1 2 9 0

UNIVERSIDADE Ð COIMBRA

Leonardo Galveias Esteves

Federated Learning for IoT Edge Computing: An Experimental Study

Dissertation within the scope of Integrated Master's in Electrical and Computer Engineering, specialization in Computers, oriented by Prof. Doctor Gabriel Falcão Paiva Fernandes, Prof. Doctor Paulo José Monteiro Peixoto and Doctor David Bina Siassipour Portugal and presented to the Faculty of Science and Technology of University of Coimbra, Department of Electrical and Computer Engineering.

February 2022

FCTUC

UNIVERSITY OF COIMBRA
FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Federated Learning for IoT Edge Computing: An Experimental Study

Leonardo Galveias Esteves

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering

Supervisors:
Prof. Doctor Gabriel Falcão Paiva Fernandes
Prof. Doctor Paulo José Monteiro Peixoto
Doctor David Bina Siassipour Portugal

Jury:
Prof. Doctor Nuno Miguel Mendonça da Silva Gonçalves
Prof. Doctor Cristiano Premebida
Prof. Doctor Gabriel Falcão Paiva Fernandes

Coimbra, February 2022

Agradecimentos

O trabalho apresentado neste documento não seria possível sem a ajuda de todos os que acompanham diariamente.

Em primeiro, gostaria de começar por agradecer aos meus orientadores, professor Gabriel Falcão, professor Paulo Peixoto e professor David Portugal, por todo o apoio, ajuda e orientação. Foram, de facto, uma peça importante no desenvolvimento desta dissertação.

A toda a minha família, que foram o meu maior apoio ao longo destes últimos anos. Agradeço em especial à minha Mãe, ao meu Pai, à minha Irmã e aos meus Avós por tornarem possível o meu percurso neste curso e pelo carinho e força ao longo de toda a minha vida.

A todos os meus amigos de curso, um enorme obrigado por todos os momentos que me proporcionaram nestes últimos anos. Obrigado pela amizade, pela alegria e pelos momentos de entreajuda que me ajudaram a superar todas fases mais difíceis neste percurso.

O meu mais especial agradecimento à Joana, por tudo o que aquilo que representa na minha vida, por todo o apoio, por nunca deixar de me motivar, por nunca me deixar desistir e principalmente por todo o amor.

A todos os que estiveram comigo neste percurso e não foram aqui referidos! Por fim, agradeço também a esta cidade maravilhosa que me acolheu durante estes anos, Coimbra!

A todos, muito obrigado!

Abstract

The data generated annually is around 40 trillion gigabytes. This significant increase in data every year brings with it the need to ensure the protection of sensitive information.

Artificial Intelligence has been improving its results more and more, presenting models capable of responding rigorously in critical areas, for example medicine, autonomous vehicles, robotics, etc. These algorithms need huge amounts of available data to optimize their response to all their area of operation.

The urge to continue to improve these algorithms while maintaining the privacy and confidentiality of the data used emerged. Thus, the concept of Federated Learning was created. Federated Learning allows to continue training Machine Learning algorithms without sharing the data used for model convergence. Federated Learning has some similarities with Distributed Learning. In both concepts the training is distributed, however, Federated Learning also decentralizes the data in order to keep the information private.

The objective of this dissertation is to explore the concept of Federated Learning, as well as to directly compare this concept with centralized Machine Learning. To this end, the architecture required to build a federated solution is analyzed in depth. This dissertation also presents results obtained with federated solutions in both simulation and real-world deployment. Finally, a viewpoint of the obtained results is also presented, and options for optimizing a solution with Federated Learning are discussed.

Keywords

Federated Learning (FL); Convolutional Neural Network (CNN); Edge Computing; Cloud Computing; Federated Averaging (FedAvg); Stochastic Gradient Descent (SGD); FederatedSGD (FedSGD); Secure Aggregation Protocol; Machine Learning (ML); Deep Learning (DL); Internet of Things (IoT); TensorFlow (TF); TensorFlow Federated (TFF); Pytorch; PySyft

Resumo

Os dados gerados por anualmente rondam os 40 trilhões de gigabytes. Este aumento significativo de dados todos os anos trás a necessidade de assegurar a proteção de informação sensível.

A Inteligência Artificial tem vindo a melhorar cada vez mais os seus resultados, apresentando modelos capazes de responder rigorosamente em áreas de atuação críticas, por exemplo, medicina, veículos autónomos, robótica, etc. Estes algoritmos precisam de enormes quantidades de dados disponíveis para otimizarem ao máximo a sua resposta perante todos a sua área de operação.

Surgiu a necessidade de continuar a melhorar estes algoritmos mantendo a privacidade e confidencialidade dos dados utilizados. Desta forma, foi criado o conceito de Federated Learning. O Federated Learning permite continuar a treinar algoritmos de Machine Learning sem partilhar os dados utilizados para a convergência do modelo. O Federated Learning apresenta apresenta algumas similaridades com o Distributed Learning. Em ambos os conceitos o treino é distribuido, no entanto o Federated Learning descentraliza também os dados de forma a manter a informação privada.

O objetivo desta dissertação passa por explorar o conceito de Federated Learning, assim como comparar diretamente este conceito com o Machine Learning centralizado. Para tal, é mostrada a arquitetura necessária para a construção de uma solução federada. Este documento apresenta ainda resultados obtidos com soluções federadas tanto em ambiente de simulação como numa implementação em ambiente real. Finalmente, é também apresentado um ponto de vista dos resultados obtidos e opções de otimização de uma solução com Federated Learning são discutidas.

Palavras Chave

Aprendizagem Federada; Rede Neuronal Convulacional; Computação de Borda; Computação em Nuvem; Federated Averaging (FedAvg); Gradiente Descendente Estocástico; FederatedSGD (FedSGD); Protocolo de Aggregação Segura; Aprendizagem de Máquina; Aprendizagem Profunda; Internet das Coisas; TensorFlow (TF); TensorFlow Federado; Pytorch; PySyft

"Study without desire spoils the memory, and it retains nothing that it takes in."

- Leonardo da Vinci

List of acronyms

AI	Artificial Inteligence
CNN	Convolutional Neural Network
DL	Deep Learning
FedAvg	Federated Averaging
FedSGD	FederatedSGD
FL	Federated Learning
IoT	Internet of Things
ML	Machine Learning
SGD	Stochastic Gradient Descent
TF	TensorFlow
TFF	TensorFlow Federated

Table of contents

List of figures

List of tables

Chapter 1

Introduction

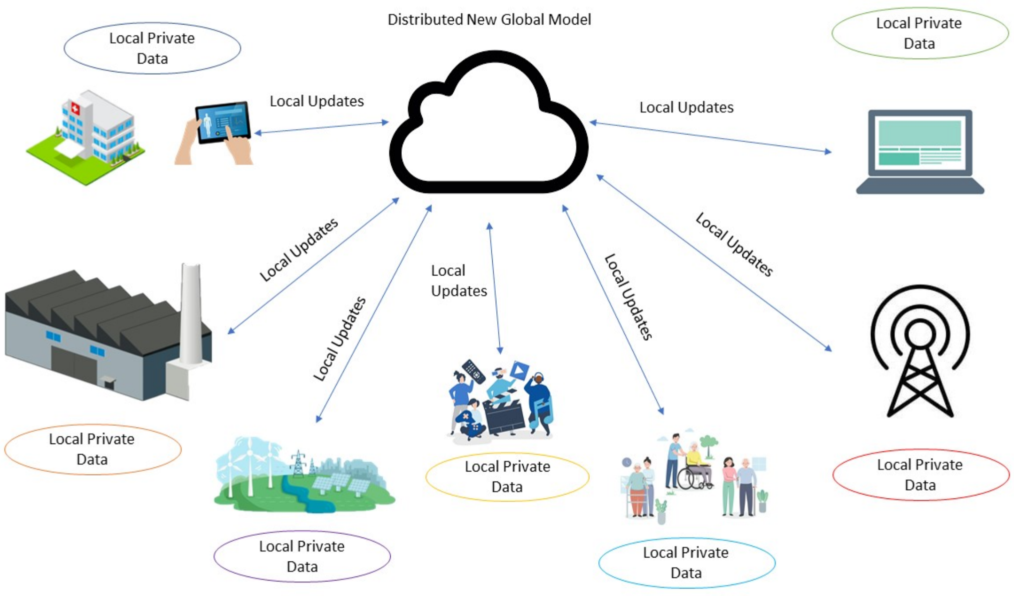

Fig. 1.1 Federated Learning Applications.

Mobile devices, wearable devices and autonomous vehicles are just some of the devices that are part of modern distributed networks that generate an abysmal amount of data every day [1] [2] [3]. Due to the increased computing power, energy efficiency, reduced latency between communications, increased bandwidth, increased data governance solutions and even storage capacity of these devices, along with concerns about private information sharing, it is increasingly attractive to allocate data locally and push computing to the edge so as to not let users' private information be exposed [4].

The concept of edge computing is not new. Indeed, computing simple instructions in low-power, distributed devices is an area of research that is several years old. Recently, some work has considered training machine learning models centrally, but serving and storing them locally [5].

However, as the computing and storage capabilities of the devices comprising a distributed network grow, it is possible to leverage enhanced local resources on each device. This has led to an increase in interest in Federated Learning [6] [7] [8], which exploits statistical training models directly on edge devices. Figure 1.1 shows that Federated Learning has numerous applications, such as healthcare systems, industry, telecommunications, etc.

This concept has some similarities with Distributed Learning. Distributed Learning is about having the data centralized but distributing the training model across different nodes, whereas Federated Learning privileges a decentralization of both the data and the model, while also contributing to a global model that aggregates all the clients' models [9] (Figure 1.2).

Throughout this dissertation the concept of Federated Learning is discussed. This work explores both the definition of the concept itself and the implementation of the whole process, from the server to the clients. It also discusses existing methods of aggregation and the updates of the local model in a global model located on the central server. In addition, some challenges/fragilities of the Federated Learning process are also presented, paving the way for the contributions of this Dissertation work.

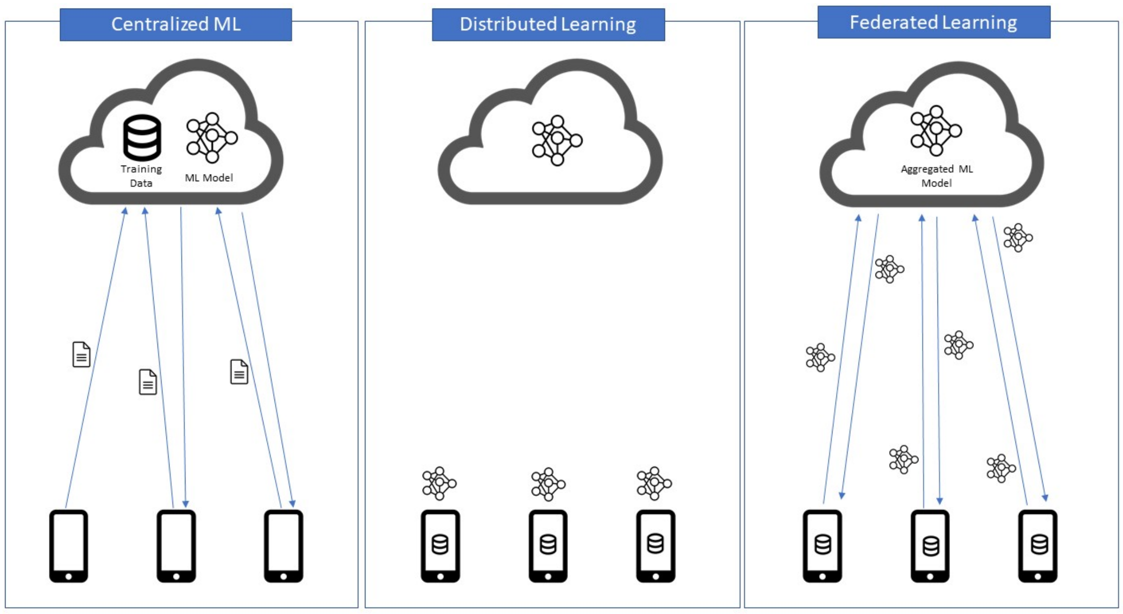

Fig. 1.2 Difference Between Centralized Machine Learning, Distributed Learning and Federated Learning.

In a later phase, the implementation of a federated solution is presented and compared with an existing solution in centralized Machine Learning (Figure 1.2) with the objective of studying the advantages and disadvantages of the presented concept. Finally, a federated solution is also developed in a real environment with the purpose of having a better perception of FL in a real context.

The structure of this dissertation and content of each chapter are the following:

- *Chapter 2* reviews the state of the art on Federated Learning, different methods of aggregation and challenges of FL;
- *Chapter 3* presents the frameworks used throughout the development phase of this dissertation and a step by step walkthrough of an implementation of Federated Learning. In addition, this chapter also presents the testing methodology used in this dissertation;
- In *Chapter 4*, the results obtained are presented and discussed;
- *Chapter 5* draws some conclusions, as well as some proposals for future work;

Chapter 2

Fundamentals and Related Work

2.1 The Definition of Federated Learning

Within the field of Artificial Inteligence (AI), the Federated Learning (FL) concept aims to create a Machine Learning (ML) model based on data distributed through multiple sites [10]. It grants users the ability to collaboratively train a shared model without sharing the personal data located on their devices [11] [8].

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

- Two or more parties are interested in building an ML model collectively. Each party holds some data that wishes to contribute to train the model;
- It is imperative that during the model-training process the data retained by each party never leave that party;
- The model may be transmitted in part from one party to another, under an encryption scheme, in such a way that other parties cannot re-engineer the data at any given party;
- In theory, the performance of the resulting model is a good approximation of an ideal ML model built with all the data located at one party.

More technically, let $\mathscr{N} = \{1,...,N\}$ designate the set of N parties, each of them has a private dataset $D_{i\in\mathscr{N}}$. Each data owner i will use their dataset D_i to train a local model w_i from and forward the local model parameters to the FL server. Subsequently, all the gathered local model parameters are aggregated $\mathbf{w} = \cup_{i\in\mathscr{N}}\mathbf{w}_i$ to produce a global model W_G [13].

A conventional architecture and training process of an FL system is illustrated in Figure 2.1. In this system, the parties collectively train an ML model with the assistance of an aggregate server. There is a presumption that all the parties involved in the process are trustworthy, which implies that they use real private data to perform the training and provide the true local model parameters obtained within the training to the FL server [13]. Naturally, this assumption might not be practical and this topic is discussed further ahead in this document.

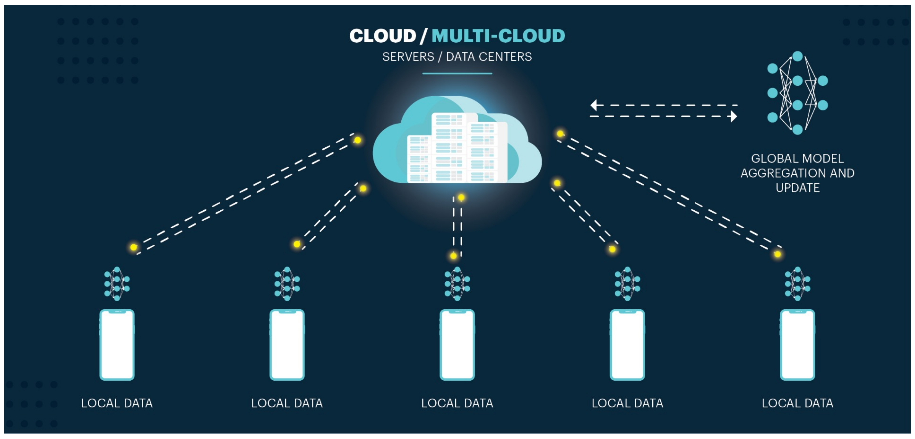

Fig. 2.1 Federated Learning Architecture.

Typically, the FL process includes 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. Furthermore, the server stipulates the hyperparameters of the global model and the training process. In conclusion, 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 their 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 instance,

$$\mathbf{w}_i^{t^*} = arg\min_{\mathbf{w}_i^t} L(\mathbf{w}_i^t). \tag{2.1}$$

Afterwards, 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 the parties selected 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 desirable training accuracy is achieved [13], i.e.,

$$L(\mathbf{w}_G^t) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{w}_i^t). \tag{2.2}$$

2.2 Related and Current Work

The Federated Learning training process can be used for several Machine Learning models that use the Stochastic Gradient Descent (SGD) method. Typically, a training dataset contains a set of n data feature vectors $x = \{x_1, ..., x_n\}$ and a set of corresponding data labels[1] $y = \{y_1, ..., y_n\}$. In addition, $\hat{y_j} = f(x_j; \mathbf{w})$ denotes the predicted result from the model $\mathbf{w}$ updated/trained by data vector x_j [13]. Table 2.1 compiles the most used loss functions of common ML models [14].

Table 2.1 Loss functions of common ML models.

Model	Loss Function $L(\mathbf{w}_i^t)$
Neural Network	$\frac{1}{n}\sum_{h=1}^{n}(y_i - f(\mathbf{x}_j; \mathbf{w}))^2$ (Mean Squared Error)
Linear Regression	$\frac{1}{2}\|\|y_i - \mathbf{w}^T\mathbf{x}_j\|\|^2$
K-means	$\sum_j \|\|\mathbf{x}_j - f(\mathbf{x}_j; \mathbf{w})\|\|$ ($f(\mathbf{x}_j; \mathbf{w})$ is the centroid of all objects assigned to x_j's class)
Squared-SVM (Support Vector Machine)	$[\frac{1}{n}\sum_{j=1}\max(0, 1 - y_j(\mathbf{w}^T\mathbf{w}_j - bias))] + \lambda\|\|\mathbf{w}^T\|\|^2$ (*bias* is the bias parameter and λ is const.)

Most of the recent successful Deep Learning [15] 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 be more amenable to optimization by simple

[1]In the case of unsupervised learning, there is no data label.

gradient-based methods. Therefore, it is natural that the algorithms of federated optimization are built based on SGD.

After the local model training and update, an essential part of the FL process arises, the global model aggregation. There are several proposed solutions to this topic.

In experiments by Chen et al. [16] show that large-batch synchronous state-of-the-art SGD optimization in the data center setting outperforms asynchronous approaches. To apply this approach in the federated setting, 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 (non-stochastic) gradient descent[2]. The baseline algorithm is `FederatedSGD` (`FedSGD`)[15].

A typical implementation of `FedSGD` with $C = 1$ and 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 $\sum_{k=1}^{K} \frac{n_k}{n} g_k = \bigtriangledown f(w_t)$ [17]. 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 takes one step of gradient descent on the current model using its local data, and the server takes a weighted average of the resulting models. Once the algorithm is written this way, more computation 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 [15]. This approach is called `FederatedAveraging` (`FedAvg`).

The amount of computation is controlled by three key parameters: C, the fraction of clients that perform computation on each round; E, the number of training passes each client makes over its local dataset on each round; and B, the local batch size used for the client updates. Selecting $B = \infty$ and $E = 1$ corresponds to `FedSGD`. For a client with n_k local examples, the number of local updates per round is given by $u_k = E\frac{n_k}{B}$ [15]. Algorithm 1 is a complete pseudo-code of the `FederatedAveraging` method.

[2]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 satisfy $\mathbb{E}[g] = \bigtriangledown f(w)$.

Algorithm 1: `FederatedAveraging`**. 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 $t = 1, 2, ...$ 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
```

Some solutions prioritize the privacy and security of all the parties involved. In the FL process, 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 [18]. In [18], the authors propose the Secure Aggregation Protocol to compute these weighted averages. This method would ensure the server becomes aware that one or more parties in this randomly selected subset gave some information relative to the type of data chosen for the training, but not which user.

The FL process faces several challenges. Mobile devices have limited resources in terms of energy and network connectivity. This introduces a certain amount of unpredictability on the number of parties available to participate in each update round and the system must be resilient to it. Because the ML model involved in the FL framework may be parametrized by millions of values, updates may be large, representing a direct cost to users on metered network plans.

Secure Aggregation proposes to operate on high-dimensional vectors, communicate efficiently, even considering a novel set of parties on each round, is robust to users dropping out, and, finally, provides the strongest possible security under the constraints of a server-mediated, unauthenticated network model [18].

Algorithm 2 is a detailed description of the Secure Aggregation protocol.

Algorithm 2: Secure Aggregation Protocol

```
Round 0:
    - Generate DH keypairs $(C_u^{SK}, C_u^{PK})$ and $(S_u^{SK}, S_u^{PK})$
    User u:
    - Send $\underline{\text{signed}}$ public keys $(C_u^{SK}, C_u^{PK}, \underline{\sigma_u})$
    Server:
    - Wait for enough users $\upsilon_1 \subseteq \upsilon$
Round 1:
    Server:
    - Broadcast list of received public keys to all users in $\upsilon_1$
    User u:
    - $\underline{\text{Validate signatures}}$, generate $b_u$ and compute $S_{u,v}$
    - Compute t-ou-of-n secret shares for $b_u$ and $S_u^{SK}$
    - Send encrypted shares of $b_u$ and $S_u^{SK}$
    Server:
    - Wait for enough users $\upsilon_2 \subseteq \upsilon_1$
Round 2:
    Server:
    - Forward received encrypted shares
    User u:
    - Compute masked input $y_u$
    - Send $y_u$
    Server:
    - Wait for enough users $\upsilon_3 \subseteq \upsilon_2$
Round 3:
    Server:
    - $\underline{\text{Send a list of at least } t \text{ survived users: } \upsilon_3 \subseteq \upsilon_2}$
    User u:
    - $\underline{\text{Abort if } |\upsilon_3| < t}$
    - $\underline{\text{Sign } \upsilon_3 \text{ and reply with a signature } \sigma'_u}$
    Server:
    - $\underline{\text{Collect signatures}}$
```

Round 4:

Server:

- Send a list $\{v, \underline{\sigma'_v}\}$ of survived users from $\mathcal{U}_4 \subseteq \mathcal{U}_3$

User u:

- Abort if $|\mathcal{U}_4| < t$, <u>validate signatures</u>
- Send shares of b_u for alive users and S_u^{PK} for dropped

Server:

- Reconstruct secrets
- Compute $\bar{x}$ (the final aggregated value)

The underlined parts are required to guarantee security in the active-adversary model.

Algorithm 2 relies on Shamir's t-out-of-n Secret Sharing [19], which allows a user to split a secret s into n shares, such that any t shares can be used to reconstruct s, but any set of at most $t-1$ shares gives no information about s. Also, it uses a Key Agreement which consists of a tuple of algorithms ($\mathbf{KA.param}, \mathbf{KA.gen}, \mathbf{KA.agree}$). $\mathbf{KA.param}(k) \rightarrow pp$ produces some public parameters. $\mathbf{KA.gen}(pp) \rightarrow (S_u^{SK}, S_u^{PK})$ allows any party u to generate a private-public key pair. $\mathbf{KA.agree}(S_u^{SK}, S_u^{PK}) \rightarrow S_{u,v}$ allows any user u to combine their private key S_u^{SK} with the public key S_u^{PK} for any v (generated using the same pp), to obtain a private shared key $S_{u,v}$ between u and v [18].

2.3 Statement of Contributions

Like all methods/algorithms, FL has its unique characteristics and issues. These challenges make the federated framework distinct from other classical problems, such as distributed learning or traditional private data analysis [5]. This dissertation describes four of the core challenges associated with this process, and clarifies the contributions of the work proposed.

2.3.1 Expensive Communication

Client/Client and Client/Server communications are a critical bottleneck in federated solutions [20]. When paired with privacy concerns over sending raw data, it becomes necessary that data generated on each party remains local. Undeniably, FL networks potentially comprise a massive number of devices, e.g., millions of smartphones, and communication in the network can be slower than local computation by many orders of magnitude, due to limited resources such as bandwidth, energy, and power [21] [22].

To fit a model to data generated by the 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, as opposed to sending the entire data set over the network. To reduce communication in such a setting, two key aspects to consider are: reducing the total number of communication rounds and reducing the size of the transmitted messages at each round [5] [22].

2.3.2 Systems Heterogeneity

The storage, computational, and communication capabilities of each device in federated networks may differ due to variability in hardware (CPU and memory), network connectivity (3G, 4G, 5G, and Wi-Fi), and power (battery level) [21]. Additionally, the network size and systems-related constraints on each device typically result in only a small fraction of the devices being active at once, e.g., hundreds of active devices in a network with millions of devices [20]. Also, it is not uncommon for an active device to drop out at a given iteration due to connectivity or energy constraints [20].

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].

2.3.3 Statistical Heterogeneity

Devices frequently generate and collect data in a highly non-identically distributed manner across the network, e.g., smartphone users have varied use of language in the context of a next-word prediction task [5]. Moreover, the number of data points across devices may vary significantly, and there may be an underlying statistical structure present that captures the relationship among devices and their associated distributions [23]. This data-generation paradigm violates frequently used independent and identically distribute (i.i.d.) assumptions in distributed optimization and may add complexity in terms of problem modeling, theoretical analysis, and the empirical evaluation of solutions [5].

Although the canonical FL problem aims to learn a single global model, there exist other alternatives such as simultaneously learning distinct local models via multitask learning frameworks [23]. In that regard, there is also a close connection between leading approaches for FL. This multitask perspective enables personalized or device specific modeling, which is often a more natural approach to handle the statistical heterogeneity of the data for better personalization [5] [8].

2.3.4 Privacy Concerns

Privacy is often a major concern in FL applications. FL makes a step toward protecting data generated on each device by sharing model updates, e.g., gradient information, instead of raw data. However, communicating model updates throughout the training process can reveal sensitive information, either to a third party or to the central server [24].

Although recent methods aim to enhance 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 [24], [18]. Understanding these tradeoffs, both theoretically and empirically, is a considerable challenge in realizing private FL systems [5].

Theoretically, when an implementation of the FL process in a real environment is performed, new variables are introduced to the equation. These variables are the challenges addressed during this chapter.

Given this information, it is permissible to propose the implementation of a solution in a real environment in order to obtain more accurate results from models trained using the FL process.

Furthermore, during this research an optimization of the model presented in section 2.2 was devised. This model performs the update of the global model from an arithmetic average of the local models coming from each worker. During this dissertation it is proposed to implement a weighted average of the local models in order to update the global model, i.e., to give more importance to the parameters coming from a worker that is contributing more to the model training than a worker where the results obtained are not being positive.

Chapter 3

Design and Development of a Federated Learning System

3.1 Methodology

Having in mind the contributions of this dissertation, the undermentioned methodology is followed:

- Become familiar with the frameworks and programming languages needed to develop solutions using Federated Learning;
- Perform training with a conventional ML algorithm to obtain a baseline for comparison with the studied FL approaches under the same conditions (i.e., the same dataset, the same parameters and on the same machine);
- Perform side by side comparisons of simulations with FL and with conventional ML, and understand the advantages and disadvantages of each method;
- Perform a real implementation of a solution with FL to assess which variables differ between the simulations and the real world solution (for example, in a simulation the communication overhead of the devices is not considered and in a real solution this parameter is expected to be critical);
- Further explore the method and optimize it for better and more accurate results.

Naturally, the dissertation work starts by getting familiar with the frameworks and the FL process itself, to understand where the main bottlenecks currently lie and where contributions are most needed.

Once the first step is completed, it is crucial to chose a baseline that will serve as a comparison against the results obtained with the FL approach used. This baseline consists of a training model under the same parameters, but using conventional ML. This performance comparison study will allow easier and accurate understanding of the advantages and disadvantages of FL when compared with existing solutions.

The third point consists in performing adequate experimental design to allow a comparative analysis between the two types of solutions under different configurations.

After consolidating the knowledge of the FL process and its results, as wells as its constraints, a real world solution will be implemented in a controlled environment, to observe whether there are significant gaps to the simulation results. This point will introduce new variables, such as communication restrictions between devices and energy efficiency of each device. In the application in question, high priority will be given to the parameters with the highest impact and the parameters considered to be less important will be fixed.

3.2 Frameworks for FL

During the initial phase of this work, a study of the available tools for training and testing FL models has been carried out. Following this, two frameworks were chosen for tests and simulations:

- *TensorFlow Federated (TFF)*: TFF [25] is a TensorFlow[1] based framework developed by Google for decentralized Machine Learning and other distributed computations. TFF consists of two layers: *i*) FL and; *ii*) Federated Core (FC). The FL layer is a high-level interface, which enables the implementation of FL into existing TensorFlow (TF) models without the user having to apply FL algorithms personally. The FC layer combines TF with communication operators that allows users to experiment with newly designed and customized FL algorithms [13].
- *PySyft*: PySyft [26] is a framework based on PyTorch[2] for performing encrypted, privacy-perserving DL and implementations of related techniques, such as Secure Multiparty Computation (SMPC), in untrusted environments while protecting data. PySyft is developed such that it retains the native Torch interface, i.e., the ways to execute all tensor operations remain unchanged from that of PyTorch. When a SyftTensor is created, a LocalTensor is automatically created to also apply the input

[1]https://www.tensorflow.org/
[2]https://pytorch.org/

command to the native PyTorch tensor. To simulate FL, participants are created as *Virtual Workers*. Data, i.e., in the structure of tensors, can be split and distributed to the Virtual Workers as a simulation of a practical FL setting. Then, a PointTensor is created to specify the data owner and storage location. In addition, model updates can be fetched from the Virtual Workers for global aggregation [13].

The frameworks presented above were chosen to explore during this dissertation, since they already present a great development and optimization when it comes to FL simulations. Moreover, these two frameworks present a very complete documentation, as well as several examples in order to explain in detail how to implement a federated solution.

3.3 Deployment of a Federated Solution: A step by step Walkthrough

In this section, a concrete example of a simulation with FL is presented to serve as a guide to deploy a federated solution. Throughout the process, the entire experimental design needed to build the solution is presented.

This first test was done with the intention of using the first TensorFlow model in TensorFlowFederated. For this purpose, a classic image classification algorithm has been tested.

3.3.1 Dataset Selection

In order to facilitate the first experiment, a dataset made available in the TFF repository was used. The dataset used consists of a version of MNIST[3] which presents a group of images of handwritten digits, adequate for people who want to experiment with machine learning techniques and pattern recognition methods on real-world data, while spending minimal efforts on preprocessing and formatting.

Federated data is typically non-independent or non-identical (non-i.i.d.). Users commonly have different distributions of data depending on usage patterns. Some clients may have fewer training examples on their devices, suffering from data paucity locally, while some clients will have training examples with higher variability.

In this simulation, the concept of data heterogeneity typical of a federated system is explored. It is important to note that this deep analysis of a client's data is only possible

[3]http://yann.lecun.com/exdb/mnist/

because this is a simulation environment where all the data is available locally. In a real-world federated solution it is not possible to inspect a single client's data.

After understanding the type of dataset to be used, it is shaped into a federated dataset. One of the ways to feed data to a TFF in a simulation is simply a *Python* list, with each element of the list holding the data of an individual user. Figure 3.1 represents an example of the dataset present in a client.

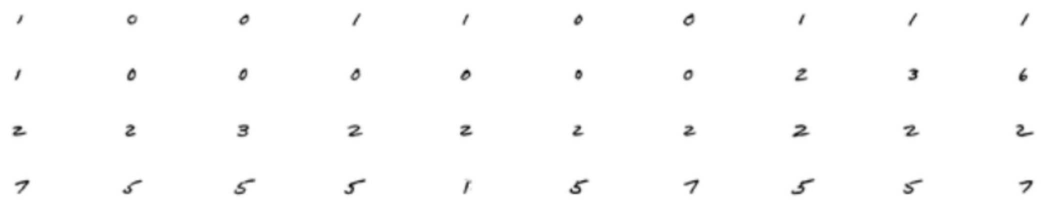

Fig. 3.1 Portion of the MNIST Dataset.

3.3.2 Client Selection

In a typical FL scenario, there is potentially a very large population of user devices. Only a fraction of which may be available for training at a given point in time. An example of this includes clients that are mobile phones, which participate in training only when plugged into a power source.

Naturally, this is a simulation environment, and all the data is locally available. Generally, when running FL simulations, a random subset of clients should be sampled to be involved in each round of training, generally different in each round.

Additionally, achieving convergence in a system with randomly selected subsets of clients in each round can take a long time. Therefore, during the work for this dissertation, the variable introduced in the equation by dropping and adding users in the middle of a training round or in between rounds will not be considered.

For that reason, the number of clients used per round during the entire training is set in prior to the training. In addition, the number of clients is a variable in the code that composes the solution, therefore, it is completely at the discretion of who is implementing the solution. For this example, 10 is the number of clients selected.

3.3.3 Model Initialization

Typically, after establishing the dataset and customer selection, it is necessary to initialize the model that will be trained. This model is initialized on the server, from where it will be sent to each available client. Subsequently, training is performed on each client using the data present there. After the training is completed, each client sends the weights obtained from this process to the server. In the server, an aggregation of the received information is done and, consequently, an update is triggered. This process is called a training round. In each training round the whole process is repeated.

In our case study with MNIST, a *Keras* model is used. *Keras* is a neural network Application Programming Interface (API) for *Python* that is tightly integrated with TF, which is used to build machine learning models. *Keras*' models offer a simple, user-friendly way to define a neural network, which will then be built by TF.

In order to use any model with TFF, it needs to be wrapped in an instance of the **tff.learning.Model** interface, which exposes methods to stamp the model's forward pass, metadata properties, etc., similarly to *Keras*, but also introduces additional elements, such as ways to control the process of computing federated metrics.

3.3.4 Model Training

Now that the model has been initialized, all the conditions are in place to perform the final part of the process, the training of the model. In this case, there are two optimizers: a client optimizer and a server optimizer. The client optimizer is used to compute the local model updates on each client. The server optimizer applies the average to the global model at the server. It is therefore recommended to start with a regular SGD. The learning rate is completely up to the implementer.

Algorithm 3: Python code to run an iterative process

```
iterative_process = tff.learning.build_federated_averaging_process(model,
client_optimizer_fn = lambda: tf.keras.optimizers.SGD(learning_rate=X),
server_optimizer_fn = lambda: tf.keras.optimizers.SGD(learning_rate=Y))
```

After testing different values for the learning rate we found that a value of 0.02 for the client and a value of 1.0 for the server allowed to obtain a better convergence of the model.

The python function presented in 3 constructs a pair of federated computations and packs them into a **tff.templates.IterativeProcess**. Federated computations are programs in TFF's internal language that can express various federated algorithms. In this case, the two computations generated and packed into **iterative_process** implement Federated Averaging.

It is a goal of TFF to define computations in a way that they could be executed in real FL solutions, but currently only local execution simulation runtime is implemented. This default interpreted environment is not designed for high performance, but TF expects to provide higher-performance simulation runtimes to facilitate larger-scale research in future releases.

Finally it is possible to conduct a training round. Algorithm 4 shows how it is possible to conduct a federated training round in the TFF.

Algorithm 4: TFF Training Round

```
state, metrics = iterative_process.next(state, federate_train_data)
```

Note that **state** represents the server state and **federated_train_data** represents the data already distributed to each client.

Table 3.1 shows the results of a complete training round process.

Table 3.1 First Training Round with a portion of the MNIST dataset

.

Round	Number of Clients	Training Accuracy	Training Loss	Total Number of Images
1	10	0.1235	3.1194	4860

Training accuracy and loss are obtained from a validation of the model with a portion of the MNIST images used only for testing.

Finally, to repeat a training round as many times as necessary, you should include algorithm 5 within a for loop that runs through all the training rounds.

Algorithm 5: Training Cicle

```
for round_num in range(X, NUM_ROUNDS):
state, metrics = iterative_process.next(state, federated_train_data)
```

In conclusion, in table 3.2 you can see the result of the example with the MNIST dataset presented in this chapter. It should be noted that this solution is very similar to one made with conventional machine learning, with the big advantage of having decentralized data.

Table 3.2 Ten rounds of Federated Learning Training with the MNIST dataset.

Round	Training Accuracy	Training Loss
2	0.1352	2.9835
3	0.1438	2.8617
4	0.1741	2.7957
5	0.1992	2.6147
6	0.2198	2.5298
7	0.2409	2.4054
8	0.2611	2.3154
9	0.3082	2.1240
10	0.3331	2.1164

3.4 Tools and Apparatus for this Dissertation

During this dissertation, a machine[4] was assembled in order to help in the simulations performed. Due to scheduling conflicts with students from other projects, it was not possible to perform all simulations on this machine. Therefore, the results presented throughout this paper were done on Godzilla or on the student's personal computer.

Godzilla specifications are:

- **RAM**: 256 GB;
- **GPU**: Zotac Gaming GeForce RTX 3090 24GB GDDR6X Trinity;
- **CPU**: AMD Ryzen Threadripper 3970X;
- **Storage**: 6TB of SSD storage;
- **Motherboard**: Gigabyte TRX40 Aorus Master;
- **PSU**: Corsair AX1600i 1600W 80 Plus Titanium Modular.

[4]This machine was named Godzilla.

One should note that the assembly of this machine was carried out during the preparation of this dissertation.

3.5 Testing Methodology

The final objective for this dissertation is to implement a real-world Federated Learning Solution. To reach that goal we outlined a succession of tests to perform. The results of this tests are presented and discussed in Chapter 4.

In the first instance, the dataset to utilize throughout this roadmap was chosen.

One of the main goals of this dissertation is to compare the behavior of Federated Learning and centralized Machine Learning solutions. Thus, we establish a baseline with centralized Machine Learning to serve as a comparison with the tests performed from this point forward.

The first simulated test allows us to analyze the behavior of a federated solution for a different number of clients selected to perform training each round. That said, the CIFAR10 dataset is distributed through 20 clients and 5 of them are selected to perform training each round. Then, we will test the scenario where all the training is performed in all the 20 available clients.

In the next step tests with 2, 4, and 8 clients will be performed. After implementing the real-world solution these tests will be repeated and then compared. This scenario will let us explore if there is a difference between the output of simulated tests and the ones performed in a real-world solution.

In a real-world implementation, the clients do not have the same amount of data available for training. Therefore, we will distribute the CIFAR10 dataset unevenly through each client to help get a clearer view of how a federated solution behaves on this occasion.

Up to this point, all the tests performed used an arithmetic average aggregation method. That said, we will test this solution with an aggregation method that performs a weighted average. The weighted average benefits clients with a larger dataset over ones that have less data locally. Therefore, the tests performed before will be repeated and compared. Figure 3.2 shows a diagram with the methodology of the tests presented in Chapter 4.

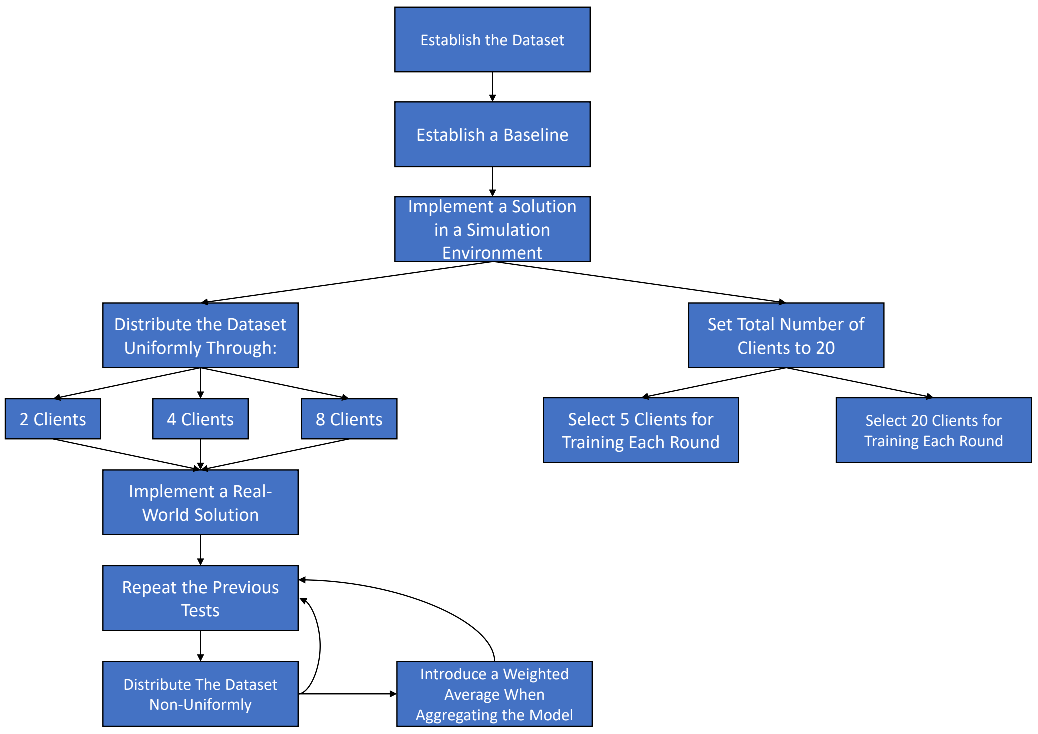
Establish the Dataset
Establish a Baseline
Implement a Solution in a Simulation Environment
Distribute the Dataset Uniformly Through:
Set Total Number of Clients to 20
2 Clients
4 Clients
8 Clients
Select 5 Clients for Training Each Round
Select 20 Clients for Training Each Round
Implement a Real-World Solution
Repeat the Previous Tests
Distribute The Dataset Non-Uniformly
Introduce a Weighted Average When Aggregating the Model

Fig. 3.2 Methodology of the tests presented in Chapter 4.

Finally, a test with a different dataset will be performed. This test aims to validate the implemented solution and the aggregation methods presented with a more complex dataset.

The illustrative example presented in this section gives an insight of how to implement a federated solution in a simulation environment. The next chapter presents a solution implemented in the scope of this dissertation in order to explore the behavior of federated learning and compare the results obtained with centralized machine learning. One should note that Chapter 4 follows the testing methodology presented above.

Chapter 4

Results and Discussion

From the previous chapter, it was possible to have an insight of the path taken to develop this dissertation. With a clear idea of the behavior of a federated solution it becomes easier to choose which points to address and which points to overlook.

The preliminary research done also made it possible to realize the limitations of simulation frameworks. Even though the frameworks presented provide a very clear view of the FL's behavior, they do not offer an exact view of the behavior of a real solution, for example, the bottleneck introduced in the system by server/client and client/server communication.

The example presented in the previous chapter makes it possible to achieve high percentages of accuracy with the MNIST dataset in a few rounds of training, i.e., in only 50 rounds of training an accuracy of 92% was achieved. Therefore, the first decision made was to change the dataset to a more complex one, which would give a clearer idea of the potential of this solution. This leads to the first section of this chapter, the dataset used during this work.

4.1 Dataset

After a thorough study of the existing datasets, CIFAR10 was chosen because it presents a balance between complexity and size. This balance allows the tests performed to see more precisely the difference between the accuracy curve per round of a solution with centralized ML and a solution with FL, while also ensuring that the tests performed do not take a prohibitive amount of time to perform. This last point is crucial, as it allows an appropriate number of scenarios to be tested.

The CIFAR10 dataset consists of 60,000 colour images of 32x32 pixels, divided by 10 classes. There are 50,000 images available for training and 10,000 images available for testing [27].

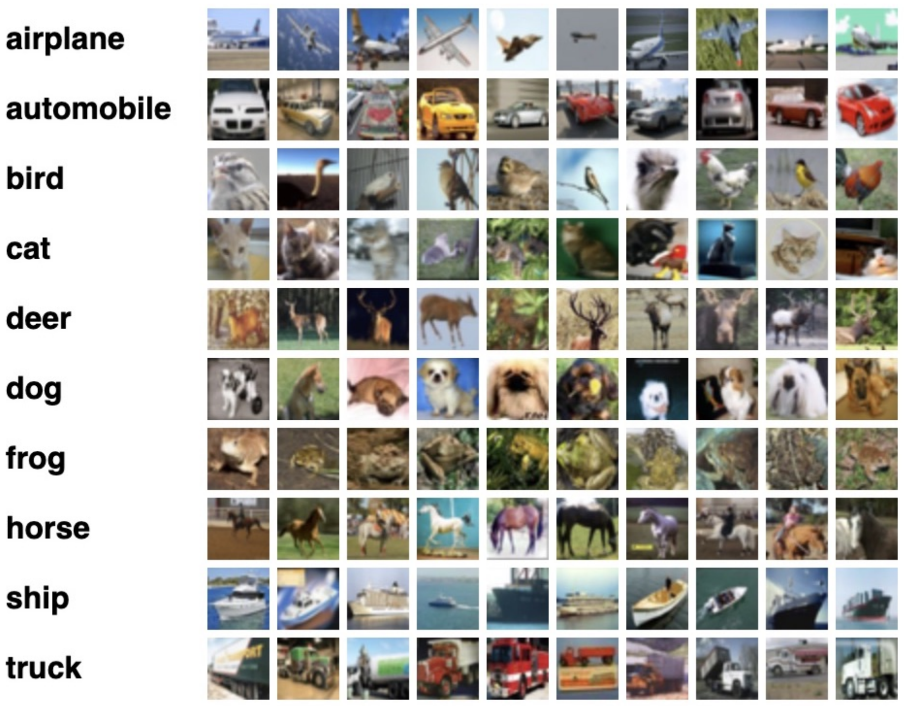

Fig. 4.1 CIFAR10 dataset classes, as well as 10 random images from each.

4.2 Establishing a Baseline

Since the main focus of this dissertation is the direct comparison between centralized Machine Learning and Federated Learning algorithms, a baseline should be made. This baseline will serve as method of comparison throughout this chapter, as well as serving as the basis for the discussion on the advantages and disadvantages of FL versus centralized ML.

In order to make accurate comparisons between the models, the baseline was performed under the same conditions as the simulations with FL, i.e., the same dataset, the same neural network, the same training rounds, etc. were used.

Having chosen the CIFAR10 dataset, the next decision to be made is the neural network used. After some research on both centralized ML solutions and FL simulations it was decided to choose the **VGG19** neural network. This neural network has 16 convolution layers, 3 Fully Connected layers, 5 MaxPool layers, and 1 SoftMax layer.

This network allows reaching a percentage of 94.71% on CIFAR10 according to [28]. However, since the FL simulations are performed on both student's personal machine and Godzilla, it was decided to perform a test with the baseline on each of these machines as well.

Based on the solution provided in [28], a test was run on each available machine. As the goal is to make a direct comparison between centralized ML and FL, it was decided to perform all the tests only up to 100 training rounds and not until the maximum possible accuracy was reached. Assumptions about high training time also made a contribution to this decision.

Finally, after performing these tests, an accuracy of 92.86% was obtained on the personal machine and 92.89% on Godzilla. That said, we have assumed 93% as the value of accuracy obtained locally. Note that throughout this chapter the time it takes to perform 100 rounds of training for any test is taken into account as a relative value, i.e., it is not used as a reference between solutions.

This decision is due to limitations in parallelizing the training of the clients in the simulation frameworks. That is, in an ideal real case, training on each client device should be performed in parallel and each round takes as long as the time it took the slowest device to perform the training. However, the frameworks presented have some vulnerabilities and therefore training is not performed completely in parallel, and all the computation is done on the same machine, which also slows down the time of each communication round.

4.3 Simulation

Now that the dataset and algorithm used and the comparative baseline have been established, the conditions are set to start developing a federated solution in a simulated environment.

4.3.1 Simulation Solution Development

After grasping the concept of Federated Learning and studied its frameworks, we move on to the implementation phase of a solution built from scratch and trained on the CIFAR10 dataset. Since this is a simulation, this solution does not go into detail about how the server-client communication is done. In addition, all clients are present on the same machine. In order to help implementing the solution some existing implementations in PySyft were addresed, so the solution presented in this section uses the same framework.

4.3.1.1 Hyper-Parameters

In this solution there are 5 controllable variables that allow testing various training scenarios.

The first hyper-parameter is the number of clients (**num_clients**). This variable represents the total number of clients participating in the federated solution. The dataset will be divided equally among these clients, thus having each one with the same number of images.

The second variable represents the number of clients selected per round (**num_selected**). These clients are the ones that will actually contribute to the training and they are chosen randomly from the total number of clients, for example, if **num_clients** is 100 and **num_selected** is 20 it means that in each training round 20 clients will be chosen randomly from the total 100.

The third parameter represents the number of rounds that the training will take (**num_rounds**). In each communication round, **num_clients** are randomly selected, training on client's devices takes place, which is followed by aggregation of the individual model weights into one global model. Note that for all the simulations the total number of rounds is 100.

The fourth hyper-parameter represents the total number of local training rounds on each selected client's device (**epochs**). This variable was set to the value 5 for all simulations.

The fifth and last parameter is a the batch size (**batch_size**). Data is loaded on a per batch basis, so it is important to choose a value for this parameter that allows not too little data to be loaded at a time, as this worsens training performance since the full capacity of the client devices is not being taken advantage of, but it is also important not to choose a value that is too high because it can cause a data overload on the clients. During all simulations this variable has the value 32.

In Algorithm 6 it is possible to see all the hyper-parameters as well as which ones will be variables during training. The values of the **epochs** and **batch_size** parameters are due to the fact that they introduce more changes in training, so in order to make all simulations under the same conditions this decision was made. The value of the **num_rounds** parameter was set because after some tests it was possible to observe that after 100 rounds it is already possible to obtain an accuracy close enough to the maximum achieved in article [28] with CIFAR10.

Algorithm 6: Hyper-parameters for this dissertation

```
num_clients = 20
num_selected = The values 5 and 20 were used for this solution
num_rounds = 100
epochs = 5
batch_size = 32
```

4.3.1.2 Model Architecture

In this implementation phase it is necessary to initialize the chosen neural network. As mentioned before, the chosen neural network is **VGG19**. This network can be seen as a successor to **AlexNet**, but it was created by a different group called the Visual Geometry Group at Oxford University. It carries and uses some ideas from its predecessors and improves on them by using deep Convolutional Neural Layers in order to achieve a higher accuracy [29].

The neural network chosen was given a fixed size of 224 RGB channels as input, leading to a 3D matrix of size 224x224x. The only preprocessing done was a subtraction of the mean RGB value from each pixel, computed over the whole training set. It uses a 3x3x kernel with a stride size of 1 pixel, this allows to cover the whole notion of the image. Spatial padding was used to preserve the spatial resolution of the image. A max pooling was performed over a 2x2x pixel windows with stride 2 (Figure 4.2). This was followed by Rectified Linear Unit (**ReLu**) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used **tanh** or **sigmoid** functions this proved much better than those. Finally, the team implemented three fully connected layers from which first two were of size 4096 and after that a layer with 1000 channels for 1000-way ImageNet Large Scale Visual Recognition Challenge (**ILSVRC**) classification and the final layer is a softmax function.

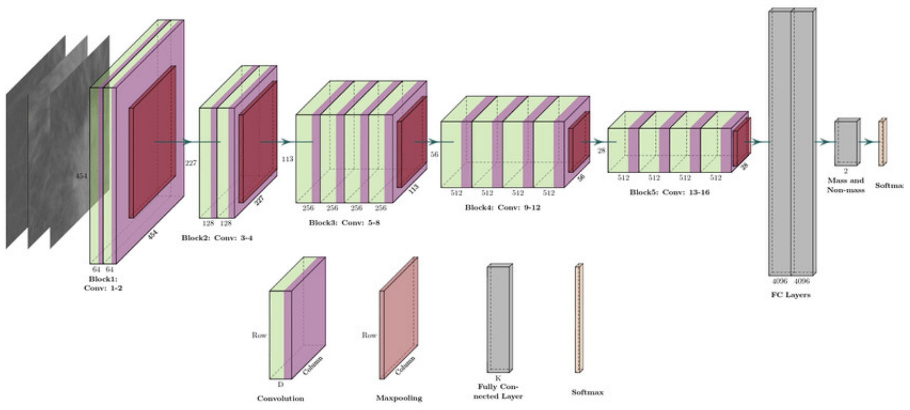

Fig. 4.2 VGG19 Neural Network Architecture.

This network is widely used for image classification for various datasets, e.g. for facial recognition applications. Its weights are easily available in other frameworks like **Keras**, so they can be tinkered with and used for as one intends.

Finally, the matrix shown in 4.1 presents the initial configuration used to create VGG19 neural network for the simulations implemented.

$$[64, 64, M, 128, 128, M, 256, 256, 256, 256, M, 512, 512, 512, 512, M, 512, 512, 512, 512, M] \tag{4.1}$$

4.3.1.3 Functions for Federated Training

In order to accomplish the federated training as desired three auxiliary functions were created. These functions help the implementation in 3 important phases of FL: training on the clients, aggregating the global model on the server and testing the global model to see the improvement of accuracy over the rounds.

A function to train the local model with the client's private data on the selected clients, which we call **client_update**. This local training takes place on **num_selected** clients.

A function to aggregate the weights coming from each client and updates the global model with the updated weights called **server_aggregate**. During the simulations performed, this aggregation was done with a simple arithmetic average and aggregation into the global weights.

Finally, a function that uses the global model and the test loader as input and returns the test loss and accuracy, which we call **test**.

4.3.1.4 Training the FL Model

At this stage, one global model, along with the individual client models is initialized with **VGG19** on a GPU. In this solution, SGD is used as an optimizer for all client models.

This operation takes place within a cycle, which performs the training on the clients for each communication round. Initially, **num_selected** clients are chosen from the available **num_clients**. Then training is performed for each selected client using the **client_update** function. Now, the aggregation of the weights takes by place using the **server_aggregate** function mentioned above. This updates the global model, which is the final model that is used for prediction. After updating the global model, this global model is used to test the training with the help of the **test** function previously mentioned.

This process continues for **num_rounds**, i.e., 100 communication rounds for all the simulations performed.

Algorithm 7: Pseudo-Code for FL training

```
for i in num_rounds do
    Select random num_selected clients from num_clients
    for j in num_selected do
        Perform client_update function
    Use server_aggregate function
    Use test function on the updated model
```

4.3.2 Simulation Results

Now that the we have implemented a solution, the next phase is to perform tests in order to pursue one of the goals of this dissertation, i.e., the comparison between Federated Learning and centralized Machine Learning. Furthermore, these results also serve to understand the limitations of simulation frameworks and their approximation to a solution in a real environment.

In a first phase, the goal of the tests performed was to understand the advantage of bringing more participants into the training program and whether it is possible to obtain satisfactory results even with few customers present per training round. For this, the dataset was divided by 20 clients (**num_clients**) and two cases were selected: 5 clients selected per training round and 20 clients selected per training round (**num_selected**). Evidently, we expect that the second case achieves a higher accuracy, since the dataset used for training is considerably larger in the second case. However, as already discussed, it is important to understand whether a solution with FL, even with few participants, might be feasible.

One should note that in a simulation environment the server/client and client/server communication is not taken into account and in a real case it is an important variable point in the equation, since it can substantially increase the time of a communication round. Therefore, the results presented below have as a major goal to explore the maximum accuracy difference achieved, always keeping in mind that FL always brings a very important point that centralized ML cannot provide: data privacy and decentralization.

In Table 4.1 it is possible to observe the differences in the results obtained in the two cases discussed above.

Table 4.1 Results obtained from two scenarios: 5 (case 1) and 20 (case 2) clients selected per round from a total of 20 clients.

Number of Elapsed Rounds	Total Number of Clients	Number of Clients Selected	Average Accuracy
10	20	5	65%
		20	80%
30	20	5	82%
		20	87%
50	20	5	86%
		20	90%
100	20	5	90%
		20	91%

In these two scenarios it is possible to observe that case 2 presents a faster convergence in the first training rounds, however, after a little more than half of the training rounds, case 1 comes very close to the second case. The model is not as responsive in classifying images for case 1 as in case 2 because it has a considerably smaller dataset than the second case. However, it is also a known fact that the more data available for training, the better prepared the final model will be.

Even though we admitted that the elapsed time for each case presented would not be taken into account because of limitations of the framework in parallelizing client training, it is important to inform that for case 1 it took an average of 1h40min of training, which gives about 1 minute per round, and case 2 took an average of 5h50min minutes to perform, i.e. about 3min30sec per round. One should note that the simulations presented were performed in Godzilla.

After drawing conclusions about the tests presented above, the next testing scenario was decided upon. This second scenario aims to analyze the impact of splitting the data by more clients in each training round. That said, CIFAR10 was divided equally by 2, 4 and 8 clients. For all cases all available clients were selected for training (**num_selected = num_clients**).

One should note that the second case from the previous shown results is also considered here, since the dataset is also equally distributed over all the clients selected in each training round.

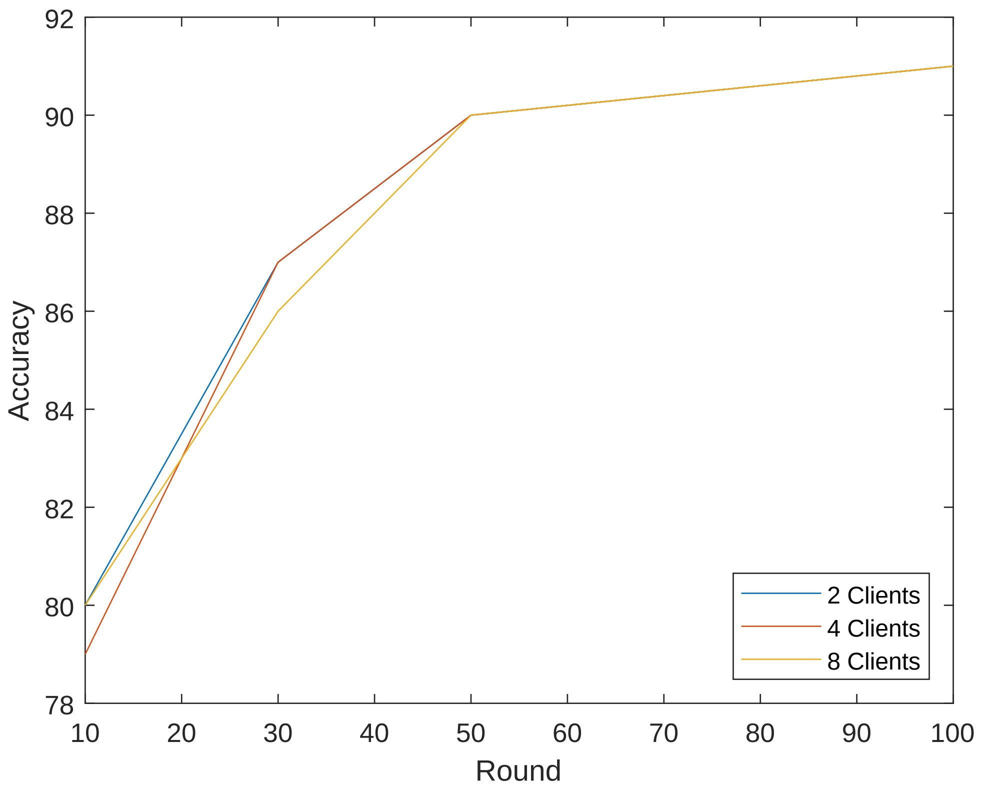

Fig. 4.3 Graphic with the curves of the accuracy of the tests with 2,4 and 8 Clients.

Figure 4.3 shows that adding more clients to the program, but keeping the same number of images for training does not affect the accuracy at any stage of the training, that is, distributing the same dataset over more clients does not worsen the quality of the result. However, it is known that the environment where these tests were performed is a simulation environment and, as mentioned before, in this environment the server/client and client/server communication is not taken into account as well as the drop out or addition of clients between communication rounds.

In a real environment solution these variables may cause the test result presented above not to be the same in practice. This is because the time spent on communications can be a major bottleneck for the program. In addition, the server has to be prepared to lose and add participants over time, just as clients have to be prepared to be able to leave the training and not start again from the point where they left. That said, it is not possible to accept the results shown in figure 4.3 as absolute.

This makes the the mock tests an inaccurate picture of FL behavior. Nevertheless, with the simulations performed it is already possible to make some comparisons between a FL algorithm and a centralized ML algorithm.

As such, we decided to invest time in designing and implementing a real-world solution, i.e. one that would give an idea of a federated solution with the clients having to communicate with the server and vice-versa.

4.4 Real-World

For the development of this solution a system design was conducted. This system design serves as a guide to the implementation of the real-world Federated Learning solution.

4.4.1 System Design

The server/client and client/server communication is done through a REST server. Since the main goal is not to encrypt the information sent nor to protect the client or server from attacks, a REST server presents a simple way to communicate through POST messages with a JSON formatted body, while allowing us to do a proof of concept study.

Another important point of this solution is the implementation of all the training logic on the client in a mobile Android application. For this, research on Android programming has been done.

The logic implemented on the server remains the same, i.e. the function that aggregates the weights coming from the clients and the function that tests the updated global model remain the same.

Finally only the last point of this solution remains to be defined: the Android devices. Purchasing devices just for testing this solution would become a considerable investment. In addition, due to the current COVID pandemic, we also faced difficulties in using existing mobile devices from our academic network. Hence, it was decided to use Android mobile device emulation tools on the student's personal machine. There are numerous emulation software for the desired purpose, such as BlueStacks, Nox Player, MEmu Play, etc., as well as Android Studio itself where the Android application (.apk) development is done.

Figure 4.4 shows a representation of how the solution performs in a real environment through a high level diagram.

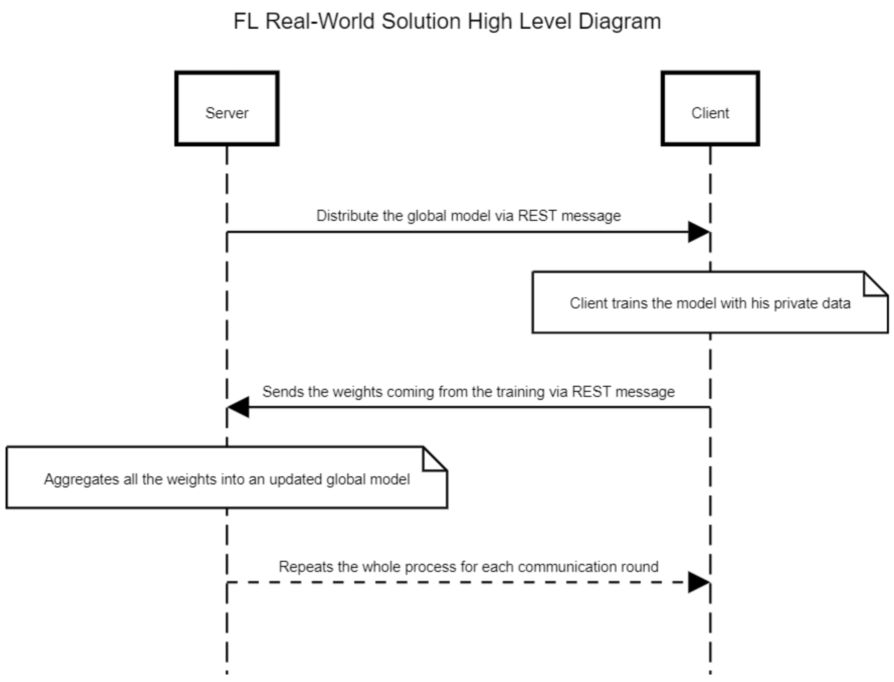
FL Real-World Solution High Level Diagram
Server
Client
Distribute the global model via REST message
Client trains the model with his private data
Sends the weights coming from the training via REST message
Aggregates all the weights into an updated global model
Repeats the whole process for each communication round

Fig. 4.4 Federated Learning Real-World Solution High Level Diagram.

4.4.2 Development and Implementation

Now that the system design has been completely aligned it is time to develop and implement the solution. In the first phase, the REST server is implemented, clarifying how the communication will be done during the whole process.

4.4.2.1 REST Server (Server Implementation)

As seen before, the REST server has the purpose to start the communication rounds and make sure that the updated weights are sent from the client to the server. This server also sends the updated model to all clients.

The architecture of this REST API is composed as follows:

- GET model to get the latest shared model in this server
- POST model to upload the updates from the clients
- GET round to obtain the info about the current round

Each training round starts when the server is initialized. Afterwards, the server receives the updates from all clients. When a client sends the updated weights to be aggregated to the global model these are stored by the server in a JSON format file. This file contains all the information from each client of all the training rounds.

After receiving all the information from a minimum number of clients, the server proceeds to aggregate the global model. In a production environment, the server should be protected to not receive new information during each training round so that there is never a system crash or deterioration of the training during the process. However, this is a controlled case, so the server only receives information from a stipulated number of clients during the implementation and admits that all available clients will participate in each training round, i.e. during this dissertation the drop out or addition of clients in the middle of the process is not covered.

Once performed the aggregation of the model, the server sends the model to all participating clients in the response of the message received initially by each client.

4.4.2.2 Android Application (Client Implementation)

In this chapter, the implementation made for training each client has already been covered. Thus, this implementation can be leveraged, by migrating to the Android language in order to create a functional .apk file, which allows any user to install the app on an Android mobile phone.

However, it is still necessary to include the communication with the server. To do this, each client sends a REST message to an endpoint created for the server shown above. This endpoint uses the IP address of the server in order to be able to communicate. The client will send the weights from the local training and receive as a response the updated global model that will be used in the next round.

During the implementation some tests were made with Android devices emulated by Android Studio. These tests showed that the software in question is quite heavy, so it would not be possible to scale the same tests for a higher number of clients used. That said, a research was done in order to understand what would be the best solution to do tests with more clients. From this research, the BlueStacks software already mentioned was chosen.

Like any other emulator, BlueStacks creates a virtual version of an Android device that runs in a window on the computer. In this virtual client the application generated in Android Studio is installed to run the federated solution.

The Multi-instance Manager lets you create multiple instances of BlueStacks 5. You can use these instances to play several games together, use different accounts at the same time and farm more easily in many different games. From this feature it is possible to emulate several devices in order to test more cases during this dissertation.

In an ideal case it is necessary to take into account the computational capacity and energy efficiency of each client, since in most cases the client devices will be heterogeneous. Obviously this point affects the training considerably, as there will be clients that can perform their local training faster than others. This not only affects the total training time, but can also affect the ability of each client to train with a considerably large dataset. That said, in this solution, all the devices will be homogeneous. Therefore, the impact of using training devices with different conditions for future work.

4.4.3 Real-World Results

All conditions are now gathered to perform tests using the solution presented above. These tests allow us to get a clearer idea of a federated solution compared to the simulations already performed. One should note that during the research of existing solutions carried out for of this dissertation few cases were found where the results obtained came from a solution in a real environment.

In a first case, a test is made that serves as a comparison with the last simulated test performed. That is, we perform a test where the CIFAR10 dataset is divided equally by android devices. This test aims to understand if the accuracy of the training is affected when the devices are real and not created by the available frameworks for FL simulation.

Table 4.2 Results obtained from the real-world solution with 2 devices.

Number of Elapsed Rounds	Number of Workers Present in Training	Average Accuracy
10	2	79%
30	2	88%
50	2	90%
100	2	91%

When comparing table 4.2 with figure 4.3 it is possible to admit that the solution in real environment does not detract from the accuracy along the rounds. Although the training time is not taken into account, it is important to mention that the training time in this solution increases a lot, however this is quite expected since the server and the Android devices are emulated on the same machine and, on the other hand, the FL simulation frameworks are more optimized than this solution. Clearly the time varies depending on the capacity of the machine used, however, the time obtained is not valid because in an ideal case each node only computes its own training and this is done entirely in parallel.

Next, focus was given to how a client with a incomplete dataset present in the training will affect the accuracy over the rounds. For this dissertation, a incomplete dataset is a quantitative dataset, that is, only the amount of images available for local training on each client will determine the quality of the dataset. Clearly, there are several conditions that make a dataset incomplete, such as the quality of the images used, i.e., there may be blurry images, pixelated images, or even images that are impossible to identify. However, these cases are left for future work on this topic.

Having said this, three test cases are presented below. In the first case (Table 4.3) there are two clients present in the training, where one has half of CIFAR10 (25,000 images) and the second has only 10 images (one image for each class in CIFAR10).

Table 4.3 Results obtained from the real-world solution with 2 devices (One with 50% of CIFAR10 and another with 0.0002%).

Number of Elapsed Rounds	Number of Workers Present in Training	Average Accuracy
10	2	27%
30	2	34%
50	2	42%
100	2	48%

In the second case (Table 4.4), the number of clients is increased to 4, the dataset is divided non-uniformly among 3 of them, but one of the clients sees its dataset being assigned with 0,0002% of the complete training batch of CIFAR10 (one image for each class in CIFAR10). The other 3 clients have 15%, 25% and 60% of the dataset, respectively.

Table 4.4 Results obtained from the real-world solution with 4 devices (15%, 25%, 60% and 0.0002% of CIFAR10, respectively).

Number of Elapsed Rounds	Number of Workers Present in Training	Average Accuracy
10	4	22%
30	4	43%
50	4	78%
100	4	90%

Table 4.5 shows the last case. In this scenario, the number of clients is increased again, this time to 8. Six of the clients have half of the dataset available to the 3 clients with a decent dataset: two clients with 7.5%, two with 12.5% and two with 30%. The two remaining

clients have 10 images each, always keeping in mind that they have one image of each class existing in CIFAR10.

Table 4.5 Results obtained from the real-world solution with 8 devices (two clients with 7.5%, two with 12.5%, two with 30% and two with 0.0002%).

Number of Elapsed Rounds	Number of Workers Present in Training	Average Accuracy
10	8	12%
30	8	26%
50	8	44%
100	8	89%

The first case clearly shows a poor result. This result is expected, since the training is done with only half of the total dataset. The second and third cases show that increasing the number of clients and, consequently, each client having a smaller local dataset, along with the introduction of clients that do not contribute at all to the overall training, still allows reaching an accuracy close to the maximum achieved in previous tests. However, it is also possible to observe that the accuracy in the initial training rounds is considerably worse than in the cases presented before.

These findings can be seen in more detail in the graph shown in figure 4.5. This graph gives a better insight into the overall training behavior shown in the previous tables.

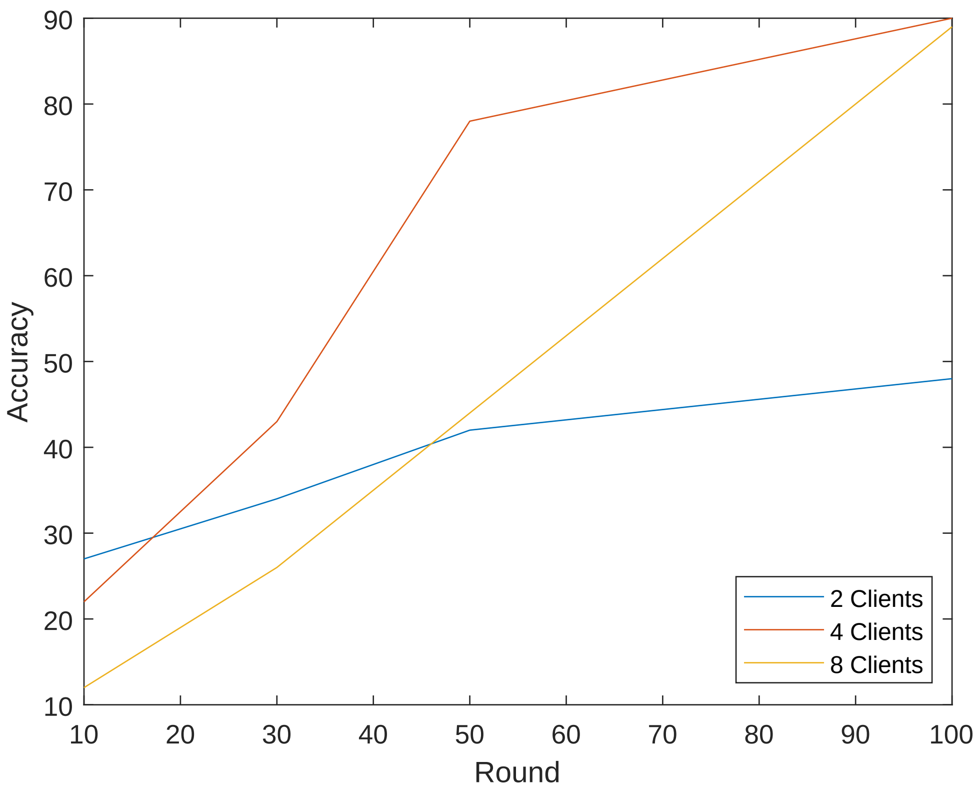

Fig. 4.5 Comparison between 2 clients with 50% and 0.0002% of the dataset, 4 clients with 15%, 25%, 60% and 0.0002% of the dataset, and 8 clients with 7.5%, 7.5%, 12.5%, 12.5%, 30%, 30%, 0.0002% and 0.0002% of the dataset.

The previous results substantiate the need to use an aggregation method that benefits the clients that contribute the most to the global model. This can be done through the use of a weighted average, where more weight is given to the weights coming from clients with a more complete dataset, that is, clients that ultimately make the model reach a higher accuracy.

That said, the aggregation model was changed in order to comply with this premise. In order to accomplish this new use, case a controlled weighted average was done, i.e., given the number of images each client has locally as well as the total number of CIFAR10 images.

Clearly, FL prioritizes privacy above all else, so this would not be possible in a real case. However, this development is only intended to visualize the contribution of an aggregation method where it is not assumed that all clients contribute equally to the overall model.

Therefore, equation 4.2 represents the weighted average used hereafter.

$$w \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w^k \tag{4.2}$$

In this equation n represents the total number of CIFAR10 images and n_k represents the number of images present in each client. With this aggregation method little importance is given to clients with 10 local images when calculating the average. Below, the cases presented above are repeated, but using now a weighted average.

The following table shows the results obtained by repeating the test with three clients (one client with 15%, one with 25% and one with 60% of CIFAR10) and one with only 10 images.

Table 4.6 Results obtained from the real-world solution with 4 devices with Weighted Average (15%, 25%, 60% and 0.0002% of CIFAR10, respectively).

Number of Elapsed Rounds	Number of Workers Present in Training	Average Accuracy
10	4	73%
30	4	84%
50	4	88%
100	4	91%

In the graph shown in Figure 4.6 it is possible to admit that the use of a weighted average when aggregating the global model improves the training in the initial rounds. However, in both cases an identical accuracy is reached at the end of the training, given enough training rounds.

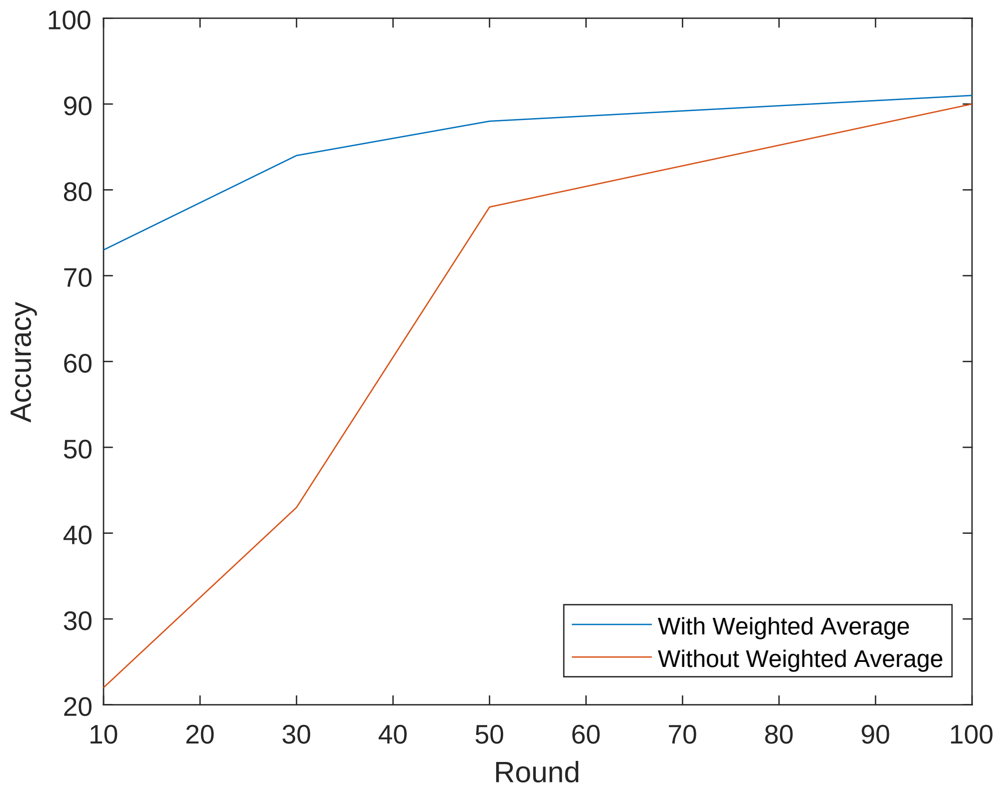

Fig. 4.6 Comparison between training with 4 devices (Weighted Average vs Arithmetic Average)

Table 4.7 presents the results obtained from the test with 8 clients (two with 7.5%, two with 12.5% and two with 30% of the dataset, and 2 clients with 10 images for each class present in CIFAR10).

Table 4.7 Results obtained from the real-world solution with 8 devices with weighted average (two clients with 7.5%, two with 12.5%, two with 30% and two with 0.0002 %).

Number of Elapsed Rounds	Number of Workers Present in Training	Average Accuracy
10	8	36%
30	8	55%
50	8	79%
100	8	90%

In addition, Figure 4.7 shows a graph comparing the test presented in the previous table with the same test performed with the arithmetic mean when aggregating the global model.

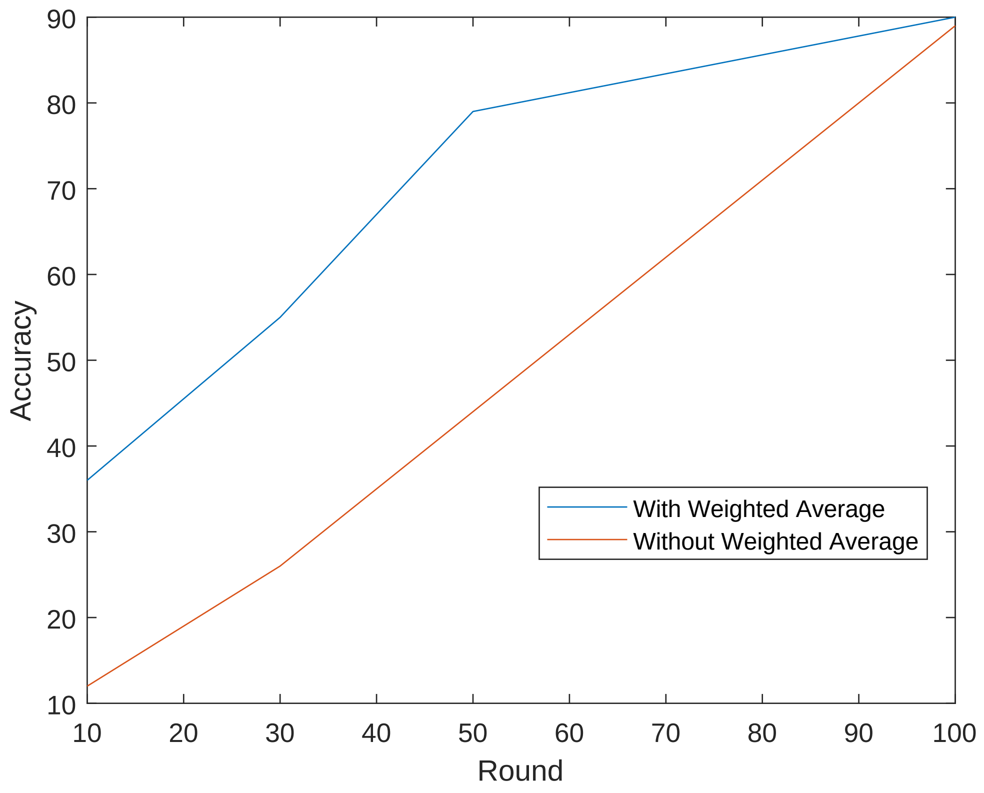

Fig. 4.7 Comparison between training with 8 devices (Weighted Average vs Arithmetic Average).

As expected, in general, the training is better when a weighted average is used for both cases. This is due to the fact that clients with more percentage of dataset assigned are more favored than clients with only 10 images, which have almost no influence on the model aggregation.

In order to test this solution more generally, a test for each of the 8-client cases (with and without weighted average) with cross-validation has been performed.

In this case, the 50,000 training images were divided into 5 batches of 10,000 images. Consequently, these five batches were divided into two with 12.5%, two with 7.5% and two with 30% of their images (figure 4.8). That said, one training run was performed for each batch of 10,000 images. This test serves to validate that the presented solution produces equivalent results for several different cases.

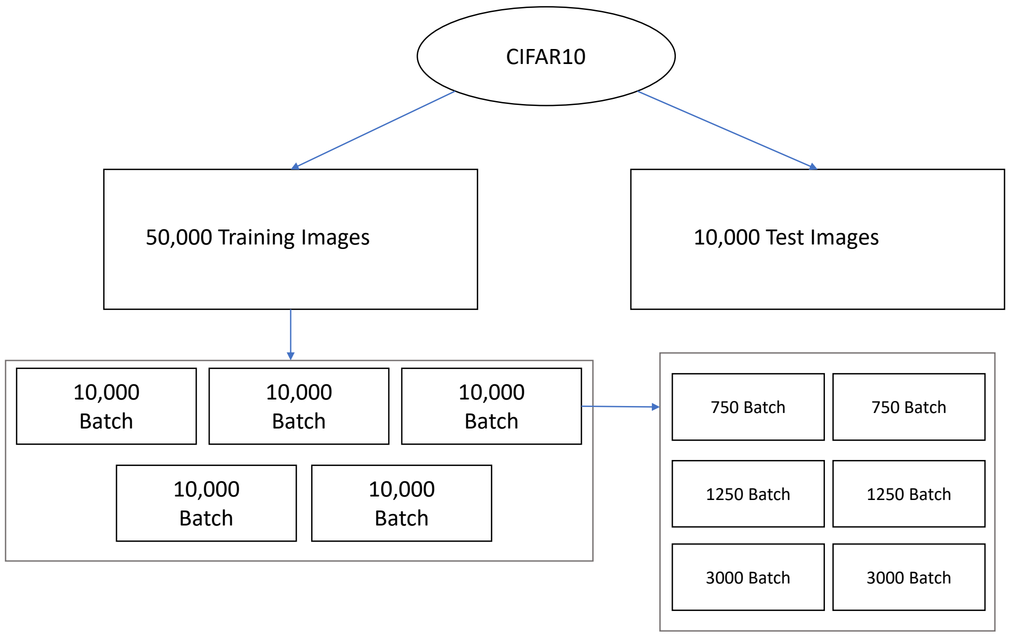

Fig. 4.8 Division of CIFAR10 used for the tests with cross-validation.

That said, the following table presents the average of the results obtained for the two cases presented.

Table 4.8 Results obtained with cross-validation (Weighted Average in green vs Arithmetic Average in red).

Number of Elapsed Rounds	Number of Workers Present in Training	Average Accuracy
10	8	28%
		8%
30	8	42%
		17%
50	8	74%
		42%
100	8	90%
		89%

Thus, it is admissible that the solution presented is valid for a generality of cases and not only the cases presented during this section.

Using the weighted average presents an overall improvement in training, however, for CIFAR10, both the weighted average and arithmetic average cases achieve a similar maximum accuracy. Therefore, we have decided to do a final test with a larger dataset – CIFAR 100 – to assess the performance of our FL solution and

This dataset has both the same number of training and test images as CIFAR10. The difference between these two datasets is in the number of image classes, i.e., while CIFAR10 presents 10 image classes, CIFAR100 presents 100 image classes (Figure 4.9).

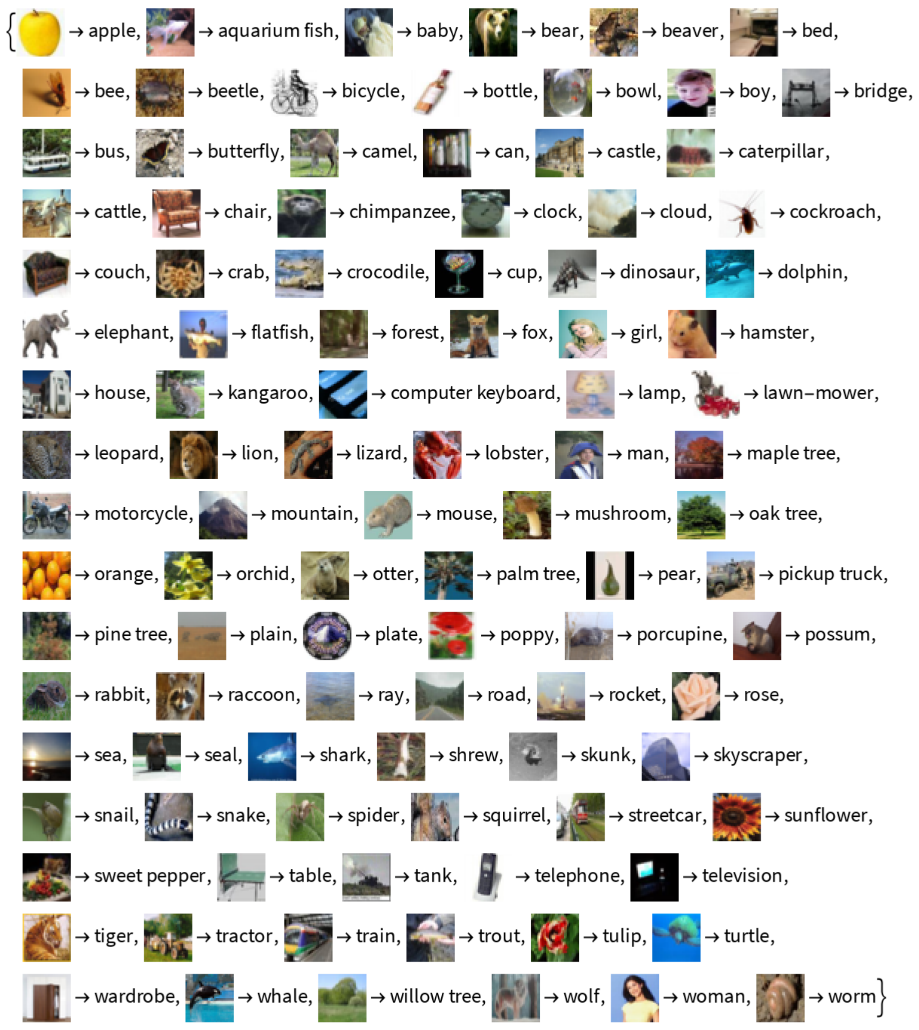
{ → apple, → aquarium fish, → baby, → bear, → beaver, → bed,
→ bee, → beetle, → bicycle, → bottle, → bowl, → boy, → bridge,
→ bus, → butterfly, → camel, → can, → castle, → caterpillar,
→ cattle, → chair, → chimpanzee, → clock, → cloud, → cockroach,
→ couch, → crab, → crocodile, → cup, → dinosaur, → dolphin,
→ elephant, → flatfish, → forest, → fox, → girl, → hamster,
→ house, → kangaroo, → computer keyboard, → lamp, → lawn–mower,
→ leopard, → lion, → lizard, → lobster, → man, → maple tree,
→ motorcycle, → mountain, → mouse, → mushroom, → oak tree,
→ orange, → orchid, → otter, → palm tree, → pear, → pickup truck,
→ pine tree, → plain, → plate, → poppy, → porcupine, → possum,
→ rabbit, → raccoon, → ray, → road, → rocket, → rose,
→ sea, → seal, → shark, → shrew, → skunk, → skyscraper,
→ snail, → snake, → spider, → squirrel, → streetcar, → sunflower,
→ sweet pepper, → table, → tank, → telephone, → television,
→ tiger, → tractor, → train, → trout, → tulip, → turtle,
→ wardrobe, → whale, → willow tree, → wolf, → woman, → worm }

Fig. 4.9 Classes of CIFAR100.

In the following table it is possible to observe the results obtained by repeating the tests presented in Tables 4.5 and 4.7.

Table 4.9 Results obtained from 8 clients (two clients with 7.5%, two with 12.5%, two with 30% and two with 0.0002 %) with CIFAR100 dataset (Weighted Average in green vs Arithmetic Average in red).

Number of Elapsed Rounds	Number of Workers Present in Training	Average Accuracy
10	8	6%
		2%
30	8	21%
		12%
50	8	32%
		24%
100	8	63%
		51%

Note that for the same training rounds with this dataset, an accuracy of 63% is reached. This is to be expected since the dataset presents a higher level of complexity. The author of [30] shows that with the **VGG19** CNN the CIFAR100 dataset achieves a maximum accuracy of 70.3%.

These outcomes support the use of aggregation methods that benefit results from clients that contribute more to training than others.

Chapter 5

Conclusion

The goal of this dissertation is to explore and implement Federated Learning solutions and, consequently, to compare this method with centralized Machine Learning. In order to achieve this goal, a functional federated solution was implemented in a simulation environment. This solution demonstrates that the distribution of training, as well as the decentralization of data, does not introduce a significant degradation in the accuracy achieved when compared with a centralized ML baseline under the same conditions. Despite these initial conclusions, it was observed that a simulated federated solution does not provide great perception of the bottleneck of the communication, that is, it does not provide a perception about the impact of server/client and client/server communication in the system.

That said, there is a need to implement a real solution, even if it leaves out some variables that come with the FL itself, to have a more accurate perception about the method. From this solution it is concluded that, in general, the overall results are equivalent to those obtained in the previous simulations. Both the real-world solution and the simulated solution achieved a maximum accuracy of 91%.

However, this solution shows up that a system built on a FL architecture has some weaknesses, such as:

- By introducing the need for communication between clients and server, the concept of drop-out and adding them on the fly needs to be addresed. For these cases, both the server and the FL architecture need to be extremely resilient. If the system is not prepared to handle a drop-out or a large influx of users during a communication round there can be considerable degradation in the final result, or even a fatal system failure;

- With the need for privacy in the data present in each client also comes the need for protection against cyber attacks or malicious clients. That said, there is a need to encrypt all communications, as well as the dataset present in each client;
- At the outset, in a federated solution, there is no knowledge of what kind of data each client possesses. That said, there may be clients that do not contribute to the increase in accuracy achieved with the global model, i.e., there may be clients that do local training with data that is not beneficial to the system. Furthermore, clients are in most cases heterogeneous, i.e., not all clients have the same computational power or energy efficiency, so there may be clients that take considerably longer to perform local training than others. In these cases the training time may increase substantially.

However, FL presents very important advantages when compared to centralized ML. Data privacy at the clients is, above all, the focal point of the presented concept. Furthermore, privacy brings two huge advantages: *i*) distributed training, i.e. there is no longer the need to have a machine on the server that is capable of running extremely heavy algorithms, and there will be lighter training on each client. In addition, in an ideal federated learning solution, this training is performed completely in parallel; *ii*) The need for distributed training implies the decentralized use of data. Thus, a federated solution presents a dataset that grows exponentially as the number of clients in the program increases, i.e., it makes the model more complete and prepared to respond positively to more different situations.

That said, this dissertation allows to suggest incorporating admission criteria when implementing a solution with Federated Learning. This type of selection allows to introduce into the system several securities:

- Ensure that all clients have sufficient computational capacity to contribute positively to the training;
- Ensure that all clients have enough data to help converge the model;
- Ensure that all clients are beneficial, i.e. do not present a privacy risk to others.

We believe that this solution can improve the overall quality of an FL implementation.

We conclude that FL presents improvements in some aspects over centralized ML. However, FL does not replace centralized ML in all its applications, i.e., when privacy or the lack of available data is not a problem for those implementing a solution, centralized machine learning continues to be the most robust solution.

That said, federated learning justifies implementation in financial, medical and any other application, where the sensitivity of customer data is critical.

Considering the work developed and the results obtained, there are several possibilities for continuing the project to improve the current work, such as:

- Implement a federated solution with physical mobile devices;
- Improve the aggregation method to further optimize model convergence;
- Experiment with encryption techniques for server/client and client/server communication and even the data present in each client;
- Test a federated solution with larger and more complex datasets in order to validate the applicability of the model for other cases;
- Explore cyber security techniques to enable sustainability for all participants;
- Implement solutions to deal with drop-out and addition of customers during the course of the solution in order to increase its robustness.

References

[1] M. Duranton, K. De Bosschere, B. Coppens, C. Gamrat, T. Hoberg, H. Munk, C. Roderick, T. Vardanega, and O. Zendra, *HiPEAC Vision 2021*, 2021. [Online]. Available: www.magelaan.be

[2] T. Coughlin, "175 zettabytes by 2025," Nov 2018. [Online]. Available: https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/?sh=59d7f05f5459

[3] "Data creation and replication will grow at a faster rate than installed storage capacity, according to the idc global datasphere and storagesphere forecasts." [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS47560321

[4] M. Satyanarayanan, "Mahadev (satya) satyanarayanan - edge computing: a new disruptive force, keynote at systor 2020," Oct 2020. [Online]. Available: https://www.youtube.com/watch?v=7D2ZrMQWt7A

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

[6] H. B. Mcmahan and D. Ramage, "Communication-Efficient Learning of Deep Networks from Decentralized Data," vol. 54, p. 10, 2017.

[7] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[9] A. Srinivasan, “Difference between distributed learning versus federated learning algorithms,” 2022. [Online]. Available: https://www.kdnuggets.com/2021/11/difference-distributed-learning-federated-learning-algorithms.html

[10] S. Greengard, “Ai on edge,” *Commun. ACM*, vol. 63, no. 9, p. 18–20, Aug. 2020. [Online]. Available: https://doi.org/10.1145/3409977

[11] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3298981

[12] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, “Federated learning,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[13] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y. C. Liang, Q. Yang, D. Niyato, and C. Miao, “Federated Learning in Mobile Edge Networks: A Comprehensive Survey,” *IEEE Communications Surveys and Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[14] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, “Adaptive Federated Learning in Resource Constrained Edge Computing Systems,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[15] I. G. Courville, Y. Bengio, and Aaron, “Deep learning by Ian Goodfellow,Yoshua Bengio, Aaron Courville,” *Nature*, vol. 29, no. 7553, pp. 1–73, 2016.

[16] X. Pan, J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, “Revisiting Distributed Synchronous SGD,” pp. 1–10, 2017. [Online]. Available: http://arxiv.org/abs/1702.05800

[17] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, “Federated learning for mobile keyboard prediction,” *arXiv preprint arXiv:1811.03604*, 2018.

[18] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

[19] T. Li and B. Rao, "Algebraic Preliminaries," *Progress in Nonlinear Differential Equations and Their Application*, vol. 94, pp. 19–30, 2019.

[20] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards Federated Learning at Scale: System Design," 2019. [Online]. Available: http://arxiv.org/abs/1902.01046

[21] C. H. K. van Berkel, "Multi-core for mobile phones," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. Leuven, BEL: European Design and Automation Association, 2009, p. 1260–1265.

[22] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[23] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/6211080fa89981f66b1a0c9d55c61d0f-Paper.pdf

[24] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–14, 2018.

[25] Google, "Tensorflow federated: Machine learning on decentralized data." [Online]. Available: https://www.tensorflow.org/federated

[26] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," pp. 1–5, 2018. [Online]. Available: http://arxiv.org/abs/1811.04017

[27] H. Yadav, "Preserving data privacy in deep learning: Part 1," Sep 2020. [Online]. Available: https://towardsdatascience.com/preserving-data-privacy-in-deep-learning-part-1-a04894f78029

[28] C. Zhu, R. Ni, Z. Xu, K. Kong, W. R. Huang, and T. Goldstein, "GradInit: Learning to Initialize Neural Networks for Stable and Efficient Training," no. NeurIPS, 2021. [Online]. Available: http://arxiv.org/abs/2102.08098

[29] A. Kaushik, “Understanding the vgg19 architecture,” Feb 2020. [Online]. Available: https://iq.opengenus.org/vgg19-architecture/

[30] R. Sterneck, A. Moitra, and P. Panda, “Noise Sensitivity-Based Energy Efficient and Robust Adversary Detection in Neural Networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.