the fog-RAN, along with throughput and latency across the slices. Simulation outcomes unveiled the proposed mechanism's superiority over other baseline approaches, as evidenced by the improved cache hit ratio and maximize slice performance.

Using a different approach in [24], the authors sought to map end-user traffic flows to pre-existing slices using a supervised learning technique. To achieve this, the authors used the edge-based DNN method to predict the optimal slice for admitting traffic flow requests. Their primary goal was to optimize the successful transmission of requested traffic over network slices. Simulation results revealed that the proposed DNN-based approach outperformed random, round-robin, and Multi-Layer Perceptron methods regarding successful transmission rate and the amount of transmitted data.

In [25], the authors consider the three generic slice templates (URLLC, eMBB, mMTC) and employing a deep reinforcement learning approach, they dynamically allocate or reallocate users to slices based on their current requirements. To evaluate the proposed method, they set up an experiment simulation scenario with 100 users, each with requirements fitting into at least one of the three generic templates. Results demonstrate that their proposal consistently maintained a high average user satisfaction score throughout the experiment.

Table 2
Summary of the network slice attribution state-of-the-art.

REF	Dataset	Model
[23]	Simulation	DNN
[24]	Private	DNN
[25]	Simulation	DNN
[26]	Simulation	DNN
[27]	Simulation	DNN
[28]	Simulation	DNN

Delving into resource allocation in RAN slicing, researchers in [26] present a deep reinforcement learning approach to tackle the inter-slice bandwidth allocation challenge, considering the substantial traffic fluctuations between slices. To enhance convergence and address a broader action space, the authors integrated a discrete normalized advantage function into the deep reinforcement learning model. Numerical results show the proposed model's superior performance compared to the conventional deep reinforcement learning method.

Similarly, in [27], a prediction-based method employing a supervised DNN for spectrum allocation was proposed. The model aimed to optimize spectrum utilization, minimize spectrum allocation costs, and ensure desired service level agreements for users. Simulation results demonstrate the proposed method's ability to forecast traffic patterns accurately.

In [28], on the other hand, researchers introduce a resource allocation method based on deep reinforcement learning to allocate radio blocks to elastic and real-time applications in the smart grid for power data transmission. The aim is to increase spectral efficiency and resource utilization. Simulation results showcase that the proposed method outperforms state-of-the-art models regarding computational cost and resource utilization.

Table 2 summarises the network slice attribution state-of-the-art, highlighting the dataset and ML model used. Where most papers rely on simulations to train DNNs.

4.2. Intrusion detection system

The lack of security in IoT devices due to their limited resources and the constant increase in the complexity of IoT networks makes them more difficult to monitor and protect, creating a new challenge for the security community.

An effective way to protect IoT networks is to use IDS based on ML algorithms. The use of ML algorithms in IDS is a promising approach to detecting and preventing attacks in IoT networks by classifying the network traffic as usual or malicious and, in case of malicious traffic, identifying the type of attack.

To develop Machine Learning models capable of detecting attacks in this type of network, it is necessary to have a large dataset that contains network traffic data from IoT devices. However, most of the data generated within these networks is private and not available for research purposes, as it is mainly generated within personal or enterprise environments. In this scenario, FL is an excellent approach to enable model training without sharing private data from the networks.

Various research papers and work done in multiple datasets about IDS, where most models perform exceptionally well. While each paper delves into distinct datasets and employs different models, these findings remain pertinent to our research because they share a common theme and similar datasets.

In [29], the authors used XGBoost for imbalanced multiclass classification-based Internet of Things intrusion detection systems. The authors used two datasets, X-IIoTDS and TON_IoT, and achieved an F1 score of 99.9% and 99.87% respectively. XGBoost was used because of its advantages, including learning from its mistakes, fine-tuning extensive hyperparameters, scaling imbalanced data, and processing null values.

The authors in [30] used Feed-Forward Deep Neural Network (FFDNN) using a Wrapper Based Feature Extraction Unit (WFEU) in two datasets, UNSW-NB15 and the AWID. For the first datasets the model achieved overall accuracies of 87.10% and 77.16% for the binary and multiclass classification schemes, respectively and for the second dataset, the model obtained an overall accuracy of 99.66% when considering binary classification and 99.77% for multiclass.

The model used in [31] is a bidirectional Long-Short-Term-Memory (BiDLSTM) deep learning approach for intrusion detection. The model was trained and tested on the NSL-KDD dataset and achieved a training accuracy of 99.95%, and a testing accuracy of 94.26%. Overall the model obtained a higher detection accuracy for U2R and R2L attacks than the conventional LSTM model and has a much more reduced false alarm rate than the existing models.

In [32], the authors used a Support Vector Machine (SVM) with naïve Bayes feature embedding for intrusion detection. The authors used four datasets, UNSW-NB15, CICIDS2017, NSL-KDD, and Kyoto 2006+. The model achieved an accuracy of 93.75%, 98.92%, 99.35%, and 98.58% for the four datasets respectively. By embedding the data quality improvement technique into SVM, the authors can obtain an effective intrusion detection model with better performances and fewer complexities.

The authors in [33] used a Convolutional Neural Network (CNN) for anomaly-based intrusion detection systems for IoT networks. The authors used two datasets, NID and BoT-IoT, and achieved an accuracy of 99.51% and 92.85% respectively. This Deep learning approach was able to examine the traffic across the IoT to predict any possible intrusion and anomalous traffic behavior.

The model used in [34] is a hybrid model that combines a CNN and a Bidirectional Long Short-Term Memory (BiLSTM) for network intrusion detection. The model was trained and tested on the NSL-KDD and UNSW-NB15 datasets and achieved an accuracy of 83.58% and 77.16% respectively. To overcome the problem of data imbalance, the authors used the one-side selection (OSS) to reduce the noise samples in the majority category, and then increase the minority samples by Synthetic Minority Over-sampling Technique (SMOTE).

In [35], the authors proposed a K-Nearest Neighobor (kNN) classifier with an improved arithmetic optimization algorithm. To enhance the accuracy of the model, they use a parallel strategy to enhance the communication between the populations and use the Lévy flight strategy to adjust the optimization. This PL-AOA algorithm was tested on the WSN-DS dataset and achieved an accuracy of 99%.

In [36], the authors developed an efficient hybrid network-based IDS model (HNIDS), which is utilized using the enhanced genetic algorithm and particle swarm optimization(EGA-PSO) and improved random forest (IRF) methods. This model is capable of dealing with the data-imbalance issues and was tested on the NSL-KDD dataset and achieved an accuracy of 88.149% and 98.979% for the binary and multiclass classification schemes, respectively.

The model proposed in [37] is a hybrid deep learning model to efficiently detect network intrusions based on a CNN and a weight-dropped, long short-term memory (WDLSTM) network. The CNN is used to extract meaningful features from IDS big data and WDLSTM to retain long-term dependencies among extracted features to prevent overfitting on recurrent connections. This model was tested on the UNSW-NB15 dataset and achieved an accuracy of 98.43% and 97.17% for the binary and multiclass classification schemes, respectively.

The authors in [38] proposed a Hybrid Convolutional Recurrent Neural Network-Based Network Intrusion Detection System (HCRNNIDS), which is made of a CNN and a Recurrent Neural Network (RNN). The CNN performs convolution to capture local features, and the RNN captures temporal features to improve the ID system's performance and prediction. This model was tested on the CSE-CIC-DS2018 dataset and achieved an accuracy of up to 97.75% with 10-fold cross-validation.

Table 3 summarises the IDS state-of-the-art, highlighting the dataset and ML model used. The most common approach used is, similarly to

Table 3
Summary of the IDS state-of-the-art.

REF	Datasets	Models
[29]	X-IIoTDS & TON_IoT	XGBoost
[30]	UNSW-NB15 & AWID	DNN
[31]	NSL-KDD	DNN
[32]	UNSW-NB15, NSL-KDD, CICIDS2017 & Kyoto 2006+	SVM
[33]	NID & BoT-IoT	DNN
[34]	NSL-KDD & UNSW-NB15	DNN
[35]	WSN-DS	kNN
[36]	NSL-KDD	Random Forest
[37]	UNSW-NB15	DNN
[38]	CSE-CIC-DS2018	DNN

what was seen in network slicing, DNNs. One key difference is that most IDS papers rely on publicly available datasets instead of simulations.

The current state of IDS leans heavily towards the utilization of DNN. Various models have demonstrated impressive performances across diverse datasets. These approaches showcase the adaptability and effectiveness of DNNs in capturing complex patterns within network data, making them a leading choice in the field of IDS.

5. Experiments

Although FL has been used extensively in research, to the best of the authors' knowledge, the overhead that FL can have compared to other training approaches has yet to be analyzed. In this section, we define the experiments performed to understand if there is an overhead when training models with FL compared to traditional learning. The code used to obtain the results is available in GitHub.[2]

5.1. Datasets and models

This work's objective is to extract conclusions that can be generalized. As such, the test performed must consider different datasets. Two datasets were selected, one based on network slice attribution and one for intrusion detection systems. Two tasks that are strong candidates for using FL approaches due to their data privacy constraints and distributed nature, as shown in Sections 3 and 4.

5.1.1. Network slicing

The dataset considered for network slicing is proposed in [39] and available in IEEE-Dataport.[3] The objective of the dataset is to classify a request according to one of three slices (URLLC, eMBB, or mMTC) given a set of resource requirements and device characteristics.

The raw dataset is composed of eight input features and one output. The output is one of three classes, making this dataset a classification one. Seven of the input features are categorical, so as a preprocessing step, they were transformed with label encoding. The only exception is the *time* feature, which remained unchanged. After the preprocessing, the dataset comprised eight features and 466,739 examples. The examples were then shuffled and partitioned into two subsets: 80% for training and 20% for testing.

Furthermore, the training set was divided into two subsets: 80% for the actual training and 20% for validation. This results in a ratio of 64/16/20 for training, validation, and testing. The final dataset is composed of 298,712 examples for training, 74,679 for validation and 93,348 for testing. As the developed models will be trained following a traditional learning approach and using a federated learning approach, it was necessary to randomly divide the training set into eight equally sized subsets. In the end, each subset contains 37,339 examples.

[2] https://github.com/rgtzths/fl_costs_extended.

[3] https://ieee-dataport.org/open-access/crawdad-umkcnetworkslicing5g.

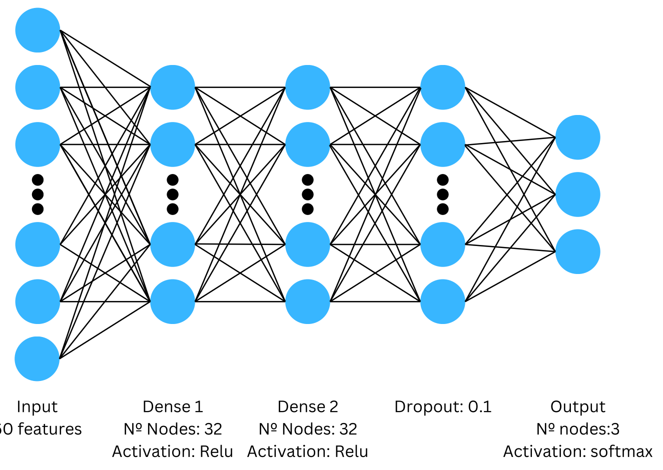


Fig. 4. Neural network used for network slice selection.

The model implemented was proposed in [40] and is the simple neural network with four layers presented in Fig. 4. The first and second layers contain 32 nodes and use the relu activation function. After them, there is a dropout of 0.1, which helps the network not to overfit. Finally, the output layer has three nodes and applies the softmax activation function. The loss function used is the sparse categorical cross-entropy. The implementation was done using the Keras API of TensorFlow.

During implementation, some decisions were made which are worth pointing out. Since the original paper did not mention which optimizer was used and most data parallel algorithms are SGD-based, we opted for the Adam optimizer with a low learning rate of 0.00001. Such a low learning rate is necessary to make the model take more epochs to converge. Otherwise, it would solve the task in a few epochs, and we would not be able to understand the costs of the different implementations. The second decision was removing two layers, the multi-category encoding and normalization.

5.1.2. Intrusion detection system

IoT-Device-Network-Logs (IoT-DNL) is a dataset tailored for network-based IDS. Initially introduced by Sahil Dixit and sourced from Kaggle,[4] this open-source dataset is designed explicitly for evaluating anomaly-based IDS in wireless environments. The dataset is flow-based and labeled, focusing on IoT devices. It has undergone preprocessing to suit the requirements of network-based IDS.

The network logs in this dataset were gathered by monitoring the network using ultrasonic sensors. Arduino and NodeMCU with the ESP8266 Wi-Fi module are employed as ultrasonic sensors to transmit data to the server via Wi-Fi. In total, the dataset comprises 477,426 instances, each with 14 variables.

The dataset contains various forms of malicious traffic or attacks, including Wrong setup, Distributed Denial of Serivce (DDoS), Data type probing (involving the transmission of string values to the server, primarily due to the use of ultrasonic sensors), Man-in-the-Middle (MITM), and Scan attacks, all monitored within the network.

Upon the analysis of the correlation between the features and the normality feature, it was possible to conclude that several features have a strong positive relationship with the normality feature, but frame.number and frame.time are highly correlated with each other but do not have any correlation with the normality feature. Therefore, it was decided to remove these two features from the dataset and keep the remaining ones, having, in the end, 11 features.

The division of the dataset followed the same procedure as the previous one, resulting in a ratio of 64/16/20 for training, validation,

[4] https://www.kaggle.com/datasets/speedwall10/iot-device-network-.

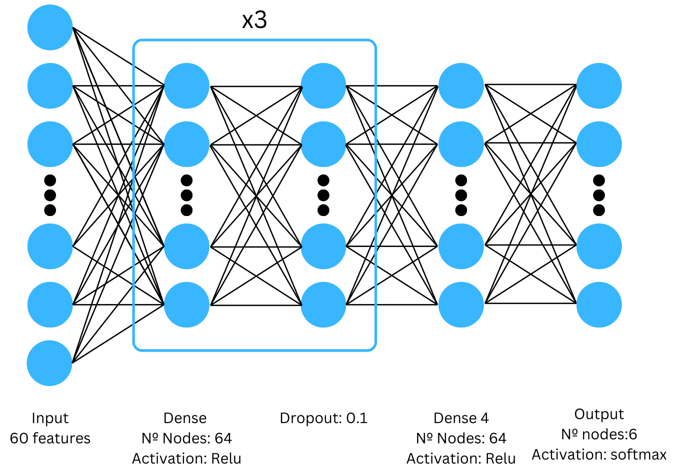


Fig. 5. Neural network used for intrusion detection.

and testing. The final dataset comprises 305,552 examples for training, 76,388 for validation and 95,486 for testing. Similarly to what was done for the previous dataset, the training set was also randomly divided into eight equally sized subsets. In the end, each subset contains 38,194 examples.

The model implemented was based on the model proposed in [41] and is the simple neural network with eight layers presented in Fig. 5.

The proposed model didn't specify the number of nodes in each dense hidden layer, so we considered it 64.

The six layers can be seen as three blocks composed of one dense layer with 64 nodes and a relu activation function and one dropout layer with a value of 0.1. After them, there is a dense layer with 64 nodes and a relu activation function, followed by the output layer with 6 nodes and a softmax activation function. The loss function used is the sparse categorical cross-entropy, and the optimizer is Adam, with a learning rate of 0.000005. Similarly to the previous model, a low learning rate is necessary to evaluate the costs of different implementations. The implementation was also done using the Keras API of TensorFlow.

5.2. Federated learning algorithms

Since this work aims to evaluate the impact of FL on the costs of implementing FL in its typical scenario, four federated learning algorithms were implemented, two synchronous and two asynchronous. The first uses decentralized optimization with synchronous communication [9], the second uses decentralized optimization with asynchronous [42], the third uses centralized optimization with synchronous communication [43], and the fourth uses centralized optimization with asynchronous communication [7].

It is also worth mentioning the hyperparameters used in the FL algorithms. The training performed with federated learning was done using eight, four, or two workers, and for the single host training, we used the parameter server. The hyperparameters for each approach are the following:

- **Centralized Synchronous:** 200 epochs, batch size of 128,
- **Centralized Asynchronous:** 200 epochs, batch size of 128,
- **Decentralized Synchronous:** 300 global epochs, 3 local epochs, batch size of 128,
- **Decentralized Asynchronous:** 300 global epochs, 3 local epochs, 0.2 updated bound for the master/worker, batch size of 128,
- **Single host training:** 200 epochs, batch size of 1024.

In the following sections, we detail the implementations of the federated learning approaches.

5.2.1. Centralized synchronous

As mentioned in section 2, centralized approaches require the worker to share the gradients instead of model weights. In this implementation, the workers share their gradients in each batch, and the parameter server does a weighted average based on the number of examples each worker has. Once the parameter server averages the gradients, he applies them to their local model and shares the updated model weights with all the workers. This implementation is synchronous because the parameter server waits for the gradients of every worker before applying them.

In the original approach, the authors used the concept of backup workers, where the gradients of the slowest worker are dropped. We decided not to use this feature and opted for never dropping the gradients as the testbed had only eight workers.

5.2.2. Centralized asynchronous

The asynchronous approach is similar to the synchronous one, as the parameter server still uses a weighted average when applying the gradients. However, instead of waiting for every worker to share their weights, whenever a worker shares the model weights, the parameter server applies them and sends the updated model just for the worker who shared the weights.

This implementation's downfall is that the only device with the most recent model is the parameter server and the last worker that shared the heights. Consequently, when applying the gradients, the model to which they are applied differs from the one in which they were obtained.

Nonetheless, if the learning rate is slow enough, the impact of a bad set of gradients is minimized.

5.2.3. Decentralized synchronous

The decentralized synchronous approach does a weighted average similar to the centralized approach, differing on what is averaged. Instead of averaging the gradients, the decentralized approach averages the model weights. This allows the workers to create the gradients for more than one batch and process multiple local epochs before sending the weights to the parameter server.

This added liberty has the downside of requiring the workers to be capable of doing the complete training epoch, which might not be possible when considering larger DNNs. Even so, the DNNs considered are relatively shallow, and the workers can do a complete epoch alone. Since the models can process more data before sending the results to the parameter server, the number of communication rounds is significantly reduced, creating less overhead on the network.

A critical aspect of this algorithm is that if the model does too many local epochs, it can drift significantly from the last model. Consequently, when the parameter combines the weights from the workers, the outcome can be sub-optimal and prevent the FL approach from converging.

5.2.4. Decentralized asynchronous

Like the asynchronous approach of the centralized algorithm, the decentralized asynchronous method applies each worker's weights whenever it receives them. However, this implementation does not do a weighted average of the model weights. Instead, it applies linear interpolation of the weights. Thus, when the worker shares the weights with the parameter server, it calculates the displacement for itself and the worker. The parameter server applies its displacement, which makes it get closer to the worker's position and sends the worker displacement to get it closer to the master.

This also means that the workers' models can be significantly different from the parameter server at each point in time and even be overfitted to their training data. However, the parameter server is, in theory, balanced as each worker has a limited impact on the direction the model takes.

5.3. Cost metrics

With this in mind, we selected various metrics to evaluate the proposed FL algorithms. The first one is Matthews's Correlation Coefficient (MCC), as regardless of how the model is trained, it must converge and achieve good performance.

The MCC [44] offers several advantages over other classification metrics like F1-Score and Accuracy, particularly when dealing with imbalanced datasets. MCC considers all the categories of the confusion matrix (as seen in Equation (1)), providing a more comprehensive evaluation of model performance. Unlike Accuracy, which can be misleading in imbalanced scenarios, MCC penalizes incorrect predictions across all classes, ensuring a balanced assessment.

While F1-Score is a useful metric that considers both precision and recall, it might not be as robust as MCC in certain situations. F1-Score can be heavily influenced by class imbalance, as it primarily focuses on the positive class. In contrast, MCC takes into account both positive and negative classes, making it less susceptible to class imbalance issues. Additionally, MCC's range from -1 to 1 allows for a clear interpretation, with 1 indicating perfect prediction, -1 indicating complete disagreement, and 0 representing random prediction.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{1}$$

The second metric is the execution time, as one of the objectives of FL is to reduce the time a model takes to converge. The third metric is communication overhead, as depending on the implementation, a model might make more communication rounds but use less network in each communication. This way, we measure how much data needs to be transferred on the network to achieve the desired performance and how much time the workers spend communicating.

5.4. Experiment environment

To evaluate the cost of using FL, we created a network of nodes to simulate the FL environment. Each node represents a device, and for this case of study, we considered nine nodes in total: 8 nodes for training named worker nodes and one central node for aggregation named parameter server. Each node is a Virtual Machine running Ubuntu 22.04.3, hosted on a GIGABYTE R120-T32-00 ARM server with 48 CPUs and 64Gb of RAM. Each training node was configured with four vCPUs, 6 GB of RAM, and 40 GB of disk space, which resembles a traditional edge device like a Raspberry Pi 4 in terms of computing power and CPU instructions. The parameter server was configured with eight vCPUs, 10 GB of RAM, and 40 GB of disk space. The nodes used the Message Passing Interface (MPI) protocol for communication.

6. Results

6.1. Baseline

The baseline in our experiments is the results of training a model using the traditional method on a standalone device. This is the typical ML approach where the dataset is centralized in one machine, and the training is performed using a single machine. The device it was trained on was the parameter server described earlier. Although the parameter server is slightly more powerful than the workers, the comparisons are still valid. If the results are similar to the ones seen in the literature, the expected performance increase of FL should be enough to surpass the baseline.

Regarding the slice attribution dataset, Fig. 6 displays the validation results obtained during training. As we can see, the model converges steadily, achieving an MCC of 1.0 in the validation set. Regarding communication, as expected, the device does not require any message exchange, so the final result is zero communication overhead. As for the

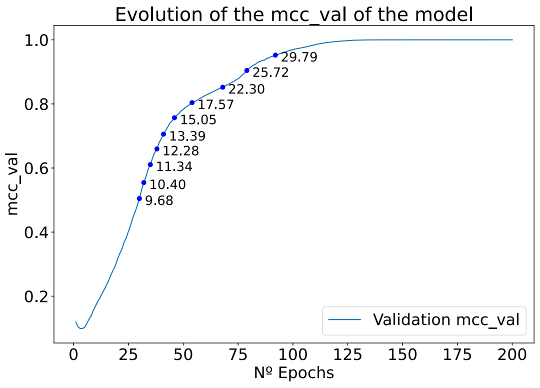


Fig. 6. Baseline learning curve for the slice attribution dataset.

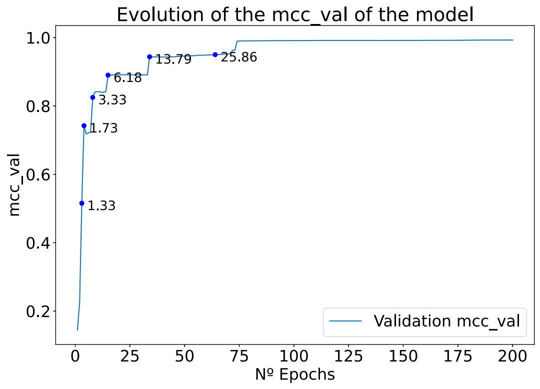


Fig. 7. Baseline learning curve for the IDS dataset.

time it takes to train, the model achieves 0.95 MCC at 29 minutes of training.

As for the IDS dataset, Fig. 7 displays the validation results obtained during training. As we can see, the model convergence is less smooth, although it still achieves an MCC of 0.99 in the validation set. Similar to the other baseline, it also does not have communication overhead, and regarding the time it takes to train, the model achieves 0.95 MCC at 25.86 minutes of training.

6.2. Federated learning

To understand the differences between FL approaches and provide a more detailed analysis, this section provides the user with an characterization of the time taken and communication generated by the different FL approaches to achieve various MCC thresholds. Furthermore, an analysis of the time spent by each worker on different tasks is also presented.

6.2.1. Network slice attribution

Starting with network slice attribution, it is clear that depending on the approach, the results can be widely different, as both centralized approaches took considerably longer than their decentralized counterparts, as presented in Table 4. The faster approaches when considering the communication time were always the asynchronous ones.

Regarding the communication exchanged, as displayed in Table 5, the results are not surprising. Since centralized approaches communicate per batch instead of per epoch, there would be many more communication rounds even if they ran the same number of epochs. Comparing the synchronous and asynchronous communications for the decentralized approach, one can also state that the faster approaches (asynchronous) converge slower, as more communication overhead means they use more epochs. This is the opposite of what is seen in the centralized approach.

Table 4
Results regarding execution time (in minutes) for eight workers in the 5GSlicing dataset. Missing values result from the model improving more than 0.1 in an epoch (global epochs when considering decentralized methods).

Performance (MCC)	Cent. Async	Cent. Sync	Decent. Async	Decent. Sync
0.50	–	35.66	2.11	3.48
0.55	20.26	39.23	2.35	3.71
0.60	–	41.64	2.58	4.17
0.65	24.30	45.21	2.82	4.40
0.70	28.34	48.77	3.05	4.86
0.75	32.46	54.73	3.40	5.55
0.80	36.53	64.29	3.87	6.26
0.85	48.69	80.93	4.82	7.89
0.90	56.80	95.17	5.77	9.74
0.95	68.94	110.64	6.84	11.63

Table 5
Results regarding communication exchanged (in MB) for eight workers in the 5GSlicing dataset. Missing values result from the model improving more than 0.1 in an epoch (global epochs when considering decentralized methods).

Performance (MCC)	Cent. Async	Cent. Sync	Decent. Async	Decent. Sync
0.50	–	845.59	1.54	1.45
0.55	140.93	930.14	1.74	1.54
0.60	–	986.52	1.93	1.74
0.65	169.12	1071.07	2.12	1.83
0.70	197.3	1155.63	2.32	2.03
0.75	225.49	1296.56	2.61	2.32
0.80	253.68	1522.05	2.99	2.61
0.85	338.23	1916.66	3.76	3.28
0.90	394.61	2254.89	4.54	4.05
0.95	479.16	2621.31	5.41	4.83

This can seem counterintuitive when looking at the training times seen in Table 4, since although the decentralized asynchronous approach should be slower in theory, as it took more epochs, the faster communication rounds made it take less time. As for centralized learning, besides being a faster communication scheme, the asynchronous approach converges faster than the synchronous one. This means that, depending on the approach, we can trade off speed for communication overhead.

In Fig. 8, the bar plots show the workers' average time on the various steps of the FL algorithms. A logarithmic scale was applied since most presented times were very small. That said, it is clear that in the decentralized approach, the synchronous version spends more time communicating and presents a similar training time. In the centralized approach, the asynchronous version takes longer to communicate. However, faster learning makes it a better approach when compared with the synchronous version, as it takes fewer epochs overall.

Since the send/receive is blocking, the time spent sending the model is expected to be considerably higher than the time spent receiving, as the parameter server could answer other worker requests or calculate the performance values. Nonetheless, it is crucial to note that the workers spend more time on the communication steps than training the model, regardless of the approach.

6.2.2. Intrusion detection system

As for the results in IDS task, presented in Table 6, although quite similar, some significant results must be highlighted. The first noticeable difference is that the models can converge fast to lower values of MCC since they achieve around 0.7 MCC in only one epoch but take longer to achieve the higher thresholds. The only exception to this rule is the centralized synchronous learner, which takes a couple of epochs to reach the 0.7 MCC value.

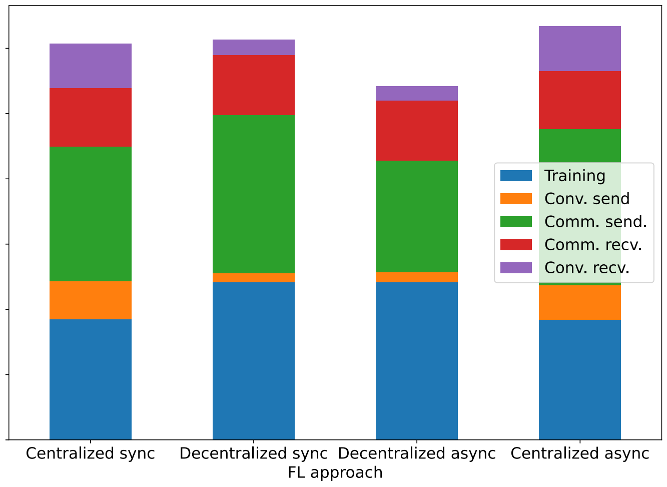


Fig. 8. Epoch time division by instructions using a logarithmic scale in the Slicing5G dataset.

Table 6
Results regarding execution time (in minutes) for eight workers in the IoT-DNL. Missing values result from the model improving more than 0.1 in an epoch (global epochs when considering decentralized methods).

Performance (MCC)	Cent. Async	Cent. Sync	Decent. Async	Decent. Sync
0.50	–	6.06	–	–
0.55	–	–	–	–
0.60	–	–	–	–
0.65	–	–	–	–
0.70	–	8.07	0.68	0.82
0.75	–	16.11	1.06	–
0.80	6.58	18.13	1.28	1.49
0.85	13.17	32.26	1.84	2.50
0.90	39.57	70.44	4.00	5.92
0.95	85.69	139.14	8.55	12.80

Table 7
Results regarding communication exchanged (in MB) for eight workers in the IoT-DNL dataset. Missing values result from the model improving more than 0.1 in an epoch (global epochs when considering decentralized methods).

Performance (MCC)	Cent. Async	Cent. Sync	Decent. Async	Decent. Sync
0.50	–	788.17	–	–
0.55	–	–	–	–
0.60	–	–	–	–
0.65	–	–	–	–
0.70	–	1050.89	1.76	1.76
0.75	262.72	2101.78	3.51	–
0.80	525.45	2364.51	4.39	3.51
0.85	525.45	4203.57	7.03	6.15
0.90	1576.34	9195.30	16.69	14.94
0.95	3415.40	18127.88	36.90	32.51

Looking at the communication overhead presented in Table 7, it is clear that the overhead is significantly higher. Since the model is considerably bigger than the one used in the 5GSlicing dataset, the amount of data communicated will be considerably higher even when using the same communication rounds. Besides that, the same trends can be seen in the 5GSlicing dataset here.

The bar plots in Fig. 9 show the same patterns regarding the various approaches. However, some differences are also noticeable. For example, centralized async is no longer the approach that takes longer to perform an epoch, which is, in this case, the decentralized sync. High-

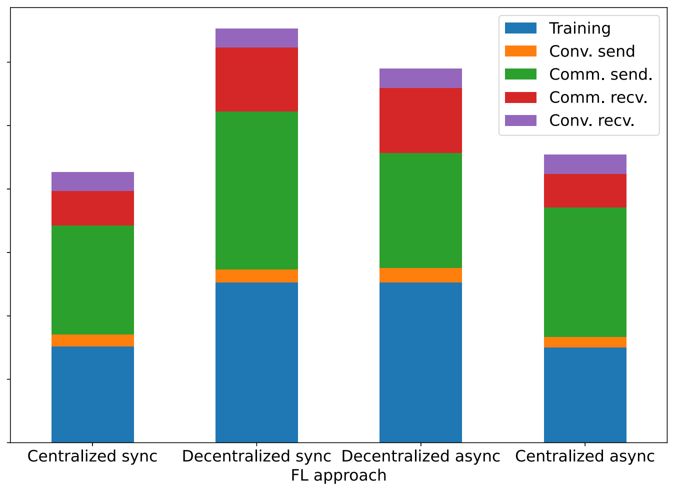


Fig. 9. Epoch time division by instructions using a logarithmic scale in the IoT-DNL dataset.

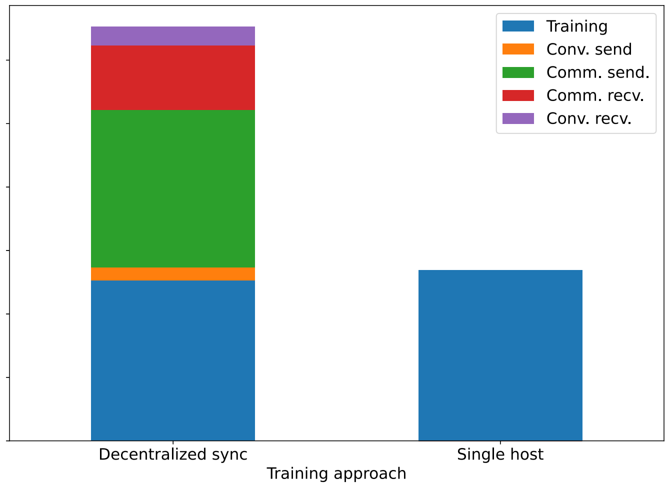


Fig. 10. Epoch time division by instructions for the best-federated models vs single host training in the IOT-DNL dataset.

Table 8
Acceleration comparison between single host training and the fastest federated approach for the IDS dataset.

Performance MCC	Execution time (min)		Acceleration (%)
	Single host	Federated	
0.50	1.33	–	–
0.55	–	–	–
0.60	–	–	–
0.65	–	–	–
0.70	1.73	0.68	60.69
0.75	–	1.06	–
0.80	3.33	1.28	61.56
0.85	6.18	1.84	70.23
0.90	13.79	4.00	70.99
0.95	25.86	8.55	66.94

Table 9
Acceleration comparison between single host training and the fastest federated approach for the slicing dataset.

Performance MCC	Execution time (min)		Acceleration (%)
	Single host	Federated	
0.50	9.68	1.54	84.09
0.55	10.40	1.74	83.27
0.60	11.34	1.93	82.98
0.65	12.28	2.12	82.74
0.70	13.39	2.32	82.67
0.75	15.05	2.61	82.66
0.80	17.57	2.99	82.98
0.85	22.30	3.76	83.14
0.90	25.72	4.54	82.35
0.95	29.79	5.41	81.84

lighting that the size of the model and dataset can significantly impact the communication overhead and the epoch duration.

6.3. *Comparison*

Regardless of how the FL approaches compare with themselves, we must understand their advantages over the traditional approach. That said, the remainder of this section compares the best FL approaches for each dataset with their single host training counterparts.

Starting with the IDS, the best FL algorithm is decentralized synchronous, and looking at Table 8, it is clear that the FL approach is much faster than its single host counterpart, achieving an average acceleration of 66.08%.

Comparing the time per epoch, presented in Fig. 10, shows that although the FL approach takes a similar time to perform the training part of the algorithm, overall, a FL epoch is considerably slower. This means that the speedup results from running fewer epochs (converging faster) than the single host approach, justifying the acceleration presented in Table 8.

In the slicing dataset, the results follow a similar trend. Looking at Table 9, where the single host is compared with the decentralized asynchronous approach, it is clear that FL presents a clear advantage over traditional learning. Obtaining an average acceleration of 82.87%, even more than the IDS result.

Similarly, when comparing the time per epoch, as displayed in Fig. 11, the single host continues to have a shorter time per epoch. This means that the FL can also converge in fewer epochs in this scenario.

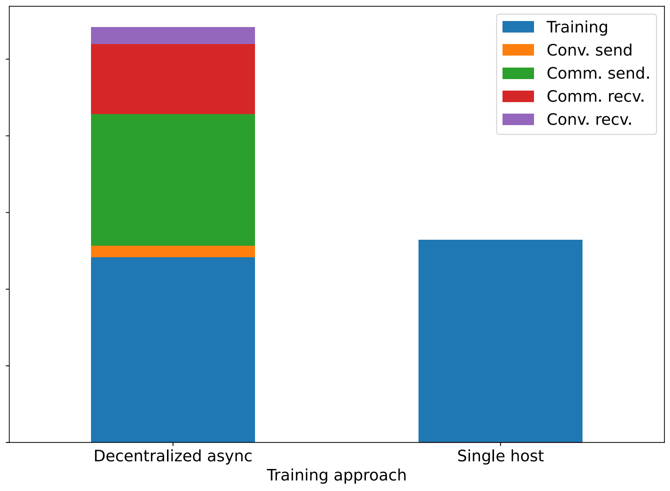


Fig. 11. Epoch time division by instructions for the best-federated models vs single host training in the Slicing5G dataset.

7. Discussion

With the results presented, it is clear that there are differences between datasets and also FL approaches. To better understand why these differences exist and why it is vital that they are addressed, this section discusses the results obtained.

7.1. Differences between the datasets

The first unexpected result was the shift in performance between the decentralized approaches between datasets. Although the datasets have a similar number of examples and the models used can achieve MCC of 0.99, the way the optimizer gets there is very different. Looking at Figs. 7 and 6, it is noticeable that the validation curve of the slicing dataset is much smoother than the one presented for the IDS.

This means that the optimization path in the IDS dataset is not as straightforward as in the slicing one. Consequently, the asynchronous approach has a more challenging time finding the optimal solution since it cannot simultaneously leverage every worker's exploration.

This difference is also seen when comparing the best FL algorithms with the single host training. Since the slicing dataset has an easier path for the solution, it also improves the convergence speed considerably, as the main difference is in the number of epochs performed, as we can infer from Figs. 10 and 6, where the difference in epoch time between the single host and the FL approach is very similar.

Another critical difference in the results is the communication overhead, which is directly influenced by the model size. Comparing the amount of information exchanged, one can state that, depending on the model and FL approach, the data consumed from the devices can start to be a concerning factor, as the centralized synchronous approach for the IDS system produce 2 GB of communications per worker to train the model, which is more than sharing the dataset itself.

7.2. Centralized vs. Decentralized learning

As for the differences between centralized and decentralized approaches, the results presented from this experiment are very similar to the ones we obtained in a previous study [45], where the centralized approaches' only advantage is that they require less demanding devices to perform the training. Nonetheless, the workers represent typical IoT devices, which shows that centralized learning approaches should only be used as a last resort.

This inefficiency of centralized approaches, as shown in the results, is not only tied to the number of communications performed, as the time per epoch is quite similar. The inefficiency comes from the inability to improve in each global epoch compared to the decentralized approaches. Taking more epochs to converge.

7.3. Why and when to use FL?

Although the results seem to favor FL as a better approach than single host training, some considerations must be taken to hold this statement as accurate. Considering the results obtained, especially the time per epoch, it is evident that the acceleration is a consequence of executing fewer epochs as FL is considerably slower than single-host training. That said, FL approaches are advantageous if they can converge faster than the single-host approach. Otherwise, the FL will be slower than single host training and can even generate more communication overhead than transferring the dataset itself.

Nonetheless, it is also necessary to address some of the study's limitations. The first one is the lack of analysis regarding non-iid data distribution, as the workers equally divided the datasets. This means that the FL considered is the best-case scenario for FL. Consequently, it helps with the previously stated condition for improved performance: the FL has to converge faster than the traditional approach. When using non-iid data, this might not happen as the learned models by each worker can be considerably different, taking longer to converge to a well-rounded global model.

The second limitation of the study pertains to device heterogeneity. In this scenario, every worker had the same computational power, which would not be precisely what is expected in the real world, as there are no guarantees of computational capabilities. That said, in the proposed scenario, no devices significantly delayed the model training, nor was there a need to implement any mitigation strategies regarding it. This again helps the performance of the FL approaches.

The last limitation is related to the communication bandwidth, as the devices never disconnected nor had any bandwidth limitations. This makes communication much faster than expected in the real world, as devices can achieve up to 5 Gb of bandwidth.

That said, compared to our previous study [45], which showed that FL was considerably slower than training on a single-host device. We were also able to show that it can be significantly faster. This again highlights that there should be a thoughtful consideration about applying FL as it is not a one-size-fits-all solution and has significant disadvantages. Nonetheless, one must remember why FL was proposed in the first place since it was never proposed as a performance-oriented approach to model training; rather, it was proposed as a privacy-oriented one.

With privacy in mind and the several restrictions seen to data collection in recent privacy-preserving laws such as GDPR, it is crucial to understand that even one might have to incur the overhead of implementing FL over other approaches to abide by the law. In those scenarios, it is crucial that the cost of the solution is understood and that we can assess if it is a viable approach. This study addresses this last point by showing that depending on the scenario, FL can incur extra costs or even be a more efficient approach, shedding light on the advantages and disadvantages of this training approach.

8. Conclusion

The main takeaway from this work is that FL approaches are not one-size-fits-all and that, even when considering datasets/models with similar characteristics, the results of the same FL approach can be considerably different. Furthermore, it is clearly shown that different FL approaches can have significantly different costs in training time and communication overhead.

The four FL approaches presented represent the main training approaches of FL, and the datasets used are a good starting point for the cost analysis. The results show that FL can substantially accelerate the training procedure, although some approaches can cause significant slowdowns.

Another important finding of this paper is the impact communication has on the efficiency of the FL algorithms, as it is shown that even the best FL approaches tested spend a considerable amount of time of an epoch in the communication exchange.

In future work, we intend to expand the experiments performed by considering heterogeneous devices and non-IID datasets and limiting the communication bandwidth of the devices to provide even more realistic results. At the same time, we also intend to quantify the average improvement per epoch of each FL approach. Furthermore, we intend to propose a FL approach that reduces the time spent communicating.

CRediT authorship contribution statement

Rafael Teixeira: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Investigation, Formal analysis, Data curation, Conceptualization. **Leonardo Almeida:** Writing – original draft, Software, Methodology, Investigation, Formal analysis, Data curation. **Mário Antunes:** Writing – review & editing, Validation, Supervision, Project administration, Investigation, Conceptualization. **Diogo Gomes:** Writing – review & editing, Validation, Project administration, Conceptualization. **Rui L. Aguiar:** Writing – review & editing, Validation, Supervision, Funding acquisition, Conceptualization.

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.

Acknowledgements

This study was funded by the PRR – Plano de Recuperação e Resiliência and by the NextGenerationEU funds at University of Aveiro, through the scope of the Agenda for Business Innovation “NEXUS: Pacto de Inovação – Transição Verde e Digital para Transportes, Logística e Mobilidade” (Project nº 53 with the application C645112083-00000059).

This work was supported by FCT - Fundação para a Ciência e Tecnologia, I.P. by project reference 10.54499/UIDB/50008 /2020, and DOI identifier https://doi.org/10.54499/UIDB/50008/2020.

Data availability

The data is shared in links inside the paper.

References

[1] W. Jiang, B. Han, M.A. Habibi, H.D. Schotten, The road towards 6g: a comprehensive survey, IEEE Open Journal of the Communications Society 2 (2021) 334–366, https://doi.org/10.1109/OJCOMS.2021.3057679.

[2] G. Flagship, 6g white paper on edge intelligence, https://5g-ppp.eu/wp-content/uploads/2021/05/AI-MLforNetworks-v1-0.pdf, 2021. (Accessed 15 September 2023).

[3] G. Flagship, AI-driven communication & computation co-design: initial solutions, https://hexa-x.eu/wp-content/uploads/2022/07/Hexa-X_D4.2_v1.0.pdf, 2022. (Accessed 15 September 2023).

[4] Hexa-X, Deliverable d5.1 initial 6g architectural components and enablers, https://hexa-x.eu/wp-content/uploads/2022/03/Hexa-X_D5.1_full_version_v1.1.pdf, 2022. (Accessed 15 September 2023).

[5] 5GPPP, AI and ml - enablers for beyond 5g networks, https://5g-ppp.eu/wp-content/uploads/2021/05/AI-MLforNetworks-v1-0.pdf, 2021. (Accessed 15 September 2023).

[6] M. Agiwal, A. Roy, N. Saxena, Next generation 5g wireless networks: a comprehensive survey, IEEE Communications Surveys and Tutorials 18 (3) (2016) 1617–1655, https://doi.org/10.1109/COMST.2016.2532458.

[7] M. Langer, Z. He, W. Rahayu, Y. Xue, Distributed training of deep learning models: a taxonomic perspective, IEEE Transactions on Parallel and Distributed Systems 31 (12) (2020) 2802–2818, https://doi.org/10.1109/TPDS.2020.3003307.

[8] J. Quevedo, A. Perdigão, D. Santos, R. Silva, R.L. Aguiar, 5gainer: taking the verticals into the 5g road, in: 2023 Joint European Conference on Networks and Communications & 6G, Summit (EuCNC/6G Summit), 2023, pp. 514–519.

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A.y. Arcas, Communication-efficient learning of deep networks from decentralized data, in: A. Singh, J. Zhu (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282, https://proceedings.mlr.press/v54/mcmahan17a.html.

[10] A. Desai, S. Chaudhary, Distributed decision tree v. 2.0, in: 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 929–934.

[11] M. Guillame-Bert, O. Teytaud, Exact distributed training: random forest with billions of examples, arXiv arXiv:1804.06755 [abs], 2018, https://api.semanticscholar.org/CorpusID:4938367.

[12] Y. Lu, X. Huang, K. Zhang, S. Maharjan, Y. Zhang, Blockchain and federated learning for 5g beyond, IEEE Network 35 (1) (2021) 219–225, https://doi.org/10.1109/MNET.011.1900598.

[13] T. Subramanya, R. Riggio, Centralized and federated learning for predictive VNF autoscaling in multi-domain 5g networks and beyond, IEEE Transactions on Network and Service Management 18 (1) (2021) 63–78, https://doi.org/10.1109/TNSM.2021.3050955.

[14] A.M. Elbir, S. Coleri, Federated learning for channel estimation in conventional and ris-assisted massive mimo, IEEE Transactions on Wireless Communications 21 (6) (2022) 4255–4268, https://doi.org/10.1109/TWC.2021.3128392.

[15] T.T. Vu, H.Q. Ngo, D.T. Ngo, M.N. Dao, E.G. Larsson, Energy-efficient massive mimo for serving multiple federated learning groups, in: 2021 IEEE Global Communications Conference (GLOBECOM), 2021, pp. 1–6.

[16] Z. Ji, Z. Qin, Federated learning for distributed energy-efficient resource allocation, in: ICC 2022 - IEEE International Conference on Communications, 2022, pp. 1–6.

[17] Y.-J. Liu, G. Feng, Y. Sun, S. Qin, Y.-C. Liang, Device association for ran slicing based on hybrid federated deep reinforcement learning, IEEE Transactions on Vehicular Technology 69 (12) (2020) 15731–15745, https://doi.org/10.1109/TVT.2020.3033035.

[18] Y. Otoum, V. Chamola, A. Nayak, Federated and transfer learning-empowered intrusion detection for iot applications, IEEE Internet of Things Magazine 5 (3) (2022) 50–54, https://doi.org/10.1109/IOTM.001.2200048.

[19] N. Alliance, Ngmn 5g white paper, https://ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf, 2015. (Accessed 15 September 2023).

[20] X. Zhu, J. Wang, Q. Lai, X. Luo, R. Ren, H. Lu, Research on 5g network slicing type prediction based on random forest and deep neural network, in: 2023 IEEE 8th International Conference on Big Data Analytics (ICBDA), 2023, pp. 154–158.

[21] S. Wijethilaka, M. Liyanage, Survey on network slicing for Internet of things realization in 5g networks, IEEE Communications Surveys and Tutorials 23 (2) (2021) 957–994, https://doi.org/10.1109/COMST.2021.3067807.

[22] H. Babbar, S. Rani, A.A. AlZubi, A. Singh, N. Nasser, A. Ali, Role of network slicing in software defined networking for 5g: use cases and future directions, IEEE Wireless Communications 29 (1) (2022) 112–118, https://doi.org/10.1109/MWC.001.2100318.

[23] H. Xiang, S. Yan, M. Peng, A realization of fog-ran slicing via deep reinforcement learning, IEEE Transactions on Wireless Communications 19 (4) (2020) 2515–2527, https://doi.org/10.1109/TWC.2020.2965927.

[24] T. Zhang, Y. Bian, Q. Lu, J. Qi, K. Zhang, H. Ji, W. Wang, W. Wu, Supervised learning based resource allocation with network slicing, in: 2020 Eighth International Conference on Advanced Cloud and Big Data (CBD), 2020, pp. 25–30.

[25] D. Shome, A. Kudeshia, Deep q-learning for 5g network slicing with diverse resource stipulations and dynamic data traffic, in: 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), 2021, pp. 134–139.

[26] C. Qi, Y. Hua, R. Li, Z. Zhao, H. Zhang, Deep reinforcement learning with discrete normalized advantage functions for resource management in network slicing, IEEE Communications Letters 23 (8) (2019) 1337–1341, https://doi.org/10.1109/LCOMM.2019.2922961.

[27] R. Abozariba, M. Kamran Naeem, M. Asaduzzaman, M. Patwary, Uncertainty-aware ran slicing via machine learning predictions in next-generation networks, in: 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), 2020, pp. 1–6.

[28] S. Meng, Z. Wang, H. Ding, S. Wu, X. Li, P. Zhao, C. Zhu, X. Wang, Ran slice strategy based on deep reinforcement learning for smart grid, in: 2019 Computing, Communications and IoT Applications (ComComAp), 2019, pp. 6–11.

[29] T.-T.-H. Le, Y.E. Oktian, H. Kim, Xgboost for imbalanced multiclass classification-based industrial Internet of things intrusion detection systems, Sustainability 14 (14) (2022) 8707.

[30] S.M. Kasongo, Y. Sun, A deep learning method with wrapper based feature extraction for wireless intrusion detection system, Computers & Security 92 (2020) 101752.

[31] Y. Imrana, Y. Xiang, L. Ali, Z. Abdul-Rauf, A bidirectional lstm deep learning approach for intrusion detection, Expert Systems with Applications 185 (2021) 115524.

[32] J. Gu, S. Lu, An effective intrusion detection approach using svm with naïve Bayes feature embedding, Computers & Security 103 (2021) 102158.

[33] T. Saba, A. Rehman, T. Sadad, H. Kolivand, S.A. Bahaj, Anomaly-based intrusion detection system for iot networks through deep learning model, Computers & Electrical Engineering 99 (2022) 107810.

[34] K. Jiang, W. Wang, A. Wang, H. Wu, Network intrusion detection combined hybrid sampling with deep hierarchical network, IEEE Access 8 (2020) 32464–32476.

[35] G. Liu, H. Zhao, F. Fan, G. Liu, Q. Xu, S. Nazir, An enhanced intrusion detection model based on improved knn in wsns, Sensors 22 (4) (2022) 1407.

[36] A.K. Balyan, S. Ahuja, U.K. Lilhore, S.K. Sharma, P. Manoharan, A.D. Algarni, H. Elmannai, K. Raahemifar, A hybrid intrusion detection model using ega-pso and improved random forest method, Sensors 22 (16) (2022) 5986.

[37] M.M. Hassan, A. Gumaei, A. Alsanad, M. Alrubaian, G. Fortino, A hybrid deep learning model for efficient intrusion detection in big data environment, Information Sciences 513 (2020) 386–396.

[38] M.A. Khan, Hcrnnids: hybrid convolutional recurrent neural network-based network intrusion detection system, Processes 9 (5) (2021) 834.

[39] A. Thantharate, R. Paropkari, V. Walunj, C. Beard, Deepslice: a deep learning approach towards an efficient and reliable network slicing in 5g networks, in: 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), 2019, pp. 0762–0767.

[40] D. Chowdhury, R. Das, R. Rana, A.D. Dwivedi, P. Chatterjee, R.R. Mukkamala, Autodeepslice: a data driven network slicing technique of 5g network using automatic deep learning, in: 2022 IEEE Globecom Workshops (GC Wkshps), 2022, pp. 450–454.

[41] D.K.K. Reddy, J. Nayak, B. Naik, G.S. Pratyusha, 11 Deep Neural Network–Based Security Model for Iot Device Network, Deep Learning for Internet of Things Infrastructure, 2021, p. 223.

[42] S. Zhang, A. Choromanska, Y. LeCun, Deep learning with elastic averaging sgd, in: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15, MIT Press, Cambridge, MA, USA, 2015, pp. 685–693.

[43] J. Chen, X. Pan, R. Monga, S. Bengio, R. Jozefowicz, Revisiting distributed synchronous sgd, arXiv arXiv:1604.00981 [abs], 2017.

[44] D. Chicco, G. Jurman, The advantages of the matthews correlation coefficient (MCC) over f1 score and accuracy in binary classification evaluation, BMC Genomics 21 (1) (jan 2020), https://doi.org/10.1186/s12864-019-6413-7.

[45] R. Teixeira, M. Antunes, D. Gomes, R.L. Aguiar, The learning costs of federated learning in constrained scenarios, in: 2023 10th International Conference on Future Internet of Things and Cloud (FiCloud), 2023, pp. 18–25.