applied sciences

MDPI

Article

Towards Mobile Federated Learning with Unreliable Participants and Selective Aggregation

Leonardo Esteves [1], David Portugal [2,*], Paulo Peixoto [2] and Gabriel Falcao [1]

[1] Instituto de Telecomunicações, Department of Electrical and Computer Engineering, University of Coimbra, 3030-290 Coimbra, Portugal
[2] Instituto de Sistemas e Robótica, Department of Electrical and Computer Engineering, University of Coimbra, 3030-290 Coimbra, Portugal
* Correspondence: davidbsp@isr.uc.pt

Abstract: Recent advances in artificial intelligence algorithms are leveraging massive amounts of data to optimize, refine, and improve existing solutions in critical areas such as healthcare, autonomous vehicles, robotics, social media, or human resources. The significant increase in the quantity of data generated each year makes it urgent to ensure the protection of sensitive information. Federated learning allows machine learning algorithms to be partially trained locally without sharing data, while ensuring the convergence of the model so that privacy and confidentiality are maintained. Federated learning shares similarities with distributed learning in that training is distributed in both paradigms. However, federated learning also decentralizes the data to maintain the confidentiality of the information. In this work, we explore this concept by using a federated architecture for a multi-mobile computing case study and focus our attention on the impact of unreliable participants and selective aggregation in the federated solution. Results with Android client participants are presented and discussed, illustrating the potential of the proposed approach for real-world applications.

Keywords: federated learning (FL); federated averaging (FedAvg); federated SGD (FedSGD); unreliable participants; selective aggregation

Citation: Esteves, L.; Portugal, D.; Peixoto, P.; Falcao, G. Towards Mobile Federated Learning with Unreliable Participants and Selective Aggregation. *Appl. Sci.* **2023**, *13*, 3135. https://doi.org/10.3390/app13053135

Academic Editors: George Drosatos, Pavlos S. Efraimidis and Avi Arampatzis

Received: 2 February 2023
Revised: 24 February 2023
Accepted: 26 February 2023
Published: 28 February 2023

1. Introduction

Mobile and wearable devices, as well as autonomous vehicles, are just some of the applications that are part of modern distributed networks that generate an abysmal quantity of data every day [1–3]. With increased computing power, energy efficiency, lower latency between communications, greater bandwidth, better data management solutions, and even storage capacity on these devices, as well as concerns about the disclosure of private information, it is becoming increasingly attractive to allocate data locally (e.g., on mobile devices) and push data processing to the edge so that users' private information is not exposed [4].

The concept of edge computing is not new. In fact, computing simple instructions in low-power, distributed devices is an area of research that is several years old. Recently, works have emerged, which consider training machine learning (ML) models centrally but serving and storing models locally in these distributed devices [5].

As the computational and storage capabilities of devices in a distributed network grow, local resources can be used on each device. This has led to an increase in interest in federated learning (FL) [6–8]. Federated learning is a machine learning technique in which an algorithm is trained on multiple decentralized edge devices or servers using local private data samples without sharing them. It has numerous applications, such as healthcare systems, industry, telecommunications, etc., as shown in Figure 1.

This concept shares some similarities with distributed learning. In distributed learning, the training model is distributed across different nodes, while FL assumes a decentralization

of both the data and the model and generates a global model that aggregates all the clients' models [9] (see Figure 2).

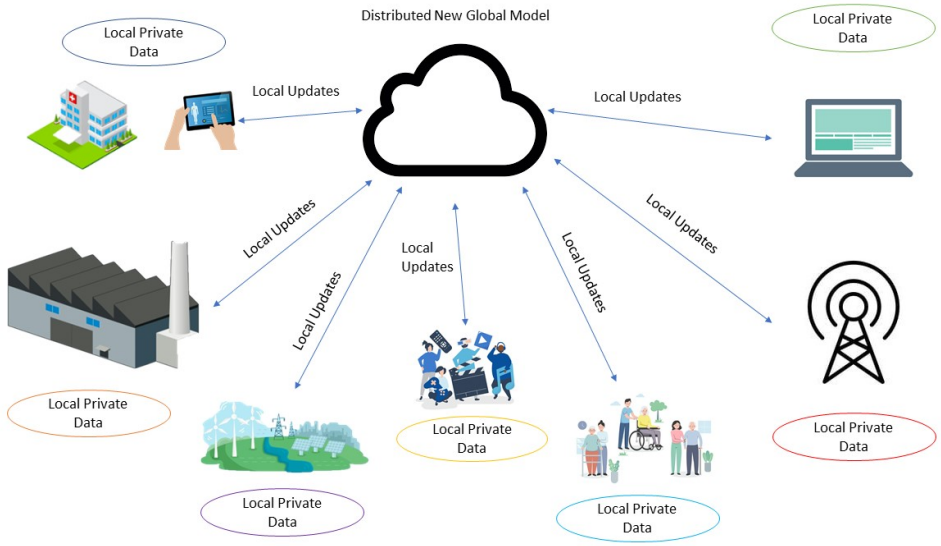

Figure 1. Federated Learning Applications.

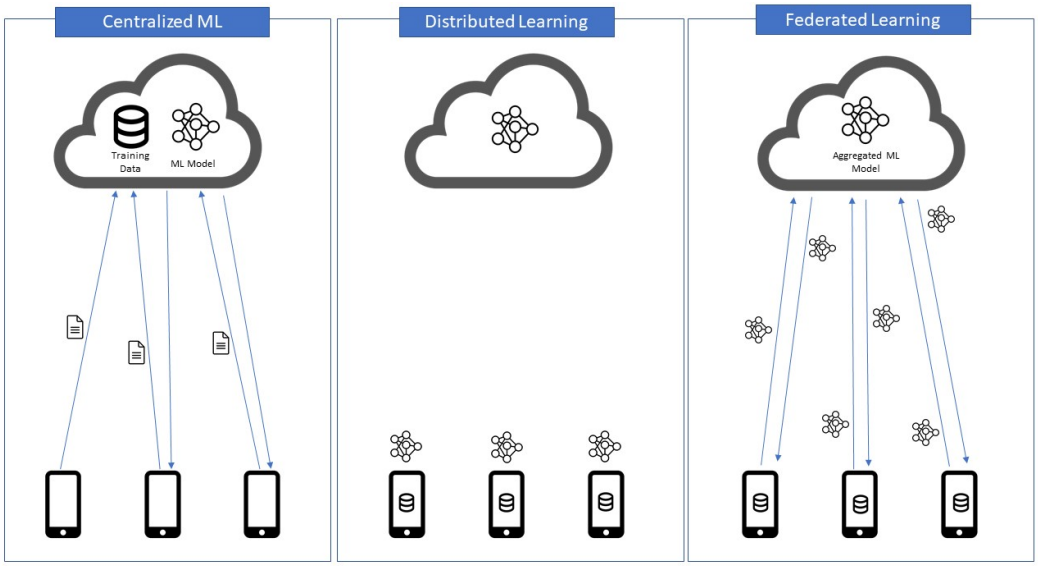

Figure 2. Difference Between Centralized Machine Learning, Distributed Learning and Federated Learning.

With the wide availability of mobile phones, we investigate the feasibility of implementing machine-learning-based techniques in these devices. Specifically, this paper explores federated learning for multimobile computing applications towards real-world deployment scenarios, by focusing on the implementation of a federated solution in Android devices. It also aims to preliminarily investigate different methods for aggregating and updating the global model managed by the central server, using unreliable client participants. In addition, the challenges and drawbacks of the FL process are discussed. Thus, the main contributions are:

- The implementation of a real-world FL application with up to eight Android mobile devices;
- A comparison of results obtained with the developed FL solution against a baseline obtained with a centralized ML approach;

- We demonstrate that a more advanced aggregation method, such as a weighted average instead of an arithmetic average, can significantly improve the overall result of an FL solution;
- Based on existing challenges, we conclude that eligibility criteria can benefit the FL framework in the presence of unreliable participants.

2. Background and Related Work

The federated learning concept aims to build a machine learning model based on data distributed across multiple sites [10]. It offers clients the ability to collaboratively train a common model without sharing personal data stored on their devices [8,11].

FL consists of two processes: model training and model inference. Typically, in an FL framework there are two main players: clients/workers/parties who contribute to the training of the model, and the server where the aggregation and update of the global model is performed. Thus, model training consists of sharing information between the workers and the server. However, the training data can never be shared. On the other hand, during the inference phase, the model is trained based on new data. Lastly, FL provides a mechanism to distribute the benefit of the whole process through all the collaborative parties. As described in [11,12], FL is a framework for building ML models that can be characterized by the following features:

- Two or more parties are interested in jointly creating an ML model. Each party has some data that will be used as input to train the model;
- It is imperative that during model training, the data kept by each party never leave that party;
- The locally trained model can be partially transferred from one party to another under an encryption scheme, so that other parties cannot reproduce the training data of a particular party;
- Theoretically, the performance of the resulting model is a reasonable approximation of an ideal ML model built using data from all parties.

More technically, let $\mathcal{N} = \{1, \ldots, N\}$ be the set of N parties, each of which has a private dataset $D_{i \in \mathcal{N}}$. Each data owner i uses its dataset D_i to train a local model w_i. The local model parameters are then forwarded to the FL server. All collected local model parameters $\mathbf{w} = \cup_{i \in \mathcal{N}} \mathbf{w}_i$ are then aggregated to create a global model W_G [13].

The conventional architecture of an FL system is illustrated in Figure 3. In that system, the parties jointly train an ML model with the assistance of an aggregate server. It is assumed that all parties involved in the process are trustworthy, which means that they use their private data to perform the training and pass the locally obtained model parameters to the FL server in each training round. Evidently, this assumption may not be practical, as discussed further ahead.

Typically, the FL process involves the following three steps [13].

- Step 1 (task initialization): The FL server determines the training assignment, i.e., the application goal and the corresponding data requirements. Moreover, the server determines the hyperparameters of the global model and the training process. Finally, the server transmits the initialized global model $\mathbf{w}_G^0$ to the selected parties.
- Step 2 (local model training and update): Following the global model $\mathbf{W}_G^t$, where t represents the current iteration index, each party individually uses its local data to update the local model parameters $\mathbf{w}_i^t$. The purpose of a party i in iteration t is to find ideal parameters $\mathbf{w}_i^t$ that minimize the loss function $L(\mathbf{w}_i^t)$, for example,

$$\mathbf{w}_i^{t^*} = arg \min_{\mathbf{w}_i^t} L(\mathbf{w}_i^t). \quad (1)$$

Then, the local model parameters are sent to the FL server.

- Step 3 (global model aggregation and update): the FL server aggregates the local models received from all selected parties and sends the updated model parameters $\mathbf{w}_G^{t+1}$ back to the participants.

Steps 2–3 are repeated until the global loss function converges or a desired training accuracy is achieved, i.e.,

$$L(\mathbf{w}_G^t) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{w}_i^t). \tag{2}$$

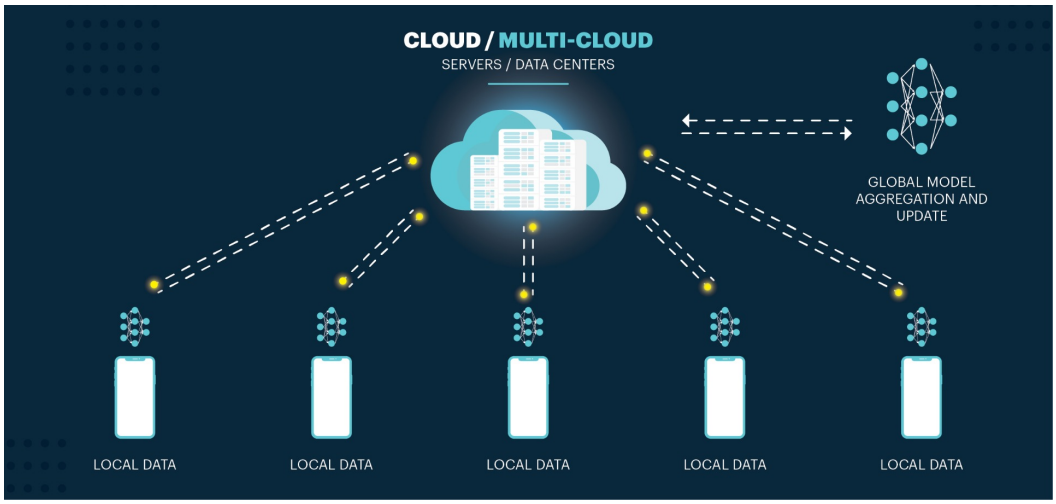

Figure 3. Federated Learning Architecture.

The FL training process can be applied to several ML models using the stochastic gradient descent (SGD) method. Typically, a training dataset contains a set of n data feature vectors $x = \{x_1, \ldots, x_n\}$ and a set of corresponding data labels. In the case of unsupervised learning, there is no data label. $y = \{y_1, \ldots, y_n\}$. Moreover, $\hat{y}_j = f(x_j; \mathbf{w})$ denotes the predicted result from the model $\mathbf{w}$ updated/trained by data vector x_j [13].

Most of the recent successful deep learning [14] applications use variants of SGD for optimization. In fact, many advances can be understood as an adaptation of the structure of the model (and hence the loss function) to facilitate optimization using simple gradient-based methods. Therefore, it is only natural that federated optimization algorithms are built based on SGD.

After training and updating the local model comes an essential part of the FL process: global model aggregation. There are several proposed solutions to this issue.

Experiments by Chen et al. [15] show that large-batch synchronous state-of-the-art SGD optimization outperforms asynchronous approaches, in a data center application. To apply this approach in a federated context, a C-fraction of clients/parties is selected at each round, and the gradient of the loss over all the data held by these parties is computed. Thus, C controls the *global* batch size, with $C = 1$ corresponding to full-batch (nonstochastic) gradient descent. While the batch size selection mechanism is different than selecting a batch by choosing individual examples uniformly at random, the batch gradients g computed by `FedSGD` still satisfies $\mathbb{E}[g] = \bigtriangledown f(w)$. The baseline algorithm is `FederatedSGD` (`FedSGD`) [14].

A typical implementation of `FedSGD` with $C = 1$ and a fixed learning rate η has each client k compute $g_k = \bigtriangledown F_k(w_t)$, the average gradient on its local data at the current model w_t, and the central server aggregates these gradients and applies the update $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$, since $\sum_{k=1}^{K} \frac{n_k}{n} g_k = \bigtriangledown f(w_t)$ [16]. An equivalent update is given by $\forall k$, $w_{t+1}^k \leftarrow w_t - \eta g_k$ and then $w_{t+1} \leftarrow \sum_{k+1}^{K} \frac{n_k}{n} w_{t+1}^k$. That is, each client locally performs a gradient descent step on the current model using its local data, and the server takes a weighted average of the resulting models. Within the algorithm, more computation steps

can be added to each client by iterating the local update $w^k \leftarrow w^k - \eta \bigtriangledown F_k(w^k)$ multiple times before the averaging step [14]. This approach is called `FederatedAveraging` (`FedAvg`).

The number of computational steps is controlled by three key parameters: C, the fraction of clients performing computations in each round; E, the number of training passes each client makes over its local dataset in each round; and B, the local batch size used for client updates. Selecting $B = \infty$ and $E = 1$ is equivalent to `FedSGD`. In `FedAvg`, for a client with n_k local examples, the number of local updates per round is given by $u_k = E\frac{n_k}{B}$ [14]. Algorithm 1 is a complete pseudocode of the `FedAvg` procedure.

Algorithm 1: `FederatedAveraging (FedAvg)`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \ldots$ **do**
$m \leftarrow max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow ClientUpdate(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w**):** *//Run on client k*
$B \leftarrow$ split P_k into batches of size B
for each local epoch i from 1 to E **do**
for batch $b \in B$ **do**
$w \leftarrow w - \eta \bigtriangledown l(w; b)$
return w to server

Typical FL solutions prioritize the privacy and security of all the parties involved. In fact, the server does not need to access individual information from each party involved to perform SGD. The server only requires the weighted averages of the update vectors. In [17], the authors proposed the secure aggregation protocol to compute these weighted averages. This method ensured that the server became aware of what types of data were present in the dataset, but without knowing the data that each user contained.

In mobile contexts, devices have limited resources in terms of energy and network connectivity. This introduces some unpredictability in terms of the number of parties that can participate in each round of updates, and the system must be able to respond to disconnections and new participants. Since ML models can be parameterized with millions of different values, model updates can involve a large quantity of data, which imposes a direct cost on users on metered network plans.

Secure aggregation proposes to work with high-dimensional vectors, uses efficient communication strategies, even considering the addition of new parties at each round and being robust to users dropping out, and finally, provides the strongest possible security measures under the constraints of a server-mediated, unauthenticated network model.

Pillutla et al. [18] introduced the use of the robust federated averaging (*RFA*) algorithm. RFA replaces the mean aggregation of *FedAvg* with a geometric median (GM) based robust aggregation oracle. Similar to *FedAvg*, *RFA* trades off communication for local computation by running multiple local steps. The communication efficiency and privacy preservation of *RFA* follow from computing the GM as an iterative secure aggregate.

Challenges

Some challenges distinguish the FL framework from other classical ML problems, such as distributed learning or traditional private data analytics [5]. Below, we describe four of the most important challenges identified for federated learning:

Expensive Communications

Client–client and client–server communication is a critical bottleneck in FL solutions [8]. When paired with privacy concerns over the transmission of raw data, it becomes necessary for the data generated on each side to remain local. It is undeniable that FL networks potentially include a huge number of devices, and network communications can be many orders of magnitude slower than local computations, due to limited resources such as bandwidth, energy, and power [19,20].

To fit a model to the data generated by devices in the federated network, it is important to develop communication-efficient methods that iteratively send small messages or model updates as part of the training process, rather than sending the entire dataset across the network. To reduce communication in such an environment, there are two key aspects to consider: minimizing the total number of communication rounds and the size of messages transmitted in each round [5,20].

Systems Heterogeneity

The storage, computational, and communication capacities of individual devices in federated networks may differ due to differences in hardware (CPU and memory), network connectivity (3G, 4G, 5G, and Wi-Fi), and power (battery level) [19]. In addition, the network size and systems-related constraints on each device typically result in only a small fraction of the devices being active at once. Moreover, it is not uncommon for an active device to lose connection with the server (dropout) in a given iteration due to connectivity or power constraints [8].

These system-level characteristics dramatically increase the number of challenges, such as straggler mitigation and fault tolerance [5]. Proposed FL approaches must therefore anticipate a low amount of participation, tolerate heterogeneous hardware, and be robust enough to dropped devices in the communication network [5,8].

Statistical Heterogeneity

Devices often generate and collect data that are not identically distributed across the network, e.g., smartphone users have a varied use of language in the context of a next-word prediction task [5]. Furthermore, the number of data points may vary significantly between devices, and there can be an underlying statistical structure present that captures the relationship among devices and their associated distributions [21]. This data-generation paradigm frequently violates the independent and identical distribution (i.i.d.) assumptions commonly used in distributed optimization and can increase the complexity of problem modeling, theoretical analysis, and empirical evaluation of solutions [5].

Although the canonical FL problem aims to learn a single global model, there are other alternatives such as learning different local models simultaneously via multitask learning frameworks [21]. In this respect, there is also a close connection between leading approaches for FL. This multitasking perspective allows personalized or device-specific modeling, which is often a more natural approach to handling statistical heterogeneity in the data for better adaptation [5,8].

Privacy Concerns

FL makes a step toward protecting data generated on individual devices by sharing model updates, e.g., gradient information, instead of raw data. However, sending model updates throughout the network during the training process can expose sensitive information, either to third parties or to the central server [22].

Although recent methods aim to improve the privacy of FL using tools such as secure multiparty computation (SMC) or differential privacy, these approaches often provide privacy at the cost of reduced model performance or system efficiency [17,22]. Understanding these tradeoffs, both theoretically and empirically, is a significant challenge in implementing private FL systems [5].

FL is already a case study for many applications. Simsek et al. [23] presented a study on AI-driven autonomous vehicles, and in [24], an energy efficient distributed analyt-

ics solution for IoT was proposed. In addition, Malekzadeh et al. [25] used FL for the transmission of sensor-data transformations.

The next section details the implementation of a mobile FL framework using Android devices as a case study, focusing on the impact of unreliable participants and selective aggregation methods in a multimobile computing application. Since this is the focus of the work, we do not explicitly address some of the challenges mentioned above, such as client heterogeneity or privacy concerns.

3. Proposed Federated Learning Infrastructure

In what follows, we describe an implementation which aims to serve as a guide to design a real-world multimobile FL solution.

3.1. Training Parameters

For our FL implementation, there were 4 controllable variables that allowed the testing of various training scenarios.

The first parameter (**num_clients**) represented the total number of clients participating in the federated solution. The dataset was divided among these clients so that each client received a predetermined number of dataset elements.

The second parameter represented the number of rounds that the training should take (**num_rounds**), and the third parameter represented the total number of local training rounds on each selected client's device (**epochs**). The fourth and last parameter was the batch size (**batch_size**). Data were loaded per batch, so it was important to choose a value that did not allow too little data to be loaded at once, as this would degrade training performance by not utilizing the full processing power of the client devices. However, it was also important not to select a value that was too high, as this could lead to data overload on the clients.

After the initial definition of **num_clients** and **num_rounds**, in each communication round (a communication round in federated learning represents the server–client and client–server data exchange associated to the training performed at each client), training on the client's devices with the local dataset took place, according to the number of **epochs** and the **batch_size**. Then, the server aggregated the individual models by weighting the corresponding parameters into one global model.

Below, we present the training parameters chosen in this work. The values for **num_rounds** were set empirically, since for the chosen dataset (see Section 5), we achieved an accuracy after 100 rounds that was approximately equal to the maximum achieved by Zhu et al. [26]. The **num_clients** was limited by the resources available for Android deployment in our work. The values of **epochs** and **batch_size** were fixed to eliminate their influence on the results and to allow a comparison of different configurations under the same test conditions.

```
num_rounds = {10,30,500,100}
num_clients = {2, 4, 8}
epochs = 5
batch_size = 32
```

3.2. Model Architecture

The chosen architecture of the neural network was **VGG19**. This network can be seen as a successor to **AlexNet** [27], which builds on its predecessors and improves them by using deep convolutional neural layers to achieve a higher accuracy [28].

The architecture of the VGG19 network provides 19 weight layers (16 CNN layers, 3 fully connected layers, and a final layer for a softmax function) as seen in Table 1. It assumes 224 RGB channels as input, leading to a 3D matrix of size $224 \times 224 \times 3$. In a first preprocessing stage, the average RGB value calculated for the whole training set is subtracted from each pixel. A 3×3 kernel with a stride size of 1 pixel is used, allowing the entire image to be covered. Spatial padding is used to preserve the spatial resolution of the

image. A max-pooling is performed over a 2×2 pixel windows with a stride of 2. This is followed by a rectified linear unit (**ReLu**) to introduce a nonlinearity that improves the classification performance of the model and the computation time compared to previous models that used **tanh** or **sigmoid** functions. Finally, there are three fully connected layers. The first two have a size of 4096 and the last one has 1000 channels. The final layer is a **softmax** function. This network is widely used for image classification in various fields, such as face recognition.

Table 1. Configuration of the VGG-19 Network.

Input (224 × 224 RGB Image)
conv3-64 + ReLu
conv3-64 + ReLu
maxpool
conv3-128 + ReLu
conv3-128 + ReLu
maxpool
conv3-256 + ReLu
conv3-256 + ReLu
conv3-256 + ReLu
conv3-256 + ReLu
maxpool
conv3-512 + ReLu
conv3-512 + ReLu
conv3-512 + ReLu
conv3-512 + ReLu
maxpool
conv3-512 + ReLu
conv3-512 + ReLu
conv3-512 + ReLu
conv3-512 + ReLu
maxpool
FC-4096 + ReLu
FC-4096 + ReLu
FC-1000 + ReLu
softmax

3.3. Training the FL Model

At this stage, a global model and the individual client models were initialized. In this solution, SGD was used as an optimizer for all client models. The training process took place within a cycle, i.e., training was performed by the clients before each round of communication. After receiving the updates from the local modules, the aggregation of the weights was performed on the server according to a specific aggregation method, which updated the global model that was then used to validate the training. This process continued for **num_rounds**, i.e., 100 communication rounds for all the tests performed. Accordingly, Algorithm 2 summarizes the FL training process in our implementation.

Algorithm 2: Training steps of our FL implementation.

```
for i in num_rounds do
    for j in num_clients do
        Perform local training
    end
    Send the global model to the server
    Aggregate the global model
    Validate the global model
end
```

3.4. System Design

The server–client and client–server communication was done through a REST server. Since we focused exclusively on the impact of unreliable participants and alternative aggregation methods, a REST server presented a simple way to communicate through POST messages with a JSON formatted body. This allowed us to perform a proof-of-concept study while ignoring aspects that were outside the scope of this work, such as protecting clients and the server from external attacks.

Another major point of our solution was the implementation of all the training logic on several clients using a mobile Android application.

The logic implemented on the server included a function that aggregated the weights coming from the clients and a function to test the updated global model. Figure 4 presents a high-level diagram of the interaction between server and clients in the proposed solution.

Finally, it only remains to define the last point of our solution: the deployment of clients on Android devices. Acquiring multiple devices to test this solution would represent a significant investment. Therefore, we decided to use Android mobile device emulation. There are numerous emulation software applications, such as BlueStacks (https://www.bluestacks.com/, accessed on 28 February 2023), Nox Player (https://www.bignox.com/, accessed on 28 February 2023), MEmu Play (https://www.memuplay.com/, accessed on 28 February 2023), etc., as well as Android Studio (https://developer.android.com/studio, accessed on 28 February 2023) itself where the Android application (`.apk`) development is done.

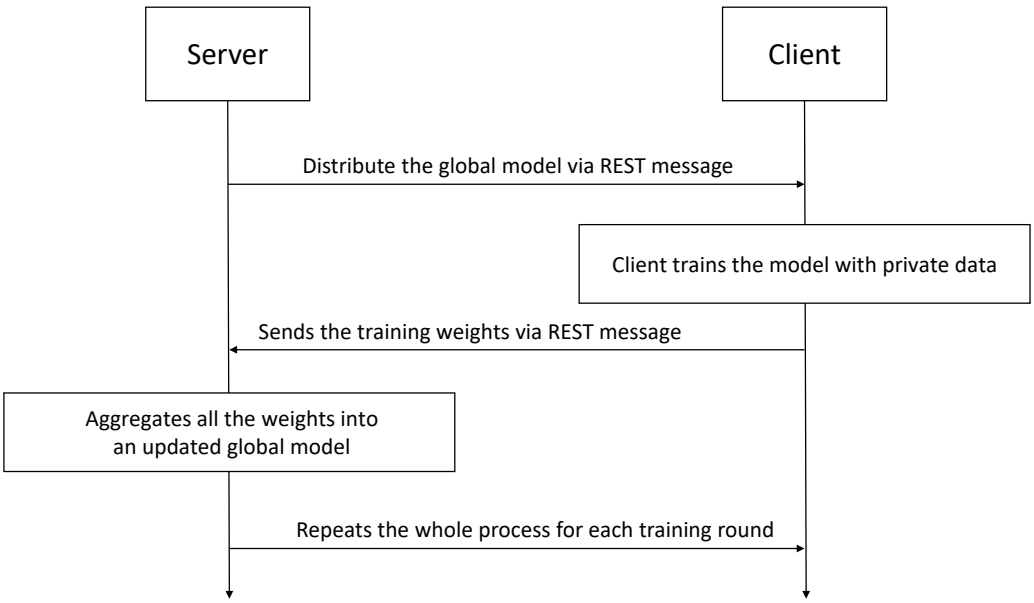

Figure 4. Federated Learning Server–Client Interaction in the Proposed Solution.

3.5. Development and Implementation

In this section, we first describe the implementation of the FL server and then we provide details about the implementation of the mobile clients.

REST Server (Server Implementation)

As seen before, the REST server started the communication rounds and monitored whether all updated weights were sent to the server by the clients. This server also sent the updated model to all clients.

The services supported by the REST API were as follows:

- A GET method to retrieve the latest shared model on the server;
- A POST method to upload the updates from the clients;
- A GET method to retrieve the information about the current round.

The first training round started immediately after the server was initialized. Once the local training was finished, the server asynchronously received updates from all clients. When a client sent the updated weights to be aggregated to the global model, the server stored them in a file in JSON format. This file contained all the information about each client's training rounds known so far.

After the server received information from all clients, it proceeded to aggregate the weights to create the global model. In a production environment, the server should have protection mechanisms to prevent it from receiving new information during the model aggregation process, as this can lead to system degradation. It was assumed that all available clients participated in each training round. Therefore, this paper did not consider the elimination or addition of clients during the training process.

Once the model aggregation was complete, the server sent it back to all participating clients in response to the previously received update message.

Android Application (Client Implementation)

An Android client application was developed using Android Studio based on the above-mentioned strategy. A functional `.apk` file was created that allowed any user to install the application for local training on an Android mobile device.

This application had to be able to communicate with the server. To do this, each client sent a REST message to an endpoint created for the server described above and with a known IP address. The clients sent weights from the local training and received in a response the updated global model to be used in the next round.

The client side implementation consisted of four phases:

- Setting up the connection to the server and loading the client dataset into memory;
- Receiving the global model through a REST message;
- Performing training on the client dataset;
- Sending the resulting weights to the server through a REST message.

The training function was developed using TensorFlow Federated for Android (https://www.tensorflow.org/federated, accessed on 28 February 2023). For our implementation, tests were initially performed with Android devices emulated in Android Studio. These tests showed that the software application would consume large amounts of computational resources and would therefore affect scalability with a larger number of clients. After further testing with alternative solutions, BlueStacks software was selected as it proved to scale better.

Like any other emulator, BlueStacks creates a virtual version of an Android device that runs locally on any computer. On this virtual device, the application was installed to run the developed FL client.

The multi-instance manager lets a user create multiple instances of BlueStacks. Using this feature, one can emulate several Android devices to test scenarios with multiple clients.

Ideally, one should consider the computational capacity and energy efficiency of each client, since in most real-world cases, client devices are heterogeneous. This has a significant impact on training, as there are clients that perform their local training faster

than others. This not only affects the total training time but also the ability of each client to train with a considerably large dataset. However, in our work, all devices were considered homogeneous. The effect of using training devices with different capabilities is therefore reserved for future work.

4. Evaluation Design

The final goal of this paper was to implement a multimobile Android-based solution for federated learning. To reach this goal, we outlined a succession of tests to perform.

First, the dataset to be used for experimental evaluation was selected, and a baseline method was run to serve as a reference for the remaining results. Then, tests were run with 2, 4, and 8 clients, with the selected dataset evenly distributed

In a real-world implementation, clients do not have the same quantity of data available for training. Therefore, we also tested an uneven distribution of the dataset across all clients to get a clearer picture of how a federated solution would perform in such scenarios.

In all the tests described previously, the arithmetic mean was used as the aggregation method. Alternatively, we repeated these tests using an aggregation method that used a weighted average. The weighted average benefited clients with a larger dataset over ones that had less data, so conclusions about the effects of selective aggregation could be drawn by directly comparing the results.

We then performed cross-validation tests that allowed us to test the generalization capability of the implemented solution, i.e., we evaluated the results obtained with different parts of the dataset.

Finally, we experimented with another, more complex dataset to validate the implemented solution and the proposed alternative aggregation method. Figure 5 shows a diagram of the experimental methodology for the results discussed in Section 5.

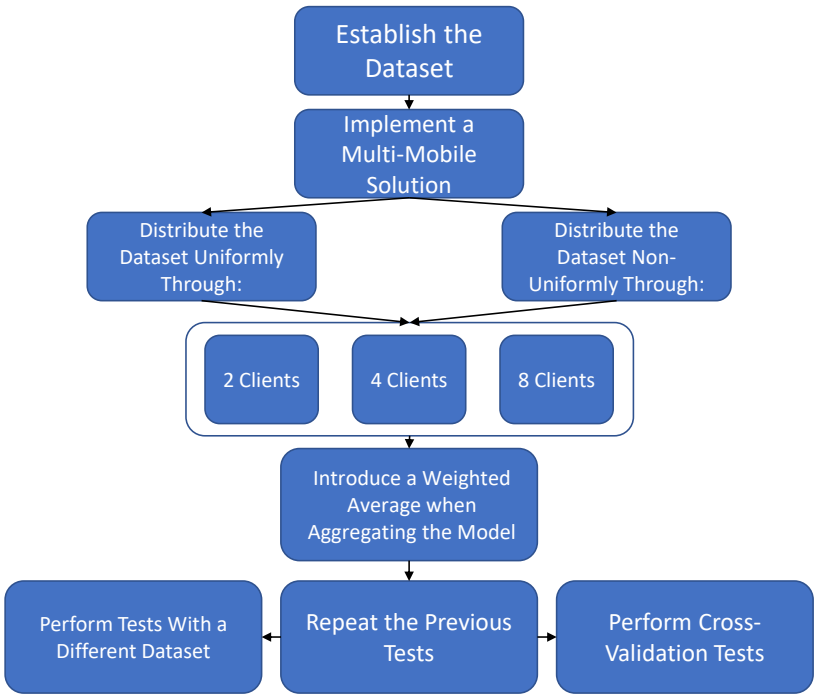

Figure 5. Experimental Methodology.

In all experiments, the performance measure used was the average accuracy after multiple rounds. This was calculated by the server, which progressively recorded the arithmetic mean of the accuracy achieved in each round. Moreover, each test configuration was repeated for five trials.

5. Results and Discussion

After a thorough survey of existing datasets, the CIFAR10 dataset was selected due to its appropriate balance between complexity and size. This allowed a clear analysis of the accuracy curve per round of an FL solution, while ensuring that the tests performed did not take too long and thus a reasonable number of configurations could be tested.

The CIFAR10 dataset consists of 60,000 color images of 32 × 32 pixels, divided into 10 different classes. There are 50,000 images available for training and 10,000 images for testing [29].

For all tests performed, the parameters mentioned in Section 3.1 were used.

5.1. Machine Learning Baseline Test

To make accurate comparisons between the models, a baseline ML method was run locally with the same parameters, i.e., the same dataset (CIFAR10), the same neural network (VGG 19), the same training rounds (100), epochs (5), and batch size (32).

According to [26], the VGG19 network can achieve up to 94.71% accuracy on CIFAR10 with sufficient training rounds. Based on this, our preliminary test was performed on a local machine to establish a baseline. Note that we used 100 rounds of training, as this was an adequate balance between training time and achieved accuracy.

After running these tests with the above parameters, an average accuracy of 92.87% was achieved. That is, we assumed 93% as the baseline value for the locally achieved accuracy.

5.2. Uniform Distribution of the Dataset

Here, we performed tests while splitting the dataset evenly across the Android devices (see Table 2).

Table 2. Average accuracy results obtained with 2 devices for a uniformly distributed dataset.

Elapsed Rounds	Average Accuracy
10	79%
30	88%
50	90%
100	91%

Although the training time was not analyzed, it is important to note that the average training time for each test performed in this scenario was about 24 h. This long training time was quite expected since the server and the Android devices were emulated on the same machine. In an ideal scenario, each node computes only its own training, and this is done entirely in parallel.

This first experiment was conducted to verify whether an FL solution could achieve a similar maximum accuracy as the same solution using centralized ML. From Table 2, it can be seen that the maximum accuracy achieved was close to the maximum accuracy mentioned in the previous subsection.

5.3. Nonuniform Distribution of the Dataset

Next, we examined how a client with an incomplete dataset affected the accuracy across rounds. In this work, only the set of images available for local training at each client determined whether the dataset was incomplete. In general, a dataset may be incomplete under various conditions, such as the quality of the images used, e.g., blurred images, pixelated images, etc., which is reserved for future work on this topic.

With this in mind, three test cases were conducted (see Table 3). In the first case, two clients performed the training, one using half of the CIFAR10 dataset (25,000 images) and the second using only 10 images (one image for each class in CIFAR10).

Table 3. Average accuracy for nonuniformly distributed datasets with 2 devices (one with 50% of CIFAR10 images and another with 0.0002%), 4 devices (15%, 25%, 60%, and 0.0002% of CIFAR10 images) and 8 devices (two clients with 7.5%, two with 12.5%, two with 30%, and two with 0.0002% of CIFAR10 images).

	# Devices		
Elapsed Rounds	**2**	**4**	**8**
10	27%	22%	12%
30	34%	43%	26%
50	42%	78%	44%
100	48%	90%	89%

In the second case, the number of clients was increased to four. The dataset was divided unevenly among them, but one of the clients saw its dataset allocated with only 0.0002% of the full training batch of CIFAR10 (one image for each class in CIFAR10). The other three clients had 15%, 25%, and 60% of the dataset, respectively.

In the last scenario, the number of clients was increased to eight. Two clients had 7.5% of the dataset, two had 12.5%, and two had 30%. The remaining two clients had 10 images each, i.e., one image for each class in CIFAR10.

The first case led to very poor accuracy results. This was to be expected since the training was performed on only half of the entire dataset. The second and third cases showed that increasing the number of clients and consequently each client having a smaller local dataset, together with introducing clients that hardly contributed to the overall training, still allowed an accuracy close to the maximum achieved in previous tests. However, it could also be observed that the accuracy in the first training rounds was significantly worse than in the previously presented case.

From this point on, no more tests were performed with just two clients. In order to test the implemented solution for as many clients as possible, we proceeded only with tests with four and eight clients.

5.4. Proposing a Weighted Average for Model Aggregation

The preceding results demonstrated the need to use an aggregation method that benefited those clients that contributed the most to the global model. This could be achieved by giving more weight to the contributions of clients with a more complete dataset, i.e., clients that predictably contributed more to a higher accuracy of the model.

Therefore, a new aggregation model was tested to meet this premise. Instead of an arithmetic average with equal weights for each client, we introduced a weighted average where the weights were computed as a fraction of the number of images each client had locally compared to the total number of images used for training.

Equation (3) represents the weighted average used hereafter.

$$w \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w^k, \tag{3}$$

where n represents the total number of images considered for training in all clients in a round and n_k represents the number of images in each client k. With this aggregation method, little importance is given to clients with a small number of images.

Once again, we performed tests with a nonuniform distribution of the dataset with four clients: one client with 15%, one with 25%, one with 60% of the CIFAR10 images, and one client with only 10 images; and with eight clients: two with 7.5%, two with 12.5%, two with 30% of the dataset, and two clients with 10 images for each class of CIFAR10 (see Table 4).

Figure 6 shows that using a weighted average in the aggregation of the global model improved the accuracy of the training in the first rounds compared to an aggregation method using an arithmetic average with both four and eight clients. However, given a sufficient number of rounds of training, nearly identical accuracy was achieved at the end of training in both cases. As expected, training accuracy converged faster in both cases when a weighted average was used, as clients with large assigned datasets were favored over clients with only a few images that had almost no effect on model aggregation.

To further evaluate the performance of our solution for as many clients as possible, only the tests with eight clients were performed in the following sections.

Table 4. Average accuracy for a weighted average aggregation method using nonuniformly distributed datasets with 4 devices (15%, 25%, 60%, and 0.0002% of CIFAR10 images) and 8 devices (two clients with 7.5%, two with 12.5%, two with 30%, and two with 0.0002% of CIFAR10 images).

	# Devices	
Elapsed Rounds	**4**	**8**
10	73%	36%
30	84%	55%
50	88%	79%
100	91%	90%

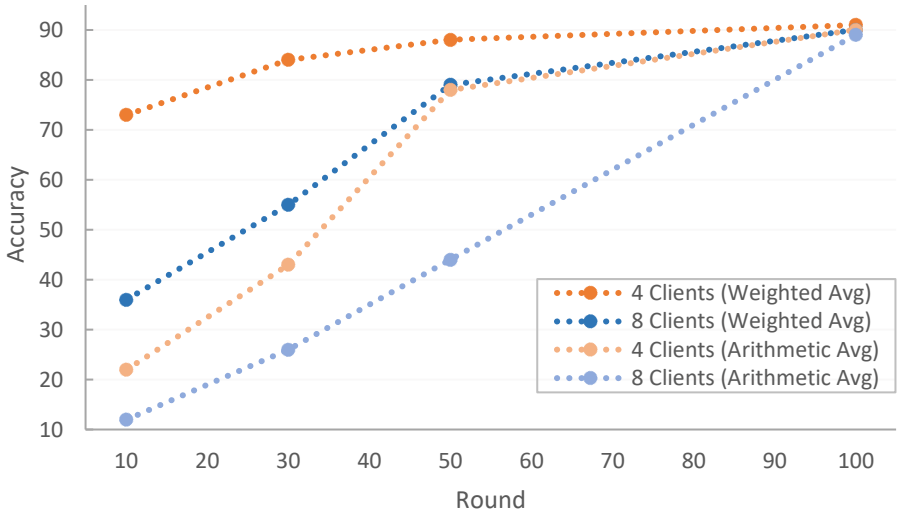

Figure 6. Training 100 Rounds with 4 and 8 Devices (Weighted Average vs Arithmetic Average).

5.5. Cross-Validation

A cross-validation test was performed to obtain a more comprehensive evaluation of the proposed approach. In this case, the 50,000 training images were divided into five batches of 10,000 images. These five batches were again divided into two batches of 12.5%, two batches of 7.5%, and two batches of 30% images (see Figure 7). A training run was performed for each batch of 10,000 images. The aim of that test was to validate the presented solution for different image samples used for training, demonstrating the generalization capacity of the proposed model.

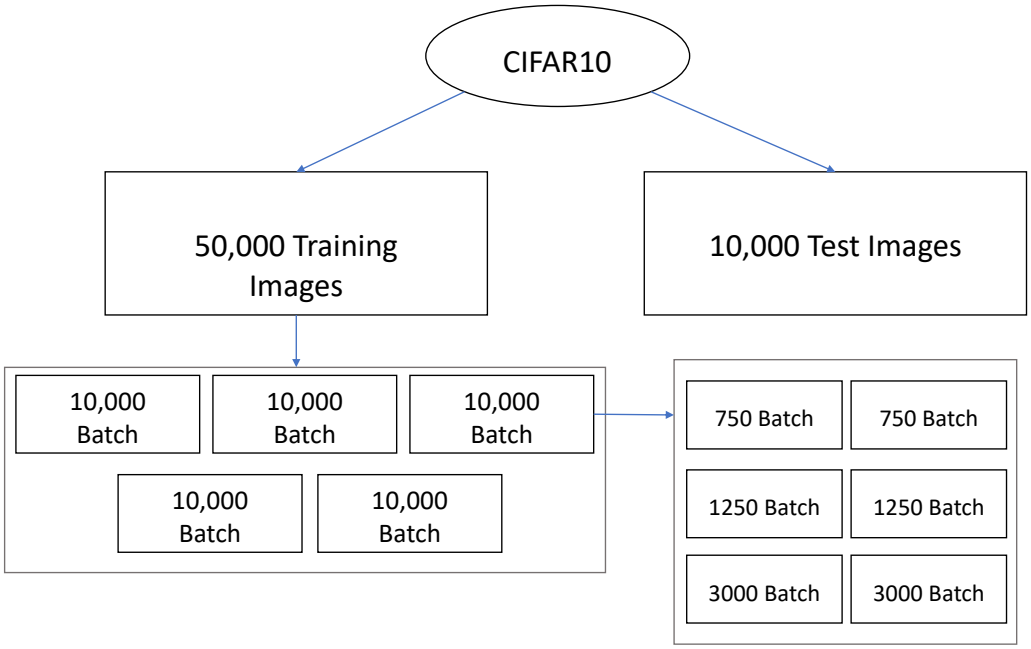

Figure 7. Division of the CIFAR10 dataset for the cross-validation tests.

Table 5 shows the average results obtained for the two aggregation models discussed. The results show that the presented solution was valid for a wide range of cases and not only for the cases presented in this study, since the maximum accuracy obtained after 100 rounds of training in previous tests (91%) and the maximum accuracy obtained after 100 rounds of training with the cross-validation tests (90%) were only slightly different.

Table 5. Results obtained with cross-validation for 8 devices.

Elapsed Rounds	Weighted Average Accuracy	Arithmetic Average Accuracy
10	28%	8%
30	42%	17%
50	74%	42%
100	90%	89%

5.6. Increasing the Complexity of the Dataset

The use of the weighted average aggregation led to an overall improvement in training. However, for CIFAR10, both the weighted average and the arithmetic average achieved a similar maximum accuracy. Therefore, we performed a final test on a larger and more complex dataset—CIFAR 100 [30]—to further evaluate the performance of our FL solution.

This dataset has the same number of training and test images as CIFAR10. The main difference is in the number of image classes. While CIFAR10 considers 10 image classes, CIFAR100 considers 100 image classes.

In Table 6, we present the results obtained by repeating the tests with eight clients with an unbalanced dataset for both the arithmetic average and the weighted average aggregation methods.

Table 6. Results obtained from 8 clients (two clients with 7.5%, two with 12.5%, two with 30%, and two with 0.0002%) with the more complex CIFAR100 dataset.

Elapsed Rounds	Weighted Average Accuracy	Arithmetic Average Accuracy
10	6%	2%
30	21%	12%
50	32%	24%
100	63%	51%

Note that for the same rounds of training with this dataset, only a maximum accuracy of 63% was achieved. This was to be expected since the dataset had a much higher complexity. Sterneck et al. [31] showed that with the CIFAR100 dataset, the **VGG19** CNN achieved a maximum accuracy of 70.3% after 100 training rounds, which was obtained with a balanced dataset and a centralized machine learning method with access to all training data. In our case, since we performed distributed training with unreliable clients (due to the lack of dataset balance), we would need a much larger number of rounds to converge closer to that maximum accuracy.

These results support the use of aggregation methods that benefit those clients who contribute more to training.

6. Lessons Learned

Although the advantages of using FL are obvious, it is also important to note that a solution based on an FL architecture has some potential weaknesses that need to be addressed, such as:

- The need for communication between clients and server also brings the need to deal with asynchronous client connection or disconnection (client dropout), which will have an impact on the system. For these cases, both the server and the FL architecture need to be extremely resilient. If the system is not prepared to handle connection dropouts or a large influx of users during a communication round, the end result may be significantly degraded or the system may suddenly fail.
- The need to protect the privacy of every client also brings with it the need to protect against cyberattacks or malicious clients. Therefore, all communications must be secured, as well as the access to the dataset at each client.
- In a federated solution, there is no prior knowledge of the data that each client contains locally. This means that there may be clients that do little to improve the accuracy of the global model. Moreover, in most cases, the clients are heterogeneous, meaning that not all clients have the same computational power or energy efficiency. Accordingly, some clients may take significantly longer to train locally than others. In these cases, the training time in each round depends on the slowest client training and transmitting its local updates.

However, FL has very important strengths compared to centralized ML. Above all, client data protection is the focus of the concept presented. Moreover, two main advantages arise: (*i*) distributed training, i.e., it is no longer necessary to have a superpowerful machine on the server to run computationally intensive algorithms, and training becomes easier while being performed in parallel; (*ii*) the need for distributed training implies the decentralized use of data. A federated solution has the potential to use a dataset that grows exponentially with the number of clients connected to the system, i.e., the model becomes more complete and robust and responds more accurately to a wider variety of situations.

Based on the analysis performed, this work suggests the inclusion of admission criteria in the implementation of an FL solution. This would allow the introduction of several requirements in the system, for example:

- Ensure that all clients have sufficient computing capacity to make a positive contribution to the training;
- Ensure that all clients have sufficient data to ensure model convergence;
- Ensure that all clients are trustworthy, i.e., do not pose a risk to the privacy of others;
- Ensure that all clients have a stable and secure connection to the server.

We strongly believe that the inclusion of admission criteria should be further explored as it would improve the overall quality of an FL architecture.

We also believe that FL could benefit from other aggregation methods. A clear example of this is reported in [32], where a protocol with user selection was proposed to reduce the negative effect of correlation within the training data of nearby participants, which impaired the optimization of the FL process. In our work, we demonstrated that using a slightly more complex method improved the overall accuracy of the solution. In addition to a weighted averaging of each client's contribution, other approaches could be explored, such as:

- Performing a validation of the local model in each client and giving preference to the clients that achieve a higher accuracy locally;
- Maintaining and updating a historical score on the server side to weight each client's contribution according to training performance over time;
- Sending updates to the server along with the relevant client status (e.g., connection status, local dataset size, etc.) to continuously adjust the aggregation method.

7. Conclusions

This paper presented a multimobile Android-based implementation of a federated learning solution and studied the impact of unreliable participants and selective aggregation in the federated solution towards the deployment of mobile FL solutions in real-world applications.

After establishing a baseline using a traditional ML approach, we showed that we were able to achieve similar performance using an FL solution with two clients and a uniformly distributed dataset. When the dataset was not evenly distributed among participants, a lower accuracy was observed in the early stages of training, and the network typically required a higher number of rounds before performance converged. After the introduction of a weighted average selective aggregation, we found a significant improvement in convergence when using a dataset unevenly distributed among clients. However, the accuracy achieved for four and eight clients after the 100 rounds of training was not different from the previous tests. This was due to the complexity and dimension of the dataset, which led to a training accuracy near the accuracy that was close to the known maximum accuracy when enough training rounds were performed.

Afterwards, we ran cross-validation tests to demonstrate the generalizable nature of our results. Finally, we conducted experiments with a more complex dataset and found that using the weighted average aggregation method not only improved the accuracy across rounds but also improved the maximum accuracy achieved after 100 rounds of training.

By comparing an arithmetic average aggregation method and a weighted average aggregation method, we concluded that there was room for improvement in the aggregation methods used for FL. In the context of the existing challenges of FL, we also concluded that FL would benefit from the introduction of admission criteria for clients.

For future work, it would be important to test the implemented solution for a larger number of clients (i.e., higher than eight) and further analyze the impact of the number of clients in the training process and communications. In addition, our experience has shown that the proper handling of disconnections and newly added clients during training is critical for real-world deployment. Other opportunities to be explored include using heterogeneous clients with different computational capacities, investigating the impact of clients with unfavorable data (e.g., blurred or pixelated images), improving the aggregation method to further optimize model convergence, testing the federated solution with even larger and more complex datasets to validate the applicability of the model to

other use cases, and exploring cybersecurity techniques and secure connections for privacy preservation to enable sustainability for all participants.

Author Contributions: Conceptualization, D.P., P.P. and G.F.; methodology, L.E., D.P., P.P. and G.F.; software, L.E.; validation, L.E.; formal analysis, L.E., D.P., P.P. and G.F.; investigation, L.E.; resources, D.P., P.P. and G.F.; data curation, L.E.; writing—original draft preparation, L.E. and D.P.; writing—review and editing, D.P., P.P. and G.F.; visualization, L.E. and D.P.; supervision, D.P., P.P. and G.F.; project administration, G.F.; funding acquisition, D.P., P.P. and G.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Instituto de Telecomunicações and Fundação para a Ciência e a Tecnologia, Portugal, under grants UIDB/50008/2020 and EXPL/EEI-HAC/1511/2021. This work was also supported by Instituto de Sistemas e Robótica (Project UIDB/00048/2020), funded by the Portuguese Science Agency "Fundação para a Ciência e a Tecnologia" (FCT).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the first author.

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

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial intelligence
CPU	Central processing unit
CNN	Convolutional neural network
FedAvg	Federated averaging
FL	Federated learning
FedSGD	Federated stochastic gradient descent
GM	Geometric median
IP	Internet Protocol
JSON	JavaScript Object Notation
ML	Machine learning
ReLu	Rectified linear unit
RGB	Red green blue
REST	Representational state transfer
RFA	Robust federated averaging
SMC	Secure multiparty computation
SGD	Stochastic gradient descent

References

1. Duranton, M.; De Bosschere, K.; Coppens, B.; Gamrat, C.; Hoberg, T.; Munk, H.; Roderick, C.; Vardanega, T.; Zendra, O. HiPEAC vision 2021: high performance embedded architecture and compilation. *Eur. Netw. High-Perform. Embed. Archit. Compil.* **2021**, *1*, 5–29.
2. Coughlin, T. 175 Zettabytes By 2025. 2018. Available online: https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/?sh=59d7f05f5459 (accessed on 28 February 2023).
3. Data Creation and Replication Will Grow at a Faster Rate than Installed Storage Capacity, According to the IDC Global DataSphere and StorageSphere Forecasts. Available online: https://www.businesswire.com/news/home/20210324005175/en/Data-Creation-and-Replication-Will-Grow-at-a-Faster-Rate-Than-Installed-Storage-Capacity-According-to-the-IDC-Global-DataSphere-and-StorageSphere-Forecasts (accessed on 28 February 2023).
4. Satyanarayanan, M. Mahadev (Satya) Satyanarayanan—Edge Computing: a New Disruptive Force, Keynote at SYSTOR 2020. Available online: https://www.youtube.com/watch?v=7D2ZrMQWt7A (accessed on 28 February 2023).
5. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [CrossRef]
6. Mcmahan, H.B.; Ramage, D. Communication-Efficient Learning of Deep Networks from Decentralized Data. *Artif. Intell. Stat.* **2017**, *54*, 10.

7. Shokri, R.; Shmatikov, V. Privacy-preserving deep learning. In Proceedings of the Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; pp. 1310–1321.
8. Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konečnỳ, J.; Mazzocchi, S.; McMahan, H.B.; et al. Towards federated learning at scale: System design. *arXiv* **2019**, arXiv:1902.01046.
9. Srinivasan, A. Difference between distributed learning versus Federated Learning Algorithms. 2022. Available online: https://www.kdnuggets.com/2021/11/difference-distributed-learning-federated-learning-algorithms.html (accessed on 28 February 2023).
10. Greengard, S. AI on Edge. *Commun. ACM* **2020**, *63*, 18–20. [CrossRef]
11. Yang, Q.; Liu, Y.; Chen, T.; Tong, Y. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* **2019**, *10*, 1–19. [CrossRef]
12. Yang, Q.; Liu, Y.; Cheng, Y.; Kang, Y.; Chen, T.; Yu, H. Federated learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **2019**, *13*, 1–207.
13. Lim, W.Y.B.; Luong, N.C.; Hoang, D.T.; Jiao, Y.; Liang, Y.C.; Yang, Q.; Niyato, D.; Miao, C. Federated Learning in Mobile Edge Networks: A Comprehensive Survey. *IEEE Commun. Surv. Tutorials* **2020**, *22*, 2031–2063. [CrossRef]
14. Courville, I.G.; Bengio, Y.; Aaron. Deep learning by Ian Goodfellow,Yoshua Bengio, Aaron Courville. *Nature* **2016**, *29*, 1–73.
15. Pan, X.; Chen, J.; Monga, R.; Bengio, S.; Jozefowicz, R. Revisiting Distributed Synchronous SGD. *arXiv* **2017**, arXiv:1604.00981.
16. Hard, A.; Rao, K.; Mathews, R.; Ramaswamy, S.; Beaufays, F.; Augenstein, S.; Eichner, H.; Kiddon, C.; Ramage, D. Federated learning for mobile keyboard prediction. *arXiv* **2018**, arXiv:1811.03604.
17. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H.B.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical secure aggregation for privacy-preserving machine learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1175–1191. [CrossRef]
18. Pillutla, K.; Kakade, S.M.; Harchaoui, Z. Robust Aggregation for Federated Learning. *IEEE Trans. Signal Process.* **2022**, *70*, 1142–1154. [CrossRef]
19. van Berkel, C.H.K. Multi-Core for Mobile Phones. In Proceedings of the 2009 Design, Automation & Test in Europe Conference & Exhibition, Nice, France, 20–24 April 2009; pp. 1260–1265.
20. Konečnỳ, J.; McMahan, H.B.; Yu, F.X.; Richtárik, P.; Suresh, A.T.; Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv* **2016**, arXiv:1610.05492.
21. Smith, V.; Chiang, C.K.; Sanjabi, M.; Talwalkar, A.S. Federated Multi-Task Learning. In *Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Dutchess County, NY, USA, 2017; Volume 30.
22. McMahan, H.B.; Ramage, D.; Talwar, K.; Zhang, L. Learning differentially private recurrent language models. In Proceedings of the 6th International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018; pp. 1–14.
23. Simsek, M.; Boukerche, A.; Kantarci, B.; Khan, S. AI-driven autonomous vehicles as COVID-19 assessment centers: A novel crowdsensing-enabled strategy. *Pervasive Mob. Comput.* **2021**, *75*, 101426. [CrossRef]
24. Valerio, L.; Conti, M.; Passarella, A. Energy efficient distributed analytics at the edge of the network for IoT environments. *Pervasive Mob. Comput.* **2018**, *51*, 27–42. [CrossRef]
25. Malekzadeh, M.; Clegg, R.G.; Cavallaro, A.; Haddadi, H. Privacy and utility preserving sensor-data transformations. *Pervasive Mob. Comput.* **2020**, *63*, 101132. [CrossRef]
26. Zhu, C.; Ni, R.; Xu, Z.; Kong, K.; Huang, W.R.; Goldstein, T. GradInit: Learning to Initialize Neural Networks for Stable and Efficient Training. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 16410–16422.
27. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 84–90. [CrossRef]
28. Kaushik, A. Understanding the VGG19 Architecture. 2020. Available online: https://iq.opengenus.org/vgg19-architecture/ (accessed on 28 February 2023).
29. Yadav, H. Preserving Data Privacy in Deep Learning: Part 1. 2020. Available online: https://towardsdatascience.com/preserving-data-privacy-in-deep-learning-part-1-a04894f78029 (accessed on 28 February 2023).
30. Krizhevsky, A.; Hinton, G. Learning multiple layers of features from tiny images. Master's Thesis, Department of Computer Science, University of Toronto, Toronto, ON, Canada, 2009.
31. Sterneck, R.; Moitra, A.; Panda, P. Noise Sensitivity-Based Energy Efficient and Robust Adversary Detection in Neural Networks. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *41*, 1423–1435. [CrossRef]
32. Sun, C.; Wu, S.; Cui, T. User Selection for Federated Learning in a Wireless Environment: A Process to Minimize the Negative Effect of Training Data Correlation and Improve Performance. *IEEE Veh. Technol. Mag.* **2022**, *17*, 26–33. [CrossRef]