Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Master's diploma thesis

in the field of study Computer Science and Information Systems
and specialisation Artificial Intelligence Methods

Federated learning enabler - design and implementation

Piotr Niedziela

student record book number 276855

thesis supervisor
Dr hab. Marcin Paprzycki, prof. nzw.

WARSAW 2021

....

supervisor's signature author's signature

Abstract

Federated learning enabler - design and implementation

Federated Learning is a machine learning approach, where training is based on datasets distributed across different clients.

The key feature of federated learning is that the data does not “leave the owner”, while the public part is only the shared model, trained jointly. Intensive development in this field began in 2017. Since then, multiple federated learning platforms have been created. However, all these platforms support mainly their tools. In addition, these solutions focus on enabling the federated learning process to be successfully completed (for a given dataset). Their main disadvantage is that they do not support research concerning open questions in various aspects of federated learning, due to the difficulty of introducing modifications/adding new features.

The aim of this work is to develop foundation of a modular federated learning platform. This platform is to be flexible and support federated learning related research. To realise this goal, the most important articles on federated learning have been reviewed. As a result it was established what has already been done, what problems have to be faced, and examples of applications in various fields.

Based on conclusions gathered in the extensive literature review, the platform has been designed. Its key feature is flexibility. By using dynamically loaded modules, it enables instantiation of different forms of machine learning. This allows the use of various tools and adding modules that are hardware optimised, or facilitate additional functions that are not part of standard machine learning approaches. It also allows to coordinate learning process workflows. Conducted work includes also platform tests that show how the accuracy of learning depends on various parameters. Additionally, it illustrates how one can easily complete varying scenarios.

Keywords: Federated Learning, Machine Learning

Streszczenie

Środowisko uczenia federated learning – project i implementacja

Federated Learning to podejście oparte na uczeniu maszynowym, w którym szkolenie opiera się na zbiorach danych rozproszonych między różnymi klientami. Kluczową cechą uczenia federacyjnego jest to, że dane "nie opuszczają właściciela", podczas gdy część publiczna to tylko wspólnie wyszkolony model, który jest współdzielony. Intensywny rozwój w tej dziedzinie rozpoczął się w 2017 roku. Od tego czasu powstało wiele platform umożliwiających przeprowadzenie uczenia federacyjnego. Jednak wszystkie powstałe platformy obsługują tylko swoje narzędzia. Ponadto, rozwiązania te koncentrują się na umożliwieniu pomyślnego zakończenia federacyjnego procesu uczenia się (dla danego zestawu danych). Ich główną wadą jest to, że nie wspierają badań dotyczących otwartych problemów w różnych aspektach federacyjnego uczenia się, ze względu na trudność wprowadzania modyfikacji/dodawania nowych usprawnień.

Celem tej pracy jest opracowanie podstaw modułowej platformy uczenia federacyjnego. Platforma ta ma być elastyczna i wspierać badania związane z federacyjnym uczeniem się.

Aby zrealizować ten cel, dokonaliśmy przeglądu najważniejszych artykułów na temat federacyjnego uczenia się. W efekcie ustalono, co już zostało zrobione, z jakimi problemami trzeba się zmierzyć i przykłady zastosowań w różnych dziedzinach.

Platforma została zaprojektowana na podstawie wniosków zebranych w obszernym przeglądzie literatury. Jej kluczową cechą jest elastyczność. Korzystanie z dynamicznie ładowanych modułów umożliwia uruchomienie różnych przypadków federacyjnego uczenia. Pozwala to na korzystanie z różnych narzędzi i dodawanie modułów zoptymalizowanych pod kątem sprzętu lub ułatwiających korzystanie z dodatkowych funkcji, które nie są częścią standardowych podejść do uczenia maszynowego. Pozwala również na koordynację procesu uczenia się.

Prowadzone prace obejmują również testy platformowe, które pokazują, jak dokładność uczenia się zależy od różnych parametrów. Dodatkowo ilustruje, jak łatwo można wykonać różne scenariusze.

Słowa kluczowe: uczenie federacyjne, uczenie maszynowe

Warsaw, ...

Declaration

I hereby declare that the thesis entitled „Federated learning enabler - design and implementation", submitted for the Master degree, supervised by dr inż. Marcin Paprzycki, prof. nzw., is entirely my original work apart from the recognized reference.

....

Contents

Introduction

Federated learning (FL) can be traced back to, at least, 2017 [47] (though the concept was discussed already in 2016). Hence, it is a relatively new research area. However, it has roots in research conducted at least at the end of last century. Here, one of classic contributions is the work of Cantu Paz [26], devoted to parallel genetic algorithms. In this work, separate nodes of a parallel computer evolved their own, independent, genetic models and, from time to time, exchanged best (local) genomes. Later, this approach became known as island model of genetic algorithms. One example of this approach is described in [52], where the authors investigate whether island model might be good to apply the separable nature of the problems. They also investigate how island model can track multiple search trajectories, by using the infinite population models of genetic algorithms. Another interesting example of using island model of genetic algorithms is showed in [32]. The authors consider the problem of the appropriate configuration of control parameters (e.g. mutation parameter). They describe the problem of setting control parameters using the standard, island model distributed genetic algorithm. They proposed a strategy that statically assigns random control parameter values to each node. Their experiments showed that this approach gives good results, when compared to a homogeneous distributed genetic algorithm. An interesting approach was used in the article [42]. The authors compare the performance of the parallel and serial island model genetic algorithm for solving the multiprocessor scheduling problem. They present the results of using fixed and scaled problems both using and not using migration.

In general, this work belongs to the class of approaches known as distributed machine learning (DML), which can be summarised as follows. A computational task (involving some form of machine learning) is split into sub-tasks that are independently executed on separate "nodes". Here, a node can be any computing device, e.g. a smartphone, laptop, desktop, server, etc. Next, results obtained at each node are combined to create a "common model". Combination of local "results" can be applied once, at the end of the process, or can be performed repeatedly, after a certain amount of local work is completed by each node (e.g. a training epoch). Moreover, delivery of intermediate results, to the shared model, can be facilitated using a "master-slavetype" approach, where a selected node is tasked with model aggregation. After new version of the model is assembled, the process is complete, or it is sent back to the participating "slaves"' to continue the training process. Communication of intermediate results can also be completed using some form of a "peer-to-peer" approach. Here, selected groups of nodes (or all nodes – all-to-all communication) exchange their updates (without a centralised "manager" that oversees the process). In the latter case, information exchange continues until the same global model "materializes" at each node. Both approaches have advantages and disadvantages, but discussing them is out of scope of this Thesis (for more details, see [51]).

To understand why FL was proposed, and became an instant success, one should notice that majority of, above outlined, DML is based on "restricting" assumptions. The most important of them are: (a) all data belongs to a single owner/stakeholder, and (b) all data is processed within a "single computer". The latter should be understood as follows. While it is possible that computing takes place within a network of workstations, loosely connected using PVM [31], or Condor middleware [30], or some other middleware, all nodes that are part of this infrastructure have a single owner. Hence they can and should be treated jointly as a part of single "virtual computer".

Keeping this in mind, let us reflect on changes that took place within last few years (and are still ongoing). Obviously, this list is not exhaustive. (1) Smartphones contain multiple (typically more than ten) sensors. Each of them can generate a data stream, while billions of smartphones are in use (and this number is still growing). Here, on the one hand, users may want to control data generated by their devices (this is "their private data"). According to [13], over 80% of phone users do not want to share their data in order to improve the algorithm for suggesting friends. On the other, they may want to use this data for their advantage, e.g. by deriving from it actionable knowledge. For example, people want to unlock their phone with their face, but uncommonly want to share pictures of their face in order to improve the algorithm. Here, note that although users "have data", usually they lack capabilities required for processing it. Moreover, local data covers a limited "fragment" of knowledge, and does not generalize. For instance, single person mobility patterns will not help improving smart city services. For this, mobility patterns of a majority of citizens are needed. Furthermore, to address this issue, a solution was proposed by *Google* [23]. They use federated learning to streamline Gboard, an application that is used to display tooltips as you type text. The authors state that the *data* is distributed across *millions*(!) of devices and thanks to federated learning there is no need to keep ones data in the cloud. An additional advantage is that the customer's data is kept private (if we believe that any data that is known to Google can be seen as private).

(2) With technological advancements, the size and prices of sensors continue decreasing, while the number of aspects that can be measured is growing. For instance, the research [18] predicts that between 2019 and 2026, the annual growth of the sensor market will grow by 6.22% per year. This means that during this period it will increase by over 60% in total. The research also takes into account that the price of sensors will decrease over time. Another interesting analysis of this market can be found in [10]. Figure 0.1 shows the estimated number of IoT devices in 2020-2030. It is easy to notice that the number of devices will increase by approximately 3 times by 2030.

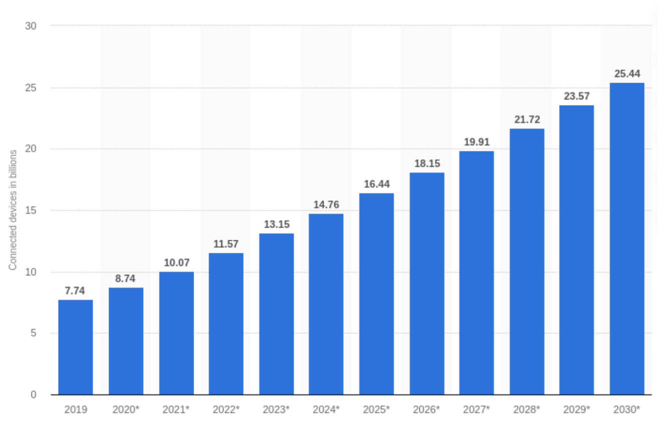


Figure 0.1: Number of IoT devices

Source: Adapted from [11]

This enables heterogeneous sensors to be easily placed at “any location”. Moreover, such sensors (sensor clusters) can belong to multiple stakeholders, even if they measure the same parameter. For example, consider the problem of predicting weather in a city. There can be many sensors in a big city. Some of them may belong to the city, others to private institutions, and still others ordinary residents that may have them placed/installed in their houses (gardens). As one can easily guess, the larger number of sensors, the better the result of weather (air pollution, etc.) prediction. Therefore, it is best to use as many sensors as possible, combining these from different sources, for prediction. Good example is presented in [12]. This article describes scientists' progress in introducing federated learning into smart cities. They claim that federated learning holds potential for protecting data for air quality monitoring, energy management, and overall daily living in a city. Moreover, they claim that machine learning algorithms could also secure data in transportation systems, protect critical infrastructure and serve police departments and emergency personnel.

Hence, delivery of user-centered applications requires use of data belonging to different owners that may not be willing/able to share it. Here, a great example is *FedVision* [4] – a machine learning platform to support the development of federated learning powered computer vision applications. The platform has been created to help customers develop computer vision-based safety monitoring solutions in smart city applications. Authors mark, that during four months of use, the platform significantly improved accuracy and reduced costs. The important thing is that the platform is used by 3 major corporate customers who cannot share their data due to privacy reasons.

(3) Proliferation of wireless networks, including first deployments of 5G networks, allows efficient implementation of sensor networks. This, in turn, makes it easy to establish communication channels between sensors, actuators, edge devices, computing nodes, gateways, cloud services, etc. According to research reported in [1], the global 5G IoT market size will grow from USD 2.6 billion in 2021, to USD 40.2 billion by 2026. They also predict that the healthcare segment will grow the fastest. IoT devices in healthcare reduce healthcare costs and improve its quality.

(4) The later supports progress in research, development and deployment of Internet of Things (IoT) ecosystems. To deliver services to the users, such ecosystems need, broadly understood, (5) machine learning (ML). A good example is an approach called *SplitNN*. It is described in [34]. The neural network is split into 2 or 3 parts and only one of these parts is trained by an IoT device. This solution was proposed because IoT devices have low computation power and are unable to train the model themselves.

Since, as noted, many newly popular ML approaches are based on large-scale neural networks (NN), (6) hardware dedicated to NN-based ML has been developed. It includes extra small devices that can be placed "almost anywhere", and may belong to different owners. Examples of such devices are Intel Movidius [9] which offers 1 tera-operations per second, and can be used as a USB pen drive. Another example is NVIDIA Jetson Nano [50], with 128 cores and capability of 0.5 TFLOPS. It is enough, for instance, to run multiple neural networks on several streams of data from high-resolution image sensors. Additionally, Texas Instruments [16] offers a solution to run the neural network on portable processor. Their solution is system-on-chip with a dedicated deep-learning accelerator on-chip, which can achieve 8 TOPS. These are just some examples but number of such devices is large, indeed.

Finally, (7) data that is generated locally, often cannot be transferred to the cloud for processing. On the one hand, data elements may be too large and too many. Estimates made by [14] say that more than 1.4 trillion photos are taken every day. Even if we wanted to use a small part to improve the algorithm (e.g. categorisation of photos) it would be impossible due to their huge size (e.g. images generated behind a narrow-bandwidth network), or they cannot be released. A good example is medical data that is legally protected. According to the article [2], each test that uses patient data must be approved by the appropriate authorities in a given country.

This means that the vision of a single owner of data used to train the model to realise individual goals is not the only one that needs to be taken into account. Thus, it should be obvious why, federated learning has been proposed. Here, the simplest metaphor describing FL is an approach to DML where the training of a common model involves multiple datasets contained in "local nodes", without explicitly exchanging (any) data samples. In other words, there exists a central (shared) model, training of which is the goal of the DML process. However, during the training, each participating node is using only its local data to train its sub-version of the model. Next, parameters that describe changes that resulted from training the local model, are "combined into the central model", which is redistributed (again) to the participating nodes. Obviously, possibility of using different communication topology (way in which the model parameter-updates propagate in the FL system) is open and subject to research. Such a different topology is described in [40]. In this case, there is no central server. Authors proposed an approach, where clients propagate the model to each other after making changes. Here, the main FL loop closes, and the process repeats until the common model is considered to be good enough quality.

Figure 0.2 shows the difference between FL and the typical machine learning process. The left image presents a standard machine learning process. Data owner(s) send their data to the server (e.g. a central cloud infrastructure). The server, using collected data, trains the model. It could be a neural network, decision tree, or any other ML model. The right Figure shows a federated learning scenario. There are several clients, for example, mobile phones. It could also be, for example a desktop computer, a laptop, or an IoT gateway. There is also a master device that is responsible for coordinating the learning process. Since the master device has also to store the shared model it should have considerable resources and very good Internet connectivity. Master device sends the model to a client. Then all clients train this model on their local/private data. It could be one epoch or more (depending on the training scenario). All clients send proposed model updates (parameters for model update only) back to the central server. The server aggregates the received parameters (some, or all, depending on variety of conditions, see next Sections) when it receives them. After the parameters are aggregated, the central shared model is updated, and shared once more with the clients. Then the entire process repeats. Which clients will be used in the next round of the process is also an open research questions (see, next Section). For instance, the number of clients that participate in each round of training could be limited. As a matter of fact, selection of number of nodes participating in each round of training is also one of open research questions (especially, when a very large number of nodes can, actually, participate). Moreover, clients can change between rounds. So, a particular client that participate in one round, does not have to participate in the next one.

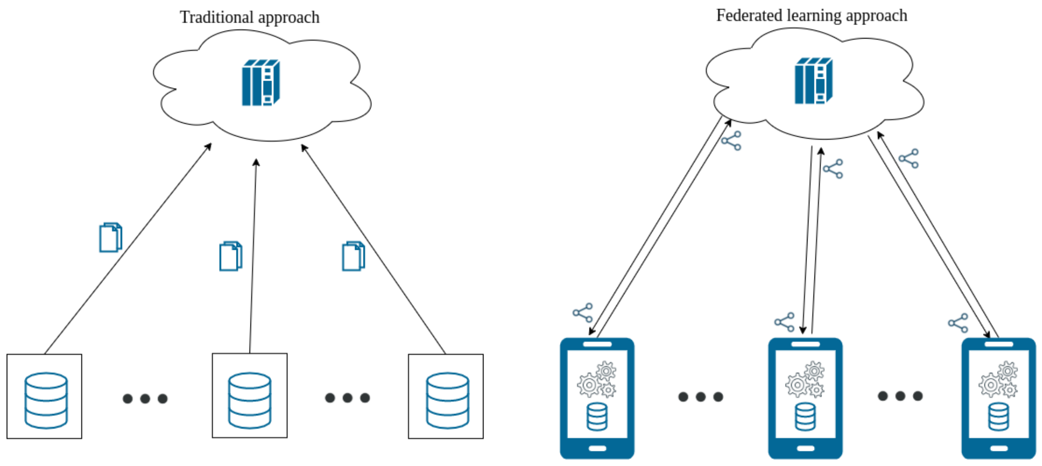


Figure 0.2: Traditional approach vs Federated Learning

While this may seem like DML, let us stress again that the underlying rationale is very different. Rather than training the model "in parallel", the focus is on protection of local data. In general, it is easy to see that only local nodes have actual access to their data, while the model is updated on the basis of exchanged parameters. It is important to note that, while the majority of current work focuses on use of neural network based ML, this is not necessary. In principle, federated learning can be applied to any approach that is amenable to DML.

Interestingly, immediate success of FL, denoted by the number of published results (see, Figure 0.3) shows that it addresses actual needs of the real-world [45]. Moreover, success of federated learning can be seen also through involvement of the largest IT companies, such as Google or NVIDIA.

Here, it can be noted, and will be described in the next Section that the ongoing FL research proceeds in two general "directions". In the first one, FL is applied in individual application domains. The main point of this work is to consider realistic (real-world grounded) scenarios, take existing, domain specific, datasets, and apply FL. This is possible, among others, since a number of platforms materialised (see, Section 1.3). Here, the main research questions are related, among others, to the relationship between splitting the data, characteristics of individual datasets, sharing only parameters of the central model, and the quality of the results. The second research area tries to deal with open issues in FL, seen as an approach to machine learning. Here,

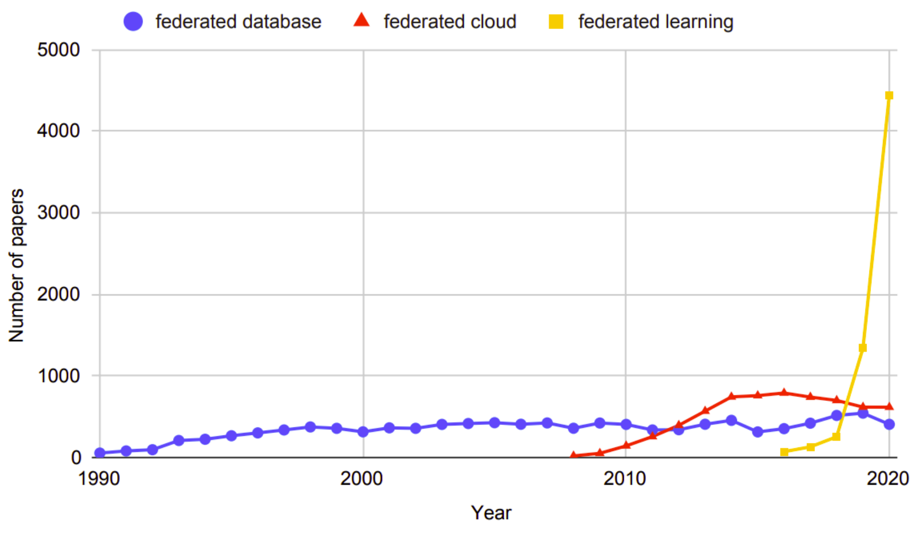


Figure 0.3: Number of published results

Source: Adapted from [45]

the main questions deal with issues that arise when FL is treated form a “meta-level perspective”. Examples of such open questions are discussed in Section 1.

Upon further reflection one can easily realise that it is the latter that suggests that a special kind of FL platform can be useful. Let us consider the case of unbalanced datasets (see, Section 2). Here, one of possible solutions, proposed in [28], requires that additional “modules” (called *Mediators*) be instantiated, and infused into the FL process. Obviously, as will be discussed, such *Mediators* cannot be easily added to standard FL platforms. Hence, researchers, who explore any ideas out of boxes of existing FL platforms have to implement “private FL engines”. Moreover, literature and Internet-sources based research (see, Section 1 and 1.3) revealed that there is no good enough solution, which would allow to run federated learning in truly distributed environments. Interestingly, most of currently existing platforms focus on simulating federated learning locally.

To address these issues a modular, flexible federated learning platform is proposed. The main idea is to provide the skeleton infrastructure for FL, and allow users to develop their own modules that can be put together, to run FL scenarios (real-world-based and research ones as well).

The remaining parts of this work report on steps that led to the development of the platform and that show its usability in FL-related research. To this effect, in Section 1.1 the most important elements of the existing (at the time of writing this Thesis) literature will be discussed. In

Section 1.3 the platforms that are currently available in the market will be presented. This Section describes their advantages and disadvantages. In Section 1.4, works presenting the architecture of platforms for federated learning will be discussed. In Chapter 2 development and implementation of the proposed platform (Sunday-FL) will be discussed. And at the end, in chapter 3, tests will be reported to illustrate key scenarios in which the platform can be naturally used, and what results it achieves.

1. Analysis of the state of the art

1.1. Literature

Let's start with a snapshot of the studies featured in the contributions stored in the arXiv repository. During the summer of 2020, we have studied 250 federated learning related articles.

Of these, 24 contained only theoretical/analytical results (no implementation at all). Here we can distinguish articles containing:

(i) mathematical analysis of communication effectiveness. These articles most often describe how the data is sent and how to better encode it so that it consumes less memory. They present various functions that describe the size of the transferred data depending on the parameters and the time of the data transfer, depending on the various variables.

(ii) mathematical aspects of federated learning. This Section includes articles that examine mathematically how the distribution of data affects learning efficiency. For example, if one of the clients has much more data than others, it has a negative impact on the effectiveness of machine learning. The same is happening for different class balances. If some users have more examples of one class than others, it will probably have a negative impact on the learning process as well. In this category of work, from the mathematical point of view, it is analysed how such aspects affect the learning process and how something can be improved. There are also articles that mathematically analyse the performance of specific networks and show why some perform better in federated learning and others do not.

(iii) Theoretical aspects of attacks against federated learning systems. If multiple users may be involved in the learning process, some may have bad intentions and may intentionally want to break the model. This can be achieved by varying the weights in the neural network (e.g. by adding random numbers to the weights) or by training the model on inadequate data. Such articles investigate such cases. They show how such a malicious update can degrade the overall accuracy of the model. They also consider how to "spoil" the model, to reduce accuracy as much as possible. There are also analyses from the other side – how to defend against such attacks.

Such articles describe various ways to counter malicious attacks. This can be achieved by checking such an update against random data. Another method presented is to check how much the model parameters have changed after the client update. In the case of neural networks, it is possible to check how much the network weights differ in the model from before the update and in the model after the update.

(iv) The various architectures used in federated learning. Such articles show different architectures depending on the needs (e.g. for edge computing). They also show how using a specific type of architecture can achieve better scalability. They describe the benefits and advantages of the proposed design.

About 20 of the articles had realistic simulations using distributed hardware. One of the articles describe how to run FL on a Raspberry Pi. The authors used different datasets for their tests – Speech Commands dataset and Electrocardiogram dataset. By using real devices, they were able to investigate aspects such as communication cost, computational overhead, power consumption and memory usage. One of the conclusions is that not all models can be trained on this type of device due to resource constraints.

Among the analysed literature, there was also a second example where a Raspberry Pi was used to test federated learning for edge computing. Edge computing is an architecture of distributed IT resources, in which data is processed at the edge of the network, as close as possible to the source of origin. Here, special algorithms adapted to fog computing were examined. Thanks to the use of the Raspberry Pi, the authors were able to examine such elements as the cost of communication or the load on the device's resources.

The Raspberry Pi has also been used to test deep neural networks. These types of networks require a relatively large amount of computing power, which can be a problem for smaller IoT devices. Thanks to the tests on the Raspberry Pi, it was checked how such learning can look like on IoT devices.

Another platform, where real devices were used is PerFit. It is used to study human activity. Mobile phones were used to test it, more specifically Samsung Galaxy S3. Thanks to this approach, it was shown that the platform is promising when it comes to using it on IoT devices.

There are also many articles on how to increase accuracy – around 100 out of 250. These articles are based on "internal simulations". By this, it means running federated learning on one machine only, by simulating that more clients are involved in the learning process. This approach allows one to check the accuracy of such a network, but it also has disadvantages. One of them is that it does not allow one to take into account the communication costs. Another feature that is impossible to check is the aspects related to the device. For example, it is impossible to check battery consumption and whether the specific device has sufficient computing power and will cope with training.

Most often, these articles describe how to increase accuracy in a specific situation: (i) for a specific type of network. One example is the articles for generative adversarial networks (GAN). Another type of analysed network is the Convolutional neural network (CNN). Such articles describe which network parameters work best in the federation approach. (ii) In a specific field (e.g., medicine). Such articles discuss a specific problem, e.g. cancer recognition. It is checked how the federated learning approach behaves in comparison to the classical approach and whether the difference in the quality of the achieved solution is large. (iii) In a specific situation (e.g., training IoT devices or training with non-IID data). In this case, for example, situations, where data is not evenly distributed between customers, are investigated.

Common algorithms for federated learning are also tested. Such articles, for example, consider algorithms used to combine the models. Various approaches are also analysed in the context of their impact on the accuracy of the model. For example, the synchronous approach – where the learning rounds are performed successively, is compared with the asynchronous approach – where the learning rounds are performed at the same time.

Customer scheduling articles are yet another category. For example, a grouping of clients is checked, it affects the effectiveness of learning. It is checked whether the appropriate selection of the client, due to the distribution of classes in his set, can improve the model.

In summary it should be stressed that of the 250 early FL-related publications, nearly 80% relied on some form of ‘’internal simulation”. Moreover, the use of FL platforms was negligible. Of the articles read, only a few actually present solutions for a distributed environment. The remaining articles only described the simulation of a distributed environment. Besides, there are few solutions where someone proposes their own platform that allows one to easily test own solutions.

We now have a sense of where the research is headed and what is lacking thanks to a comprehensive review of the literature. It is easy to see that most of the publications present research on improving the accuracy of learning in the federated learning approach. However, there are very few articles presenting real solutions (tested on a larger number of real users). Based on the analysis presented, it can be seen that there is no platform that would facilitate testing with the help of “internal simulations” as well as a larger number of real devices.

1.1.1. Federated learning foundations

Keeping in mid the birds-eye view of federated learning, as seen through content of 250 publications, let us now delve deeper into selected examples of key and/or exemplary research papers. First, let's start with one of the first articles that dealt with federated learning. As a matter of fact, it can be seen as the paper "where it all started". It was published on the Google blog [6]. While it was mentioned in the introduction, it is significant and will be explored in greater depth here. The authors describe the basic assumptions of FL. They report that they are working on implementing FL to their products. One of them is the *GBoard* – a system for suggesting words to users typing on the phone's keyboard. They also describe the problems that they had to overcome when creating their system. This results from the fact that, unlike the classic approach, data is in multiple locations. This has two main consequences.

- The cost of communication
- The need to develop a new machine learning algorithm. Traditional machine learning algorithm will not work in this case. The algorithm must take into account that each user trains his own and then it has to be combined.

The following Sections describe how to deal with these two problems

Dealing with communication costs

The article [41] explains how the first problem was solved. Specifically, the authors focus on solving the problem related to communication between devices and the global server. The basic FL implementation is that after an iteration, each device sends an update to the server. This may result in a bottleneck. This is because each device has a different upload speed and the data upload speed is much slower than the download speed. Moreover, unless devices are highly heterogeneous, their completion of work is likely to happen in "almost the same time". One solution is to change the way the model is sent.

In order to explain how to change sending the model, let us start with the way of its description how model can be described. In the basic approach, each neural network layer of the model is described by the matrix $H \in \mathbb{R}^{d_1 \times d_2}$. The authors propose to write the matrix as a product of two other matrices: $H = AB$, where $A \in \mathbb{R}^{d_1 \times k}$ and $B \in \mathbb{R}^{k \times d_2}$. The A matrix is randomly generated from the obtained seed. Thus, for the server to receive the update, only B, which is the smaller size, needs to be sent. Another method presented is to convert matrix H to the vector

h containing each element of H, and then calculate the new $\overline{h}$ vector as follows:

$$\overline{h_j} = \begin{cases} h_{max} & \text{with probability } \frac{h_j - h_{min}}{h_{max} - h_{min}} \\ h_{min} & \text{with probability } \frac{h_{max} - h_j}{h_{max} - h_{min}} \end{cases}$$

Assuming the vector is large, the above proposal reduces the size of the data by 32 times, compared to writing a single matrix cell, on a 4-bit float.

Algorithm to handle multiple clients

To solve second problems, a special algorithm [47] for deep machine learning was used.

Now, let us look into the details. The algorithms used by FL should take into account that the data is unbalanced. Additionally, they should reduce the communication overhead as much as possible. For this purpose, the authors propose to reduce the number of iterations. To achieve this, they proposed two solutions. One is to increase parallelism, the other is to increase the client-side computation in each iteration. The authors also note that deep learning applications rely almost exclusively on the stochastic gradient descent (SGD) method for optimisation. The standard use of SGD, results in the need to carry out many iterations (even despite the use of additional techniques to improve the solution). This is acceptable in the case of DML. However, increasing the number of iterations involves the need to exchange data between the client and the server, which should be avoided due to the communication problem discussed above.

To better understand the operation of the algorithm, let's start with its description. The machine learning algorithm should find such weights that the error is as low as possible. This sentence can be described as a formula:

$$min_{w \in \mathbb{R}^d} f(w) \quad where \quad f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

For a machine learning task, the function will be $f_i(w) = l(x_i, y_i, w)$. Assuming the number of clients is K and each client has P_k data items and $n_k = |P_k|$, we can convert the above equation to the form:

$$f(w) = \sum_{K=1}^{K} \frac{n_k}{n} F_k(w) \quad where \quad F_k(w) = \sum_{i \in P_k} f_i(w)$$

As the basic algorithm to compare the proposed solutions, the algorithm described in [27] was used, which is a popular synchronous algorithm using SGD for distributed learning. In a typical implementation, the weights update looks like this:

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$$

Each client performs one iteration of the algorithm and then sends a weight shift vector to the server. The server then averages the results from all clients that participated in the iteration and updates the model. Such an algorithm will be referred to as FederatedSGD (FedSGD for short).

The client can additionally repeat the following operation locally:

$$w^k \leftarrow w^k - \eta \nabla K_k(w^k)$$

Such an algorithm will be referred to as FederatedAveraging (FedAvg for short). Both of these algorithms are properly parameterised. The C parameter represents the fraction of clients that is taken for each iteration. The E parameter is the number of epochs executed by the client in each iteration. B is the size of the batch.

Below is the FedAvg algorithm pseudocode. For $B = \infty$ and $E = 1$ the algorithm works the same as FedSGD:

Algorithm 1 FedAvg algorithm

```
function SERVER EXECUTES
    initialize $w_0$
    for round t = 1, 2, ... do
        $m \leftarrow max(C \cdot K, 1)$
        $S_t \leftarrow$ (random set of m clients)
        for all client $k$ in $S_t$ do parallel
            $w_t^k + 1 \leftarrow ClientUpdate(k, w_t)$
        $w_{t+1} \leftarrow \sum_{K=1}^{K} \frac{n_k}{n} w_{t+1}^k$
function CLIENTUPDATE(k, w)
    $\mathbb{B} \leftarrow$ (split $P_k$ into batches of size $B$)
    for epoch i = 1, 2, ..., E do
        for all $b$ in $\mathbb{B}$ do
            $w \leftarrow w - \eta \nabla l(w, b)$
```

An important element that was noticed in the study [47] is that initialising two models with the same initial value gave much better results than initialising them with different values.

It is also worth adding that to test the solution, the MNIST database (containing handwritten digits) and CIFAR10 (containing photos of animals) were used, and the individual images were randomly divided between clients. The results show that the FedAvg algorithm produces very

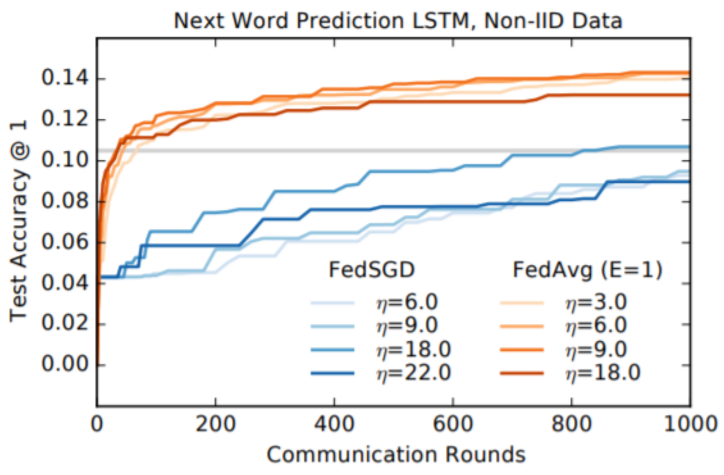


Figure 1.1: Algorithm comparison

Source: H. Brendan McMahan et al.,"Communication-Efficient Learning of Deep Networks from Decentralised Data", [47]

satisfactory results. Its operation was tested also on various architectures (convolutional networks, LSTM networks, multi-layer perceptron, and others). Additionally, the FedAvg algorithm has been shown to perform better than the FedSGD algorithm.

The Figure 1.1 in the article shows the accuracy achieved by *FedSGD* and *FedAvg* algorithms for a different number of iterations for the next word suggestion problem. Additionally, the results for the CIFAR10 database were examined. The FedSGD algorithm achieved an accuracy of 82% after 6,600 epochs, while the FedAvg algorithm after 630 epochs.

Regarding the technical details, it is worth adding that in this scenario on-device learning takes place using the *TensorFlow* library. The article does not specify exactly how the learning on the device side works.

To summarise, it is also worth noting that in order to conduct the experimental parts of the research described above, one has to be Google and have access to all Android-based mobile phones. On the other hand, when one is trying to address research questions, this is done by implementing a private FL simulation environment.

1.1.2. Handling imbalanced distributed data

Unlike the common training dataset, the data distribution of the federated system is imbalanced, which will introduce biases in the model training and cause a decrease in the accuracy of federated learning applications. A few articles are describing how to handle this problem.

In the article [28] discussed is *Astraea* framework. Article describes how to counter the imbalanced data problem. Authors mathematically prove that the imbalance of distributed training data can lead to a decrease in the accuracy of FL applications. Based on this conclusion, they designed the *Astraea* framework, the goal of which is to relieve the global imbalance and local imbalance of clients' data and recover the accuracy. They proposed a Mediator, which is responsible for rescheduling the training processing of the kinds of clients. Rescheduling is based on local data distribution information sent by a client to the server. Another approach they proposed to increase accuracy is data augmentation and down sampling. The discussed framework is not open source and there are no more details about their architecture and possibility to use in different scenarios.

The article [56] describes a solution that does well for non-convex problems with unbalanced data. The algorithm is compared with the popular solution presented by Google (FedAVG) described in the 1.1.1 Section. The authors note that FedAVG only performs well under certain conditions, and this may be because computing the gradient locally is not always a good solution. The authors created the FedPD algorithm based on the Primal-Dual method. Experiments have shown that the algorithm learns much faster than FedAVG (60% accuracy was achieved about 10 times faster).

1.1.3. Attacks on FL systems

Let us now look into another research area. In [46] address one of known FL problems – malicious clients updates. Aggregating data from such clients in harming the global model, e.g. by lowering the global model accuracy. The three types of attacks are mentioned: sign-flipping attack [44], additive noise attack [44], and backdoor attack [22]. The Anomaly Detection Model algorithm [46] uses the reference data to test local models in order to find abnormal, attacking clients. This method can also detect non-intentional erroneous updates.

In [49] the malicious updates problem is considered from a different angle. The article focuses on three types of attacks: direct attack, indirect attack and a combination of the two. A direct attack is when a device that can participate in training, sends erroneous data. An indirect attack is when an attacker accesses a device on the network and then sends erroneous data. The attack consists of selecting the network coefficients in such a way as maximise harm to the learning process. Considered text discusses a method of creating such update. The conducted research showed that the mean error increased over 4 times from 5% to 23.88%. The amount of data that has been infected was 20%.

1.1.4. Privacy preserving

Ensuring privacy protection is very important when building a federated learning system. In [39], the authors state that other solutions may use users data. To prevent this, the *Multi-Party Computation (MPC)* solution was used to ensure the security of the processed data. MPC is a set of cryptographic techniques enabling confidential computation on sensitive data. The article compares two models – *Peer-to-Peer* and a two-phase model. The Peer-to-Peer model is based on the exchange of data between every two learning participants. As the authors note, this approach is demanding due to the computational complexity. Therefore, it is not easily scalable. To solve this problem, a two-phase model was introduced. The first phase consists of selecting the clients who will participate in the data exchange in the second phase. Both models were compared in terms of the number and size of messages sent. The two-phase model produces much better results, and the difference increases as the number of participants increases.

1.1.5. Federated learning and generative adversarial networks

Occasionally, federated learning uses atypical neural networks. One such networks is generative adversarial networks (GANs). Such a combination is described in the article [29]. Publication discusses the use of FL using generative adversarial networks (GANs). GAN networks are used to generate various types of elements. They consist of two main components: a generator and a discriminator. The generator learns how to create elements as close to the real thing as possible. The discriminator, on the other hand, learns to distinguish between true and false elements. The authors present a system in which each client has a generator and discriminator module. The client additionally updates the global generator and discriminator located on the server. The authors state that due to such a system characteristic, consisting of two modules, the problem of synchronization between each client is much greater than with the FedAvg algorithm. The authors compare four methods of synchronization: synchronization of the generator, discriminator, both elements and the lack of synchronization. For real-world cases where communication costs are very high, the authors suggest generator-only synchronization. In other cases, they suggest using synchronization of both generator and discriminator.

After summarising the previous Sections, the same meta-level conclusions can be obtained. To pursue the outlined research lines one needs to implement private FL environment. This is because the existing FL platforms are not flexible enough.

We will now move on to the next Sections, where we can explore the implementation of federated learning in different fields, now that we have addressed the most relevant articles on applied solutions in federated learning.

1.2. Application areas

1.2.1. A solution in medicine

Let us start from FL in IoT in health-care. Authors of [37] describe a solution for classifying signals from an electroencephalograph (EEG). They point out that due to the aspect of personal data protection, there is a lack of sufficiently large datasets to be classified in this area. There are many small datasets that privacy policy cannot combine to create one large dataset. An algorithm using the method of covariance-based on neural networks has been proposed. The article describes how signals are processed and signals from the device so that they can constitute an input to the neural network. The authors used the averaging method, therefore the weights from customers are averaged, and then the model is updated based on them. The achieved results are satisfactory compared to other algorithms. The proposed algorithm will achieve an efficiency of 63%, while the best of the compared is 66%.

The authors of [43] describe their solution for processing medical data. The solution is tested using the MIMIC-III database. The authors do not describe in detail the algorithms used, but only the components that are part of the solution. The client consists of three parts: the first for teaching, the second for communicating with the servers, and the third for performance testing. The results were tested for different cases:

1. Balanced Data
2. Unbalanced data
3. Each customer only had data for one category
4. Category 2 and 3 combinations

For the given categories, the results of the F1-score measure were as follows: 94.6% for 1, 92.1% for 2, 90.5% for 3, and 89.1% for 4.

NVIDIA Clara [15] is another application that delivers AI to customers. Healthcare giants around the world aim at developing a personalised AI for their doctors, patients, and facilities, where medical data, applications, and devices are on the rise and patient privacy must be preserved.

NVIDIA offers an edge computing platform, which uses deep learning for radiology. Participating hospitals label their patient data and train the global model. Platform preserves privacy by sharing only partial model weights. There are already partners (mostly in UK and US) who use the described solution.

The last example is an application for drug discovery [3]. Major pharmaceutical companies agreed to build the common platform in partnership with NVIDIA, Owkin, and others. Authors claim that fewer than 12% of all drugs entering clinical trials end up in pharmacies and it takes at least 10 years for medicines to complete the journey from discovery to the marketplace. FL can accelerate this process. The authors also mentioned that they reached the first year objective (out of 4).

It should be noted that the NVIDIA platform works very well in the described situations, but the modifications (needed to conduct further research) are hard to implement and require a lot of work.

1.2.2. Usage in IoT

Federated learning is also commonly used in conjunction with IoT devices. Here, let us mention [54], the framework that enables FL to be used in internet of things (IoT) devices in the healthcare field. IoT is widely used in medical applications for remote health monitoring, helping the elderly, or monitoring chronic diseases. The authors state in the article that the solution presented by Google will not work in the problem they are considering. This is because IoT devices that are used in medicine have much lower computing power and battery capacity compared to mobile phones. Also, the link speed is lower compared to mobile devices. Due to these limitations, the authors proposed a different solution. They used a shallow subnet that is trained on the device and a deep neural network that is trained in the cloud. In their research, they used the presented framework for arrhythmia prediction. The tests showed slightly worse accuracy results about 2%. The second examined aspect was the limitation of network traffic, where savings of over 90% were achieved compared to the compared algorithms.

The solution uses an approach called *SplitNN*. It is described in [34]. A deep neural network can be considered as a F function, consisting of successive layers $L_0, \ldots, L_N$. For some inputs ($data$), the $F(data)$ function can be specified as $F(data) = L_N(L_{N-1} \ldots (L_0(data)))$. Let $G_{loss}(output, label)$ be a gradient function for the last layer. The $L_i^T(gradient)$ function will represent the backward propagation process, while the $F^T(gradient) = L_1^T(L_2^T \ldots (L_N^T(data)))$ will mean the backward propagation process for the entire neural network. The training process is that the client propagates the network forward for the initial n layers. Then the result for n-th layer is sent to the server with the correct category (label). The server propagates forward across layers, calculates a gradient-based on the result of the last layer and category, and then propagates backward. After making these calculations, the server calculates the gradient of n-th layer and sends it back to the client. A simplified algorithm for one client is depicted below:

Algorithm 2 SplitNN

```
$F_a \leftarrow L_0, \ldots, L_n$
$F_b \leftarrow L_n + 1, \ldots, L_N$
Client and server initialize wages of $F_a$ and $F_b$ respectively using same seed
**for all** *data* in *client* **do**
    $X \leftarrow F_a(data)$
    $Send((X, label), Server)$
    $output \leftarrow F_b(X)$ // *Server side*
    $gradient \leftarrow G_{loss}(output, label)$ // *Server side*
    $F_b', gradient' \leftarrow F_b^T(gradient)$ // *Server side*
    $Send(gradient', Client)$ // *Server side*
    $F_a' \leftarrow F_a^T(gradient')$
```

The article also shows that this way of computing is equivalent to the traditional treatment of the network as a whole.

User data is only partially protected in the described algorithm. All the time categories are sent to the server, which may also be sensitive data. To avoid this, we can slightly modify the solution so that the server only calculates the "middle" part of the network. The client's network layers are $L_0, \ldots, L_n$ and $L_m + 1, \ldots, L_N$, and the server is $L_n + 1, \ldots, L_m$. Then the algorithm scheme looks like this:

1. The client performs forward propagation computation for the $L_0, \ldots, L_n$ layers
2. The client sends the result for the n tier to the server
3. The server performs forward propagation calculations for the $L_n + 1, \ldots, L_m$ layers
4. The server sends the m tier result to the client
5. Client performs forward propagation computation for $L_m + 1, \ldots, L_N$ layers
6. Client calculates gradient for end layer o propagates backwards for layers $L_N, \ldots, L_m + 1$
7. The server is propagating backwards for the $L_m, \ldots, L_n + 1$ layers

8. The client propagates backwards for the $L_n, \ldots, L_0$ layers

As we can see, this version requires much more communication between the server and the client. The results were compared for different data and network architectures. It was also compared with other algorithms, including [47] described in the 1.1.1 Section. In the described cases, the algorithm gave better results than other algorithms.

Another example of using FL in IoT is described in [53]. Authors advocate a personalised federated learning framework in a cloud-edge architecture for IoT devices. The authors diagnose many problems in the field of machine learning using IoT devices. The first group of limitations is associated with device heterogeneity. The first problem is that devices differ in the hardware configuration (CPU, memory, network, and battery). This might affect learning duration, communication costs, and period of availability. Another category is statistical heterogeneity and it simply means that date is non-IID distributed. For example, in healthcare applications, the users' activity data may differ due to users' characteristics.

To tackle the heterogeneity problems authors propose their federated learning framework. The system adopts a cloud-edge architecture, which brings edge computing power in the proximity of IoT devices. Each device can send a computation task to the edge device (i.e., edge gateway at home, edge server at office). If the edge is trustworthy, the device can send its data and model to this edge device. Otherwise, the device sends only part of the model to the edge device.

1.2.3. Federated learning and blockchain

While this example was stated in the introduction, it is significant and will be explored in greater depth here. An interesting connection between FL and blockchain is described in the article [40]. It is compared with the solution proposed by Google and described in the 1.1.1 Section. The authors try to solve two main problems of Google's solution. The first is the server's susceptibility to failures, and the second is the lack of rewarding clients for the work put into training the model. The authors note that clients with larger datasets may be less dependent on collaborative learning. The operation of BlockFL is summarised as follows:

- The client trains the model locally
- The client sends the model to the associated miner with its local compute time.
- Miners broadcast the obtained local model updates. At the same time, the miners verify the received local model updates from their associated devices or the other miners. The verified local model updates are recorded in the miner's candidate block.

- Each minor begins computing Proof-of-Work (PoW) until it finishes or gets a different block.
- Miner that first computes PoW sends the block to other miners.
- Clients get the generated block from the associated miners.
- Customers update their model based on the block they receive.

The epochs are executed until the appropriate precision $|w^L - w^{L-1}| \leqslant \varepsilon$ is achieved, where w is the weights of the neural network and L is the iteration number. The blockchain network also provides rewards for data samples to the devices and for the verification process to the miners. The data reward of the device is received from its associated miner, and its amount is proportional to the data sample size. Because of the fact, those devices may inflate their samples, miners may verify truthful local updates before storing them by comparing the sample size with its corresponding computation time. This can be guaranteed in practice by Intel's software guard extensions, allowing applications to be operated within a protected environment, which is utilised in blockchain technologies.

The conducted research shows that the proposed model learns faster than the model proposed by Google and achieves comparable effectiveness (95% and 98% for different parameters).

1.2.4. Solutions for keyword spotting

Federated learning is also used in the field of mobile phones. To refine the model, it uses data obtained on the computer. An interesting solution was proposed by Google, is system for keyword spotting [35]. The model uses the encoder-decoder architecture. A modified FedAVG algorithm was used, which used Nesterov accelerated gradients for the server-side updates. Various methods of server-side optimisation were also compared: Adam, Yogi, and LAMB. The reported results are presented in the Table 1.1. The columns show false positive (FP) and false negative (FN) ratio. We can see that the FP ratio is about the same (between 0.19 % and 0.21 %) and it is very small. The bigger difference is visible in the case of the FN ratio. Here, the FedYog algorithm without any modifications gave the best results.

1.2.5. Federated Learning in computer vision

Federated learning can be applied to image processing, realised by the FedVision application [4], which aims at recognising objects in images. While this example was stated in the introduction, it is interesting and will be explored in greater depth here. In this context, the main advantage of FL is the reduction of data transmission. Here, data (large images) can be stored locally, while

Table 1.1: Results comparison

Optimisation	FP (%)	FN(%)
FedAvg	0.21	8.76
FedAvg + NAG	0.21	4.09
FedAdam	0.19	1.95
FedAdam + NAG	0.21	1.68
FedYogi	0.19	1.39
FedYogi + NAG	0.20	2.11

it is only a model that is actually communicated. It is enough for the images to be tagged using a common set of labels. It has been used by three large-scale corporate customers. The platform has helped the customers significantly improve their operational efficiency and reduce their costs while eliminating the need to transmit sensitive data around.

1.2.6. Federated Learning in the banking sector

Another example of applying FL into a real application is described in WeBank [7], the first digital bank established in China that has developed its federated learning model for credit rating. Here, the authors of the platform support the claim that companies need to cooperate, but they can't share their data. FL is a solution in that situation. Their model is restricted to measuring the credit risk of small and micro-enterprises. So far, the model has halved the number of defaults among WeBank's loans to these customers.

The analysis allowed us to see where federated learning is used. This made it possible to better specify the requirements for building the platform.

1.3. Platforms

We previously discussed the lack of appropriate frameworks for testing Federated learning cases and conducting research. We'll take a closer look at this issue in this segment. The four most common platforms will be discussed and their benefits and drawbacks will be addressed. We'll also look at how we can make it more useful for researchers.

1.3.1. TensorFlow Federated

Let's start with TensorFlow Federated (TFF, current version is 0.18.0, see [19]). It is an open source platform for decentralised ML and other computations [20]. According to the documentation, the main goal of the platform is to conduct experiments related to federated learning, without having to go deeper into how the framework works. It also allows testing the implemented algorithms on the provided datasets and models. The platform consists of 3 key parts – (i) Classes – containing classes and helper functions that help wrap up the current models so that they can be used with federated learning. (ii) Federated Computation Builders – contains support functions for creating a federated learning process. (iii) Datasets – contains databases that allow testing the created solution.

It is worth adding that the databases are adapted to federated learning and samples are divided among clients so that simulations can be carried out easily. The fundamental architectural assumption, authors made in TFF, is that the model code must be serializable as a TensorFlow graph. This is because the script is runnable, even if the client does not have a Python runtime. In addition to the high-level interfaces, lower-level interfaces are provided. Thera are a part of Federated Core which is a collection of low-level interfaces for implementing federated learning. It allows developers to independently write the functions needed to run federated learning. To allow this many structures were provided, such as data types, sequences, tuples, and Function types. All this was made to allow to "pack" federated learning code and allow to run on different machine. TensorFlow Federated also offers easy setup with Google Cloud Platform. Unfortunately, enabling federated learning to be launched via remote communication is still in the early stages of development. Now the framework is mainly used for local simulations.

With TFF, it's very easy to test basic scenarios. However, making extensive changes is problematic. Many changes are needed to be done to test a more complex scenario that deviates from the traditional – simple approach. For instance, let us consider the case where we want to test communication costs. With TFF, we can easily simulate cases locally, but testing how network protocol communication works is problematic. It is necessary to develop a solution that allows for remote communication, but this may prove difficult.

1.3.2. PySyft

PySyft, the second framework, is a Python library (current version is 0.3.0, see [48]) for deep learning [17]. The framework proposes mechanisms such as Differentially Private and Multiparty

Computation, as well as various variants that can be used through an intuitive interface. The main goal was to provide privacy preserving techniques in machine learning.

Authors of the platform created an abstraction called *SyftTensor* to allow send tensors as a chain of workers. The head of this chain is always a PyTorch tensor and any transformations are saved as a child node. There are two main subclasses of SyftTensor. First is the *LocalTensor*, which can be understood as a sort of vector. The second is called *PointerTensor*, which is a tensor sent to a remote worker. When the tensor is sent user losses the opportunity to read its data and has only the pointer, which specifies who owns the data.

There are two main options to run federated learning using PySyft. First is virtual, which means that all process is running on the single machine. To allow this Virtual Workers are used. Virtual Workers are running on the same machine and don't use any communication protocol to extend data. This option was created to allow debugging and test federated learning algorithms. The second option are Network workers. They have two implementations – first using network sockets and second using Web Sockets. Very interesting is the time comparison of these approaches. Basic approach – without any workers lasts 0.22s, with virtual worker lasts 10.1s, with virtual worker and Differentially Private mechanism lasts 15.3s and with Web Socket worker lasts 14.6. The most important information from this test is that the federated learning approach consumes a lot of time compared to the traditional approach.

Authors of the framework implemented differential privacy which allows to train networks using that mechanism. The authors also built a tool to split the database over workers in order to run tests of federated learning algorithms. The disadvantage of this framework, as with TFF, is that is not flexible enough to run more complicated tests. For example, consider the case where we want to test the operation of FL on unusual data (e.g. non-IID). Consider the example of digit recognition. We want every customer to have twice as many cases of the digit 1 as the others. This will be difficult to achieve and requires special, difficult data preparation.

1.3.3. FATE

The third framework, is FATE (current version 1.6.0, see [8]). FATE is a platform focusing on enabling federated learning in the industrial sector – between different companies. The platform consists of 6 main modules – EggRoll, FederatedML, FATE-Flow, FATE-Serving, FATE-Board, and KubeFATE. Eggroll is responsible for data storage and communication between nodes with focusing on a large-scale. FederatedML – offers the algorithms needed for federated learning. It offers many machine learning models, including neural networks, decision trees, and logistic regression. All these algorithms are properly secured, ensuring the privacy of customer data. To ensure this, it offers a mechanism such as homomorphic encryption. Another element is FATE-Flow. It is responsible for defining the process flow. Such a course may consist of various elements, such as data preprocessing, training federated model, model management, and model publishing. FATE-Serving is responsible for make reasoning based on a model. The FATE-Board is responsible for data visualisation. It allows to follow the learning process and uses graphs to visualise its effects. The last element is KubeFATE. It is responsible for deploying the application using Docker and Kubernetes. Thanks to this, it is possible to run the platform in various environments. All in all, it is a comprehensive tool that offers a lot of possibilities. Detailed documentation is a big plus. However, there are also disadvantages behind this. Due to its size, all modifications are difficult to do, and additionally, we need to know the implementation details to make it possible. For instance, consider a case where we want to select clients in unusual way. For example, due to the similar computing power, mobile phones should be trained together with other phones, and computers with other computers. Due to the high complexity of the FATE platform, this will be difficult to achieve. This requires the user to thoroughly understand how the platform works and make changes to make them compatible with the entire platform.

1.3.4. Flower

Finally, Flower [21] is another FL platform, currently under development. The authors claim that their solution is scalable it can handle a large number of clients. It is designed to handle a large number of clients (10,000 or more) and is platform-independent. It provides several FL components such as connection management on both client and server, FL loop, FL algorithm, and client-side training, evaluation to allow researchers quickly experiment. Moreover, platform offers already implemented federated optimisation methods to allow researchers to easily test their machine learning scenario. For example, there are many built-in averaging algorithms like FedAvg, FedProx, Qffedavg, and many others. It supports many existing Machine Learning frameworks including Keras, TensorFlow, MXNet, PyTorch, and even raw NumPy. It allows running FL on Android, Nvidia Jetson, macOS, and the Raspberry Pi. It also allows running tests on heterogeneous environments so it can be check if the solution is suitable for real-life use. Authors claim that the platform is easy to extend and needs low effort to make own changes. According to its creators, it follows the following core principles: (A) customizability – it allows individual configuration depending on the needs; (B) extensibility – for the creation of modern state-of-the-art architectures, elements can be expanded and overridden. The main idea of the platform is that the client can be independent of the environment on which it is run. This means that the client can use different operating systems and programming languages. Flower also provides a variety of built-in datasets allowing experiments to be carried out easily. Authors also provided Flower Protocol – a low level way to integrate with Flower components. It requires a client to handle specified messages and react according to assumptions. Flower Protocol has 2 types of messages. First are *Instructions* – the server sends a set of "commands" what should the client do with their local date and client response with the result. Second types are *Connectivity* messages. These types of messages are used to decide if the client should participate in round or if should reconnect later. In this case, the biggest disadvantage is the difficulty of changing the architecture. Let's imagine a situation that we want to connect the platform to the blockchain system - as described in Section 1.2.3. The architecture that Flower uses is simple and will suffice in most cases. Unfortunately, it is related to the difficulty of combining it with other systems.

1.4. System design of existing solutions

The architecture of FL systems will be discussed in this Section. They'll act as the foundation for our own platform. The first article describes the solution introduced by Google, which is the basis in this field. The second one, however, describes the architecture used in a real solution to cooperate with industrial machines.

1.4.1. Architecture proposed by Google

The solution proposed by Google is the foundation of all federated learning considerations, therefore their approach to architecture will be analysed first. Therefore, our solution will be based on the ideas presented in this Section. In [23], Google's researchers describe a system for Federated Learning in the domain of mobile devices, based on TensorFlow. This article has already been discussed before, but now we will focus on the aspects related to the architecture used by the authors. They describe the resulting high-level design and some challenges. They also state that their work is in the early stage of development and there are many problems that they will have to face. Authors focus on support for synchronous rounds due to, among others, security reasons. This is caused by the fact that many privacy supporting algorithms require a synchronous approach (e.g. Secure Aggregation algorithm [24]).

To begin, devices notify a server that they are willing to start a machine learning computation. From the potential tens of thousands of devices announcing availability to the server during a certain time window, the server selects a subset of typically a few hundred who are invited to work on a specific task. Here, note that existence of such large number of devices that are to participate in training is related to the fact that it is Google and Android-based cell phones that are to connect to the FL process. Otherwise this step may not be applicable. The server tells the information to the selected device about computation to run. Once a round is established, the server next sends to each participant the current global model parameters and any other necessary information. Each participant then performs a local computation based on the local dataset, and sends an update back to the server. The system enables training deep neural networks, using TensorFlow. If enough devices report in time, the round will be successfully completed and the server will update its global model, otherwise, the round is abandoned. Federated Averaging algorithm (for details see Section 1.1.1) combines weights of the model. This algorithm ensures that, on a global level, individual updates from devices are uninspectable. A special security mechanism ensures that individual changes are not stored in the cloud. The solution is checked by Google for word prediction when typing. The entire process then repeats until the desired effect is done.

Selection is made based on connection stability and information on whether the device is charging. Note, that this aspect is, again, specific to the application area, device characteristics, and device availability. The round duration may differ and might depend on factors like device goal count, timeouts, and minimal goal count to run a round.

The authors also introduce a module called *Pace Steering*. It is responsible for servers to be able to handle small numbers of devices as well as large numbers of devices. It is used to ensure that a sufficient number of devices connect to the server simultaneously. This is important both for progressing the learning ratio and ensuring secure aggregation.

A device module is combined of *Example Store*, *App Process* and *FL Runtime*. The *Example Store* is a local database in which local data is stored. *FL Runtime* executes learning using data from Example Store. *App Process* is responsible for setting learning parameters (e.g. how many epochs should be performed).

It is worth adding that the application supports running multiple models training within one service. For instance it is possible that the process of training the model for recognising digits and the model for categorising photos will run at the same time.

The server is designed around the actor model. Each actor handles a stream of messages/events strictly sequentially, leading to a simple programming model. Running multiple instances of actors of the same type allows a natural scaling to a large number of processors/machines.

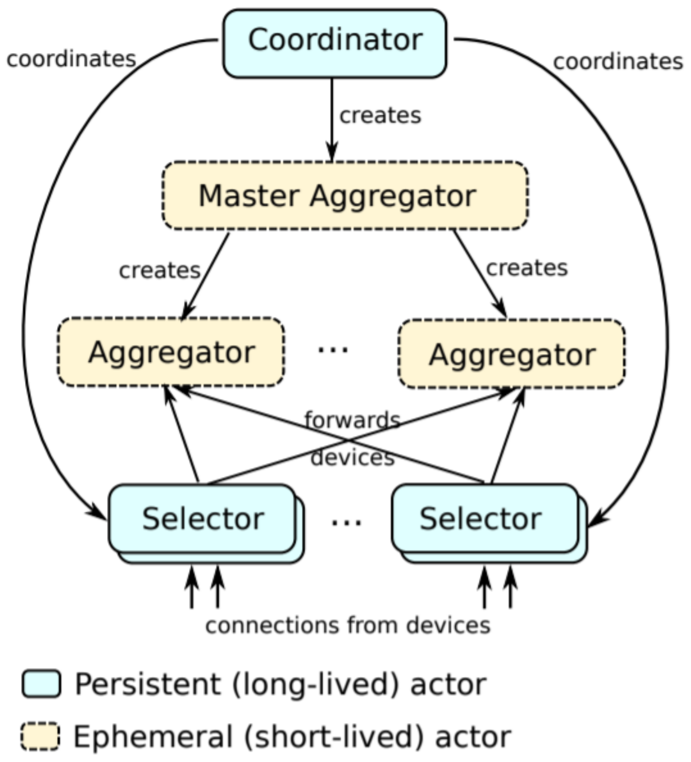


Figure 1.2: Architecture design

Source: Bonawitz et al.,"Towards federated learning at scale: System design", [23]

Now, let us describe the main actors that platform uses. First are Coordinators which are the top-level actors, which enable global synchronisation and advancing rounds in lockstep. Selectors are responsible for accepting and forwarding device connections. Master Aggregators manage the rounds of each FL task. In order to scale with the number of devices and update size, they make dynamic decisions to spawn one or more Aggregators to which work is delegated.

The authors also describe the way they gather device logs and parameters; e.g. how long training lasts and how much memory it uses. This information does not contain any personal details.

1.4.2. Industrial Federated Learning

The article [36] contains another intriguing architectural approach. Proposed *Industrial Federated Learning* is a system that supports knowledge exchange between industrial machines. Compared to the previously discussed article, this focuses more on a low level of system design. It describes modules and their cooperation. The article describes features of the problem in the industry context and presents system design. The proposed architecture is divided into 2 parts – server and client. The client is responsible for registering participating edge devices and passing general information about the organisation. The server stores this information. The server also contains modules responsible for cooperation with the client. It also contains modules for storing information about machine learning models.

1.5. Summary of state-of-othe-art review

This chapter presents the most important aspects of federated learning. These include basic algorithms, issues related to handling unbalance data, and ensuring data protection. The presented uses of federated learning in real life have shown the direction in which research in this field is currently taking. The areas in which federated learning is developing the most are healthcare, IoT devices, and mobile phones. We can see a lot of researches in this direction. At the same time, we noticed a problem in testing such solutions - especially when it comes to remote communication. Another problem encountered was that each time a more complex case had to be tested, the authors had to significantly modify the existing solutions. A review of the most popular platforms for carrying out the FL process only confirmed this thesis. It revealed that they are good, but only for basic applications. They allow to test simple scenarios, but to conduct more complex research, researchers had to modify the available solutions. This motivated us to build our own solution that solves these problems. The basis of our solution will be articles describing the architecture of FL systems. Authors showed how the architecture of the system responding to the user's needs should be built.

2. Development and implementation of the Sunday-FL platform

Across material presented thus far we have argued that use of existing all-in-one platforms is not conducive to researching open FL questions. Moreover, reported analysis of literature, showed that majority of current work involves “homemade FL” implementations. Let us provide sample reasons, why proposing a new platform may be useful in the context of FL research.

Specifically, let us look into research in two areas. First, let us consider the fact that separate devices, participating in FL, may have data distributions different from the global data distribution. In other words, they do not belong to the category of Identically and Independently Distributed (IID) datasets (see, [38]). Here, multiple situations may result in non-IID distributions. (a) Feature distribution skew – individual users with different handwriting. (b) Label distribution skew — mavrud grapes grow only in Bulgaria. (c) Same label, different features — images of village homes vary around the world. (d) Same features, different label -– labels reflecting emotions have regional variation. (e) Quantity skew -– clients hold vastly different amounts of data. Using non-IID datasets can result in significant loss of accuracy. Obviously, globally imbalanced data is another source of accuracy loss.

To mitigate this effect, several techniques were proposed, e.g.: (i) client selection, based on the degree of non-IID data [55], (ii) sharing proxy IID dataset [57], (iii) clustering clients and using multiple models (instead of a single common model [25]). Finally, (iv) self-balancing FL [28], described in more detail in Section 2.1.

Regardless of the approach, exploring it reaches beyond services offered by standard platforms (summarised in Section 1.3). Obviously, each time a private platform could be implemented, but this is leads to waste of effort.

Therefore, we decided to facilitate research in the field of FL. The conducted analysis of the literature led us to the following assumptions:

- The platform should be scalable
- The platform should be runnable on various environments

- It should be easy to make changes, so that different ways of learning can be tested
- It should be possible to run the platform on different machines

2.1. System components

By *client*, we mean a participant who is in charge of training a model on their own data in the subsequent Sections. The *server*, on the other hand, plays a significant role in the FL process' coordination. For example, consider the problem of classifying pictures. Pictures are on phones. In this case, the phones will be a client that will train the model using the pictures they have. The server, on the other hand, will be responsible for coordinating the learning process.

Based on the approach described in 1.4.1 we proposed our way to build the architecture of a federated learning system. It is based on the actor programming model. The design of the system architecture is driven by the (potential) necessity to manage a large number of devices.

Another aspect that we have to take into account is the necessity of enabling system configuration. It means that the client program can be able to run different machine learning scenarios. It should be easy to set which algorithm will be used and with what parameters. Also, more complicated scenarios should be runnable easily (e.g. scenario described in which requires additional "modules" called *Mediators*). To allow this we proposed design where the client can download the proper module according to the machine learning task, in which he would participate in.

Another problem we have to consider is the fact that the device may perform machine learning task in a specific time slot. Sometimes the number of participants may be too small. For example in case when the problem concerns a business issue. If a few companies would like to perform machine learning then might be too small clients to enable machine learning in different time slots. So they have to communicate to decide a time slot in which everyone can participate. Currently, this is a functionality that is not yet implemented. However, this must be taken into account when designing the architecture.

Due to discussed problems architecture depicted in Figure 2.1 was proposed.

The proposed platform consists of the following elements:

Cooperation manager – is responsible for determining the time slot details. Each *Personal actor* sends a possible time slot, in which it can participate in learning. Based on that, the Cooperation manager determine the proper time slot. It is important that the diagram does not

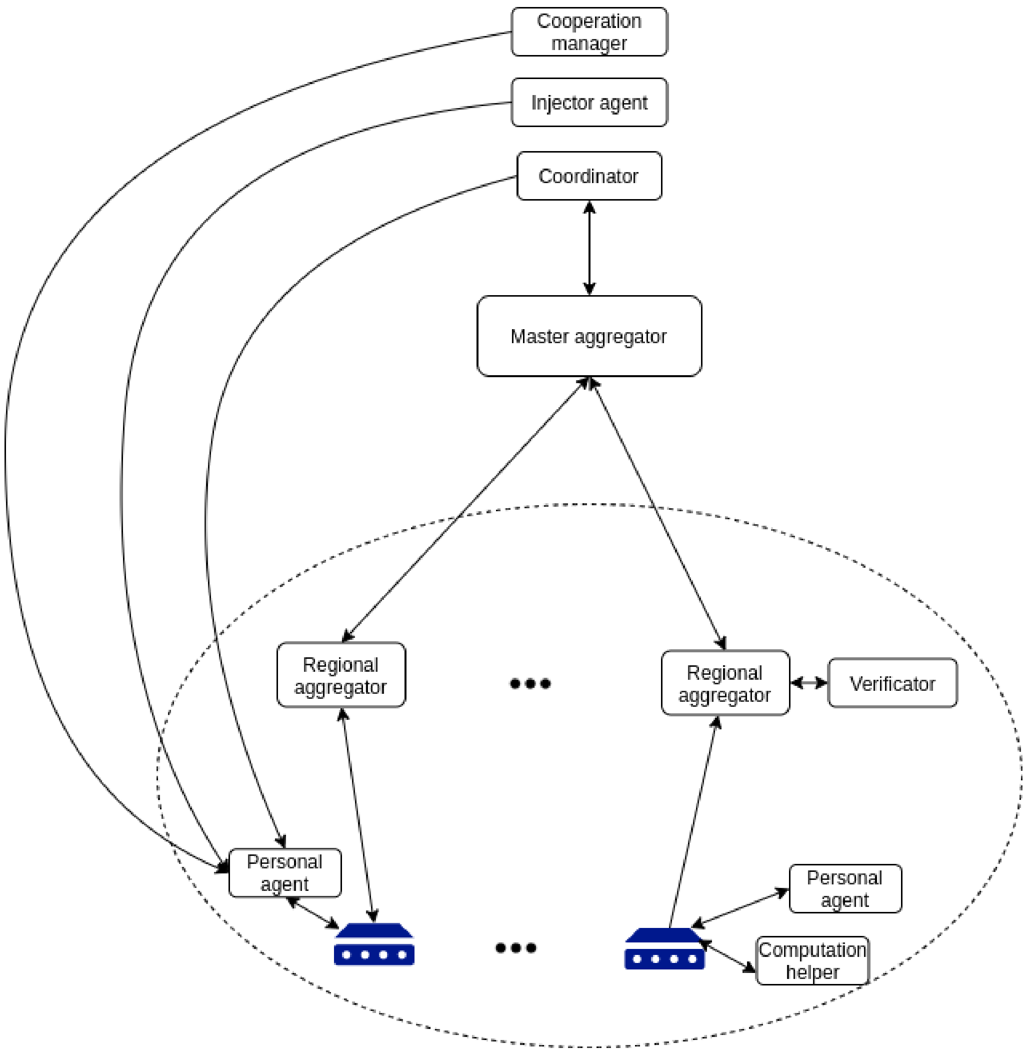


Figure 2.1: System-design

specify how exactly negotiation should look like. It just takes into account the necessity of that module.

Injector – enables client program configuration. Manages a set of modules that are needed for machine learning. A client can download the module necessary to run a specified machine learning task.

Coordinator – is the main actor responsible for synchronization and managing rounds. There may be multiple *Coordinators*, and each one is responsible specific machine learning task. Within specific machine learning task there is one *Coordinator* for for the entire training process. The *Coordinator* spawns Master Aggregator to manage each round.

Master aggregator – manage the rounds of each machine learning task. To scale with the number of devices, they make dynamic decisions to spawn one or more Regional Aggregators, to which work is delegated.

Regional aggregator – short-lived actor, which manages a single round. It is responsible for communication with *Personal actor*. It also reports changes to the *Master aggregator*.

Verificator – this optional actor will be available, in the future, to verify the FL process. In some cases, the client may have bad intentions and intentionally harm the model. In that scenario, Verificator checks if an update may improve the model. If not, the changes are rejected.

Personal actor – actor which represents each device. It participates in negotiations (in the future) with other actors about when to start learning. It is also responsible for handling modules deeded for learning (and downloads module if needed). It also asks *Selector* to join machine learning round.

Computation helper – is planned (in the future) to facilitate the realization of complex workloads. A case for its use is, for instance, when a device is not capable to handle computations itself e.g. IoT devices.

There are several advantages to using the proposed architecture. First of all, one can easily achieve the intended goals of the platform, such as the possibility of time negotiation (will be implemented in the future), verification of the model update, and adding a module that helps in the calculations. Another important factor is that the architecture is easily scalable. The number of Regional Aggregators and Selectors depends on the number of devices and may change dynamically.

2.2. Process flow

First, let us start by describing how the process looks like. Each device knows about the server – it has its address. The server doesn't know about the device until it sends the request that it wants to participate in learning. At the beginning, devices notify the server of their willingness to participate in training. The server collects information about such devices for a specified period of time. Then, the server arranges the training details with the devices. This includes determining when the training is to take place and on what principles – the training model and its various parameters that are to be selected (e.g. neural network). It is a general scheme and the details depend largely on customer needs.

Once this information is established, the learning phase begins. Each device needs to download module needed to run training on their local data. More details about that process are described in Section 2.5.

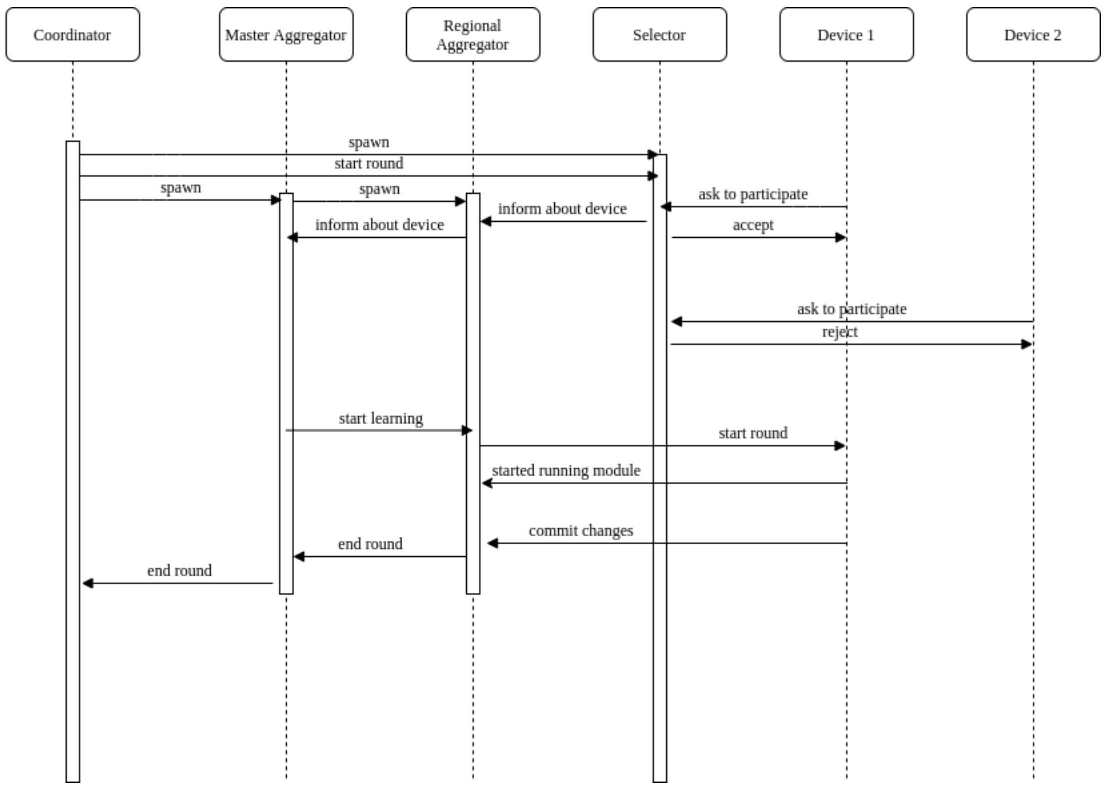


Figure 2.2: Communication diagram

Details of the learning phase are shown on Figure 2.2. Firstly long-lived actors are spawned (Coordinators and Selectors). By long-lived actors we mean actors that live longer than just one round – in that case, they live during all learning process. Then the Coordinator actor sends a message to Selector to start accepting devices to join the learning round. Then Master Aggregator actor and Regional Aggregator actors are spawned as short-lived actors – they live only within one round. Selectors, which received information about starting round, are ready to accept devices to join in learning.

The server is ready to receive devices. Each device should send a message to Selector(device knows the address) about willingness to participate in learning. The message contains a task identification number, which is related to a problem in which the device wants to participate. Each Selector informs the Regional Aggregator about the new device and the Regional Aggregator informs Master Aggregator (which is the parent object). The Selector can accept or decline the request. The device selection process can be more complicated. The Selector can accept devices based on various parameters (e.g. battery level, link speed). At the moment, the server does not have a policy, so it accepts all devices. The architecture, on the other hand, provides for the possibility that the server has the appropriate module responsible for it. The server accepts devices until the appropriate number of devices (established in server configuration) has accumulated. The server will wait for the required number of devices to join the training. If the appropriate number of devices is not reported for a sufficiently long period, the server will reduce this number so that the training can begin. If enough devices have joined, Master Aggregator sends a message to Regional Aggregator about start learning and then each Regional Aggregator sends a message to the device about start learning. After the device received this message, he starts the learning module and sends a message about it to Regional Aggregator. The Regional Aggregator sends the current model to the training device. Each device then trains the model locally on its data. After all, each device should send the model back to the server. When enough devices have sent their changes, the server updates the model. This is done in such a way that it combines the resulting changes and updates the current model based on it. A special module is responsible for updating the model on the server side (details are described in Section 2.5). One round ends in this way. The server runs these rounds sequentially until the required accuracy is achieved.

There are two ways to carry out training – synchronous and asynchronous. The synchronous approach is that the rounds go one way. Asynchronous, on the other hand, so that several rounds are executed at the same time. Both approaches have been used with success, but recent trends are moving towards synchronous learning [33]. Therefore, in this work, the architecture will be such as to enable synchronous learning.

An important ability is also that the devices themselves decide when they want to connect. There may be several strategies for determining this (e.g. in the case of mobile phones if it is connected to charging and has a WiFi connection). Currently, the client does not have a strategy, so it always connects. The architecture, on the other hand, provides for the possibility that the customer downloads the appropriate module responsible for it. This allows them to choose a time that is convenient for them (e.g. Sundays are not involved in any other learning process). Another important thing is that the clients taking part in the round are learning the model asynchronously. This means that in one round the model is trained by the devices in parallel and finally merged at the end of the round.

2.3. Actor programming model

The platform will be based around the actor programming model. The actor programming model was proposed as a way to handle parallel processing in a high-performance network. As a consequence, organisations building high-demand distributed systems face challenges that cannot be fully resolved with the traditional object-oriented programming model, but that can benefit from the actor model.

Using an actor programming model allows one to:

- Easy scaling to a large number of components. Each actor handles its mailbox sequentially which leads to the fact that each actor follows a simple logic. Ease of scaling is a very important element of the platform. This will allow us to easily test cases where many devices are involved in training.

- Actors are lightweight. There might be several million actors per GB of heap memory. This is also a very important feature when it comes to testing using many devices. Additionally, it is important for devices that have limited resources

- Actor instances can be located on the same machine or distributed across multiple geographic regions. This allows to brings actors close to the users to reduce communication problems.

All presented features above are really important in the system because there might be a lot of components – actors – that need to communicate with each other.

2.4. Privacy

As mentioned before, the main feature of federated learning is data privacy. To achieve this, a special algorithm should be used for data aggregation. It should protect users against leakage of their data. Therefore, each user should only send the model update. However, sometimes some information about the data (e.g. class distribution) can be deduced even from updating the model itself, which is very undesirable. Therefore, the update should be included in the model as soon as possible, and the user's update should not be permanently saved. The presented architecture scheme makes it possible. The update can be done at the level of the *Regional* Aggregator. There are already libraries that can safely combine several user models into one. In presented platform, algorithm FedAvg (details in Section 1.1.1) from the *Pysyft* library was used, however, thanks to the flexibility of the solution, any algorithm can be used – own or from another library (e.g. TFF).

2.5. Learning modules

One of the platform's advantages is its adaptability, which helps users to train models in a variety of ways. It should be similar to how the model would be conditioned, which is a unique structure. The user must then load the required module, which enables training, before the processing can begin. It's important to emphasise that the user can only download the module if it's necessary.

There are two types of modules – one running on the server-side and the other running on the client-side. Depicted Figure 2.3 presents the general idea of modules.

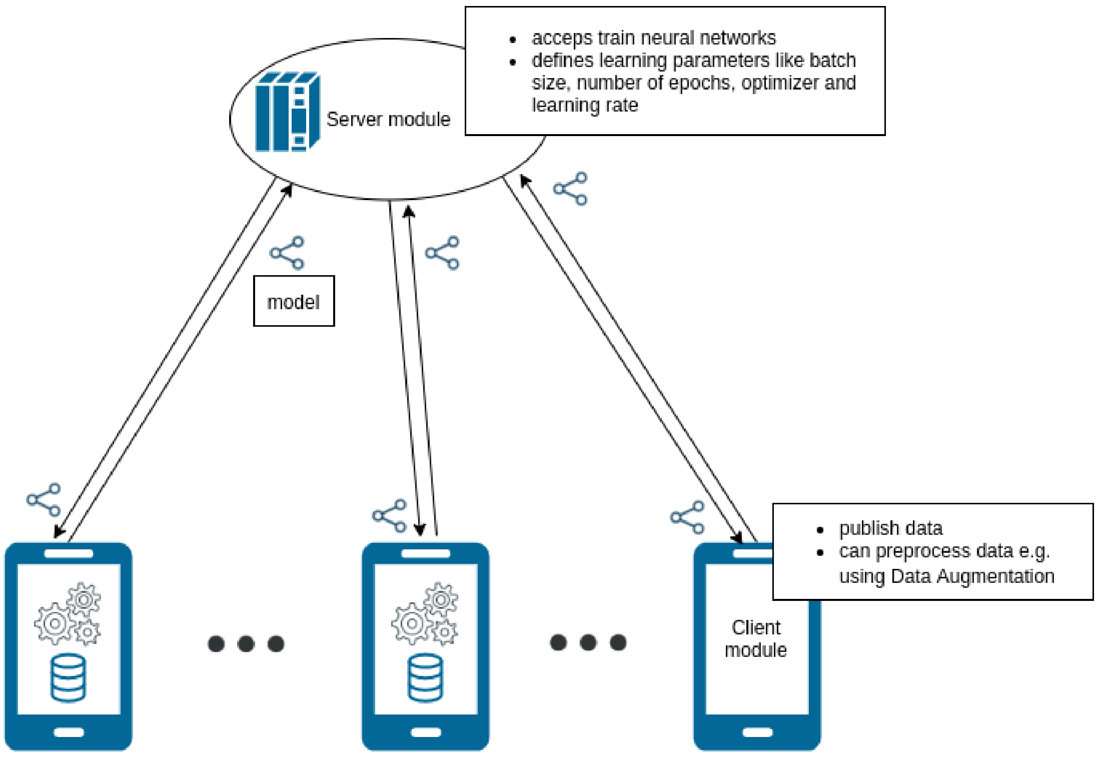

Server module
acceps train neural networks
defines learning parameters like batch size, number of epochs, optimizer and learning rate
model
Client module
publish data
can preprocess data e.g. using Data Augmentation

Figure 2.3: Division of modules

The client module part is responsible for the following:

- informing the server of its data in such a way as not to share it
- starting a TCP connection to allow connection to the server

The server module part is responsible for:

- Connecting with all clients modules which will be participating in the learning round
- Starting the learning process, which consists of the following steps – sending the model to clients, conducting client-side learning, downloading the model and connecting the model

The module is a part of a system responsible for performing machine learning on the client-side. Given module enables learning specific problem on specific kind of devices in a specific way. To explain it better let's consider the digit recognition problem. There are a lot of approaches to solving this problem, such as neural networks (with different configurations), decision trees, and any other possibilities. Each of these possibilities would be a different module in the system. But not only the learning approach that defines the module. It could also be defined by other parameters for example if CUDA should be used etc. So each client has parameters that define them, e.g. amount of RAM, parameter if learning using CUDA is possible, and so on. We proposed such a solution that the client downloads the module depending on the parameters that describe it. Each module is described by appropriate parameters.

In the system 3 parameters are used: the amount of RAM, information if learning using CUDA is possible and device type (computer or mobile phone). This is just an example and of course more parameters could be used. This characteristic is saved as a JSON file.

A sample file looks like this:

```json
{
  "useCuda": "false",
  "RAMInGB": 8,
  "instanceType": "Computer"
}
```

Each module should have 3 basic parameters: data path, identification number, and the address on which the application is running.

2.5.1. Push vs pull

There are two options for handling modules. The first one is that the client sends its specification to the server and then the server returns the proper module. The other approach is quite opposite – the server sends a list of modules with their specification, the client chooses the best suited, and then downloads it from the server.

In this work, the second option was chosen due to the fact that it better meets the needs of the project. The biggest advantage is that client does not share their information so they remain private.

2.5.2. Downloading modules algorithm

Figure 2.4 shows the algorithm of downloading modules. In the beginning Personal actor of the client checks if has a needed module for the task he wants to perform. If he already has the module he can join learning. If not, Personal actor asks the Injector for the modules list.

Modules list are stored in server application as a JSON file. A sample JSON with 2 modules look like this:

```json
{
  "clientModules": [
    {
      "learningTaskId": "mnist",
      "fileName": "mnist.py",
      "description": "Mnist without cuda",
      "useCUDA": false,
      "minRAMInGB": 4,
      "instanceType": "Computer"
    },
    {
      "learningTaskId": "mnist",
      "fileName": "mnist_cuda.py",
      "description": "Mnist with cuda",
      "useCUDA": true,
      "minRAMInGB": 4,
      "instanceType": "Computer"
    }
  ]
}
```

All modules are storied in one directory. They can be accessed using their filename (*fileName* property in the list). The modules shown here are python files, but they can be any files that are executable from the terminal and meet the requirements.

Personal actor receives the modules list in the JSON format as depicted in above JSON. Then he chooses the module best suited for himself and downloads it. Download means that the client sends a request to the server for the needed module giving the name of the module (name of the file – *fileName* property). The response contains the module - file. Once it has received the module, it must save it (in the appropriate folder) and then update the list of modules it has. This list is also stored as a JSON and looks as depicted below. This list has 1 module. Property *taskId* means the identification number of the machine learning task and fileName means filename of the module.

```
[
  {
    "taskId": "mnist",
    "fileName":"server_test.py"
  }
]
```

There is no need to do anything with the modules. The module is activated during the round when the client starts the learning process.

2.5.3. Modules design

In this Section, we describe how a module is structured and what needs to be done to add a new module. The module is divided into two parts – one running on the server-side and the other running on the client-side.

To add a new module, add the server and client parts, which will work as described above (Figure 2.4). Additionally, the modules should accept and properly handle the following parameters:

The Client module accepts following parameters:

- datapath – path, where the data is stored
- id – customer identification number
- host – the address where the client operates
- port – port where the client is running

The Server module accepts following parameters:

- datapath – path, where the test data is stored
- participantsjsonlist – participants list passed as JSON, containing id and address of each user
- epochs – number of epochs carried out

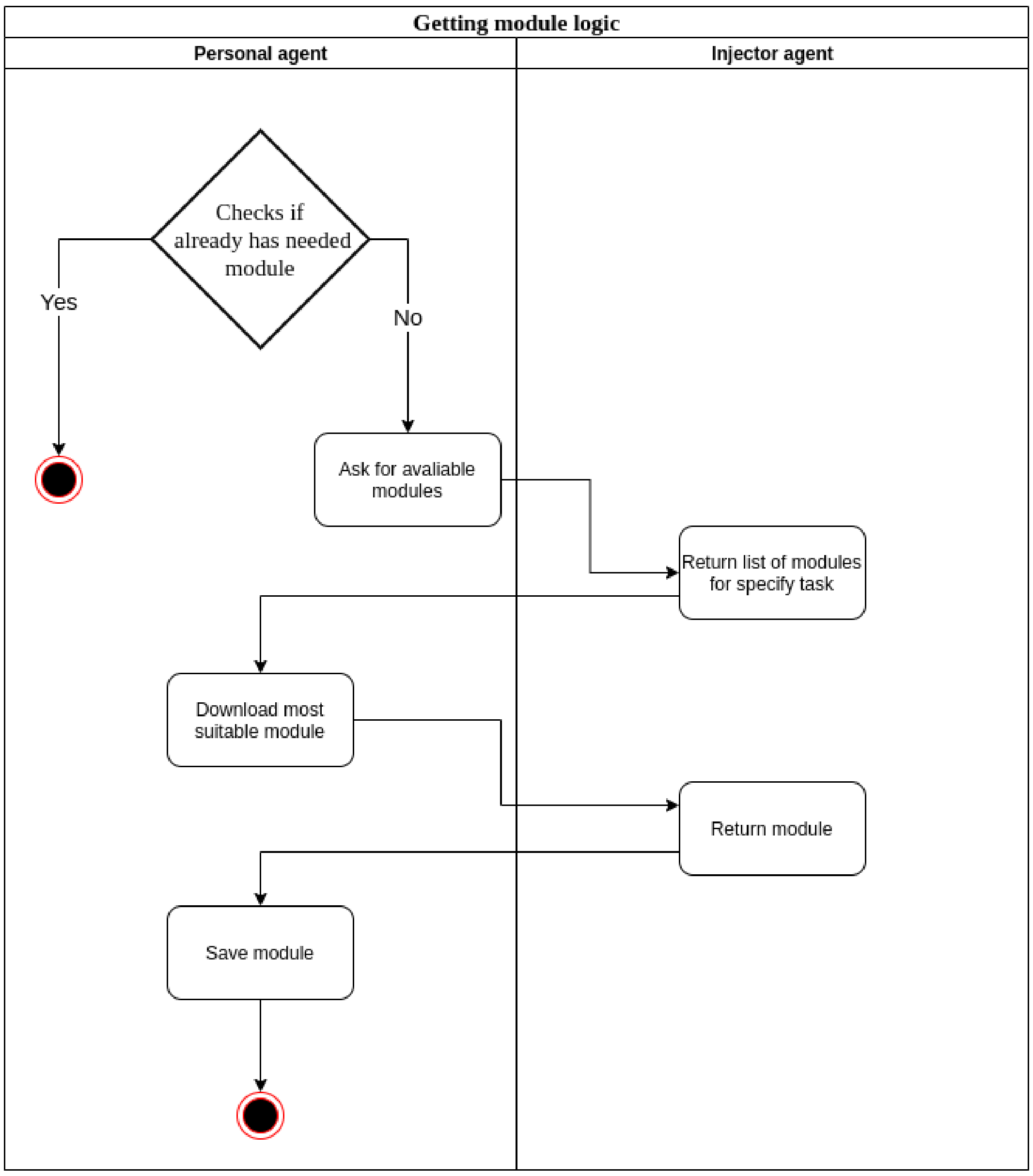

Getting module logic
Personal agent
Injector agent
Checks if already has needed module
Yes
No
Ask for avaliable modules
Return list of modules for specify task
Download most suitable module
Return module
Save module

Figure 2.4: Handling modules

- modelpath – path to model

To add a new module there is a need to pick it in a proper directory and update configuration file.

2.6. Implementation

The platfor was built using Akka with Java. Communication is handled via *Akka remoting*, which is using TCP or Aeron to transport messages. The server consists of following most important classes and functions:

- Server – the main class responsible for starting the server
- Coordinator – class which corresponds to Coordinator actor. Consists of following functions:
  - *onReceive* – responsible for handling messages
  - *startRound* – responsible for starting new round
- Aggregator – class which corresponds to Aggregator actor. Its most important functions are:
  - *onReceive* – responsible for handling messages
  - *runLearning* – responsible for running training module
- Injector – Class responsible for handling modules. In this class the most important function is *onReceive*, which handles *messeges* about modules
- Selector – class which corresponds to Aggregator actor. As in previous classes the most important function is *onReceive*, which handles messages

The client consists of following most important classes and functions:

- Client – the main class responsible for starting the client
- PersonalActor – class which corresponds to Personal actor. Consists of following functions:
  - *onReceive* – responsible for handling messages
  - *saveFile* – saves a module file when received it from the server
  - *findProperModuleStrategy* – strategy that selects the best module from received list

2.7. Data size vs. communication cost

The main reason for using FL is to protect the user's personal data. However, do not forget about the other benefits of FL. The use of FL is worth considering if the data is in different places and it is not profitable (in terms of time and memory) to transfer it to one place. In order to show when it is time-efficient in such a case, a predicting function was proposed. The following assumptions were made:

- Link capacity is 40 Mb/s
- Model size is 3 MB
- Each round consists of 10 epochs
- Round time is 132 s (calculated experimentally)

The plot 2.5 shows a function depending on the number of rounds, and the total size of the data. Light Blue colour means a situation in favour of FL. For example, if the total data size is 20GB, and we want to do 20 rounds, we will achieve it faster using FL than the classic approach. The number of clients participating in the round is irrelevant due to the synchronously performed learning. Of course, the function is only illustrative and a bit simplified, but on its basis, one can see in which cases FL is advantageous in terms of time. Such a situation may arise if the user only wants to check whether learning from the collected data makes sense. If it turns out to be yes, then it can send data to the server and carry out the learning the classical way.

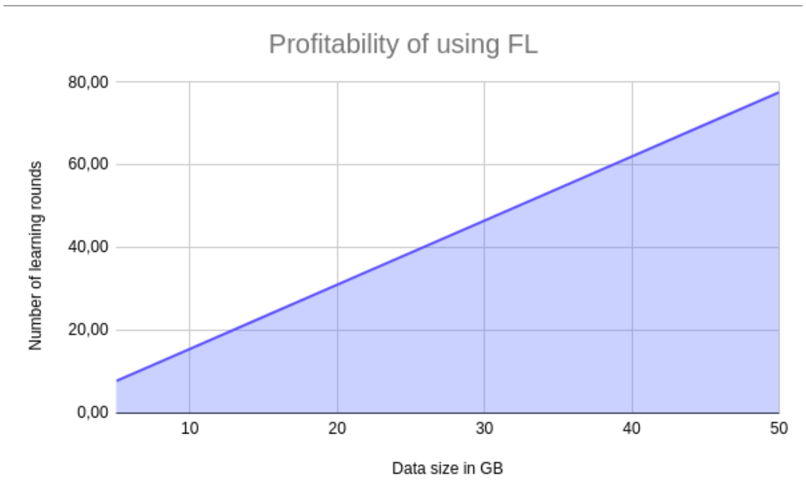


Figure 2.5: Function shows when FL is beneficial

2.8. Sunday-FL platform – summary of architecture and implementation

This Section proposes a platform architecture that meets the following assumptions

- The platform should be scalable
- The platform should be runnable on various environments
- It should be easy to make changes, so that different ways of learning can be tested
- It should be possible to run the platform on different machines

On this basis, a platform was built. It allows to start training both locally and using remote communication. The proposed modularity allows to run various federated learning scenarios, which makes it possible to use platform in various types of tests.

The only functionality that has been discussed in architecture, but has not been implemented, is the possibility of negotiating the conditions for launching the platform. Despite this, platform meets all expectations.

3. Experimental validation of developed platform

Tests can be done quickly and reliably thanks to the platform's flexibility and the ability to easily modify configuration by replacing modules. The only thing that has to be done is to update the learning modules. This Section includes a number of experiments designed to test the platform to see how changing parameters affect learning accuracy.

3.1. Experimental setup

In this Section we describe the learning behaviour in the FL environment. The digit recognition problem, using the MNIST dataset was used in all experiments.

A simple model with 1 hidden layer with 128 neurons was used. It's a very simple model, but is suitable for this problem. Another advantage is that it doesn't need much time to evaluate. It is worth to point out that each experiment was conducted 10 times.

The main target of the experiments was to check how changing parameters may influence on learning accuracy.

3.2. Tests with manipulated dataset

The tests were conducted using the platform. The server was started and the appropriate number of clients depending on the test case. The data was properly prepared using the python script. Each client was assigned an appropriate set of data needed for the experiments.

3.2.1. Accuracy depending on the number of clients

The Figure 3.1 shows data distribution per each client. We can see that data is more or less equally distributed. It means that each category has (roughly) the same number of examples.

The presented datasets are downloaded using the TensorFlow library. This library delivers a set distributed between several thousand customers.

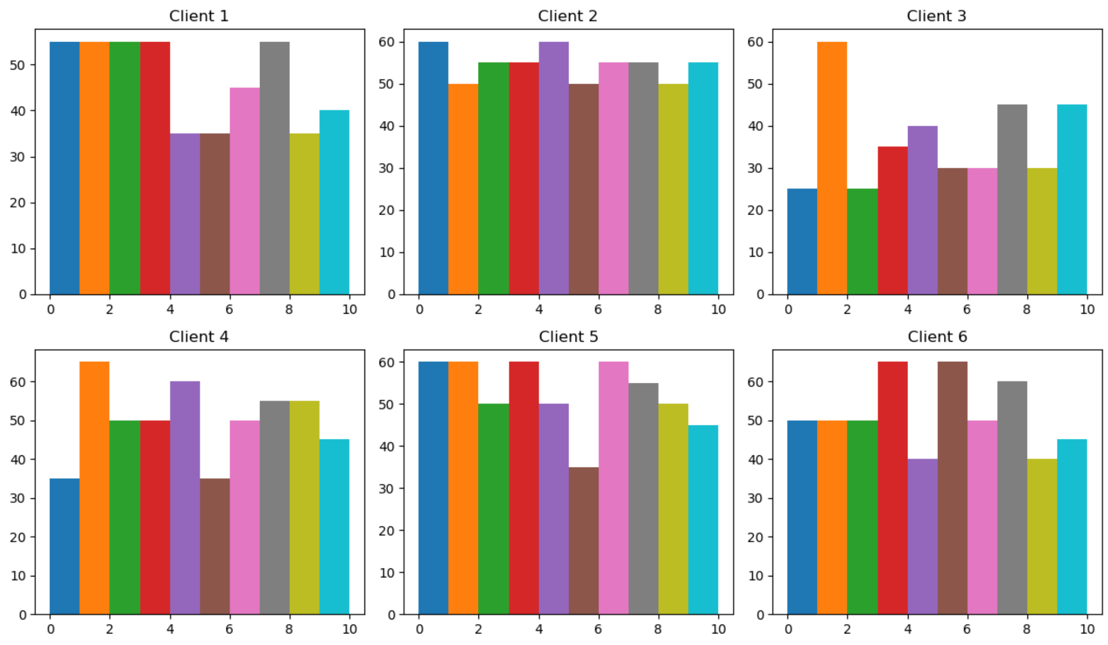


Figure 3.1: Clients data distribution

Figure 3.2 shows the results of training depending on the number of clients.

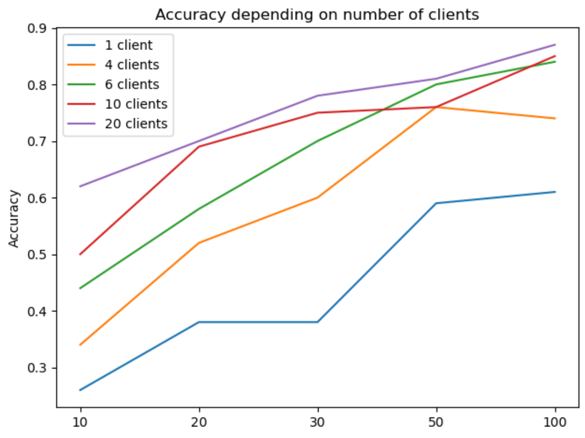


Figure 3.2: Accuracy depending on number of clients

We can see that the more clients participate in training the better results are achieved. This is because the more clients participate in the training, the larger the total data set is. Also the more rounds are performed the better results are achieved. There is no surprise that if we will increase the amount of data and number of rounds the accuracy will increase.

3.2.2. Accuracy depending on data distribution

In this example we have dataset composed of datasets of 2 client from previous example (Figure 3.3). So, the first client here got a dataset composed of two sets of data from 3.1. Hence, the data is the same, but is distributed on a smaller number of clients.

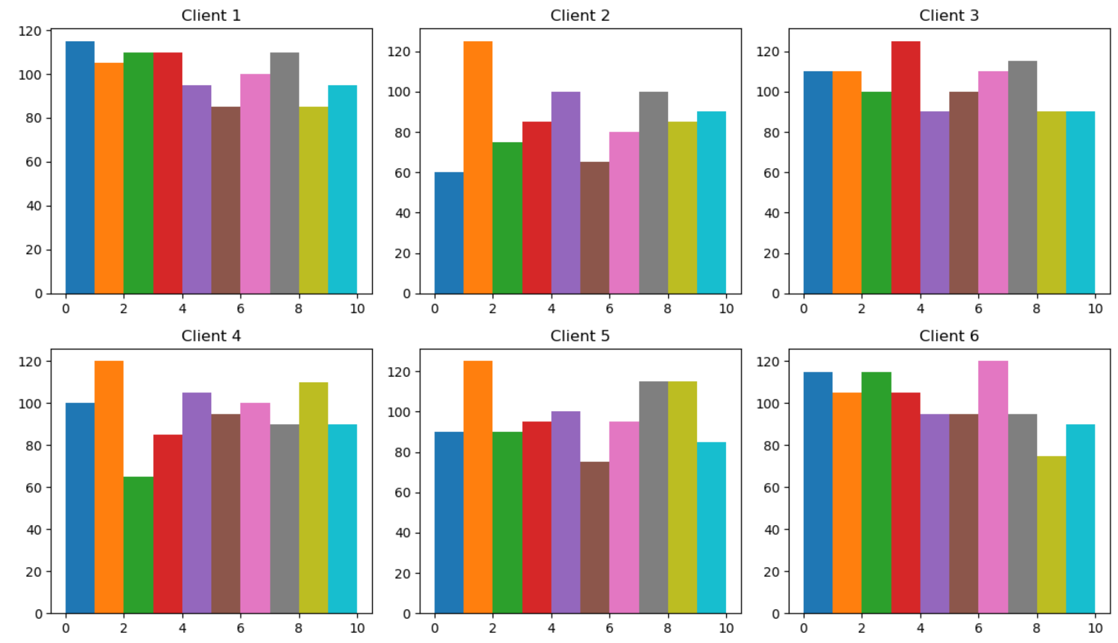
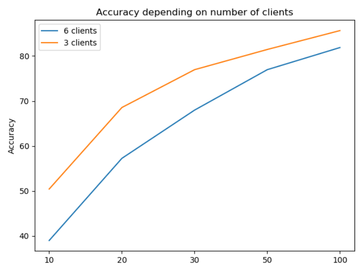


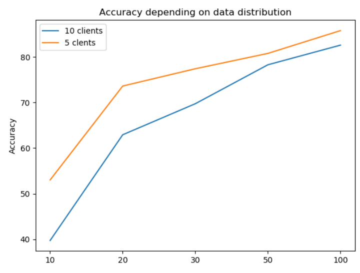
Figure 3.3: Clients data distribution

This example shows that if data is spread between less number of clients it gives as better accuracy

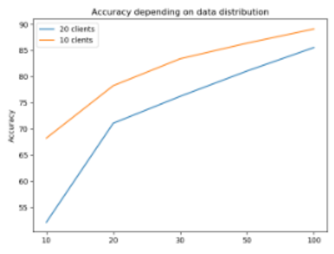
In Figure 3.4a data is distributed over 3 and 6 clients, while in Figure 3.4b over 5 and 10, and in Figure 3.4c over 10 and 20.



(a) 3 clients vs 6 clients



(b) 5 clients vs 10 clients



(c) 10 clients vs 20 clients

Figure 3.4: Accuracy depending on data distribution

3.2.3. Unbalanced data

Here is an example where one of clients has unbalanced data (Figure 3.5). More specifically, it has much more examples of digit "1" than another.

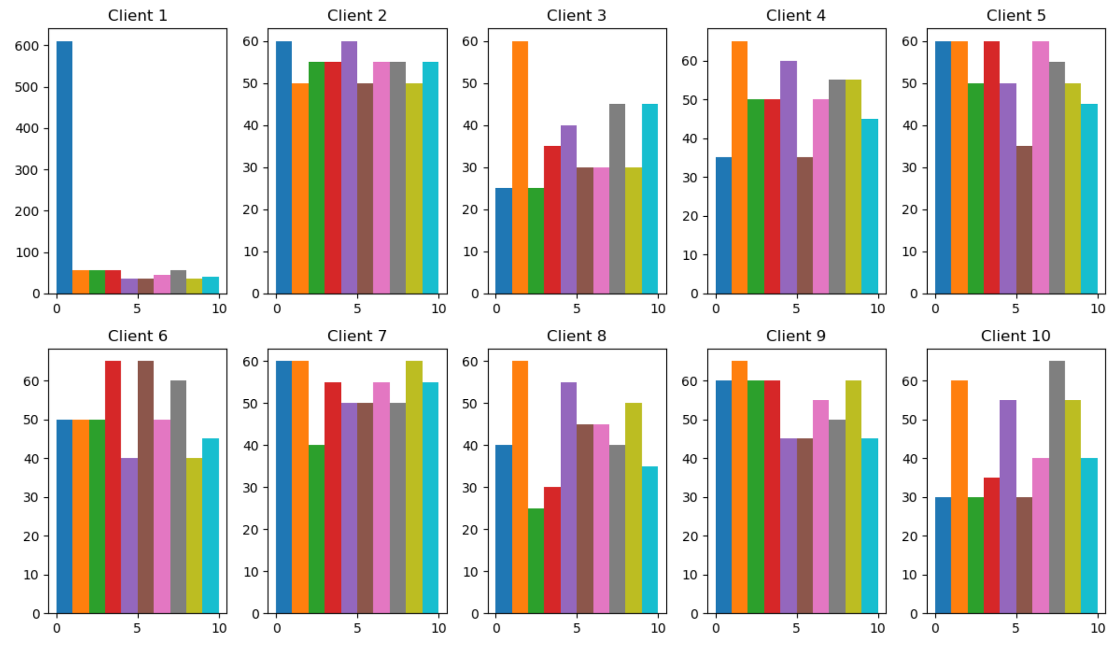


Figure 3.5: Clients data distribution

Surprisingly, scenario when one client has unbalanced data achieved better accuracy (Figure 3.6). It may be caused because 1 digit was added additionally, and dataset for the first client is bigger.

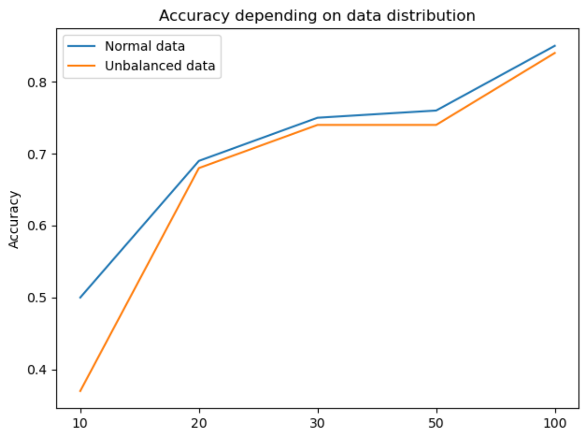


Figure 3.6: Unbalanced data

In Figure 3.7 is an example where each client has unbalanced data. Notice, that if we aggregate that data, composed result will be more or less equally distributed

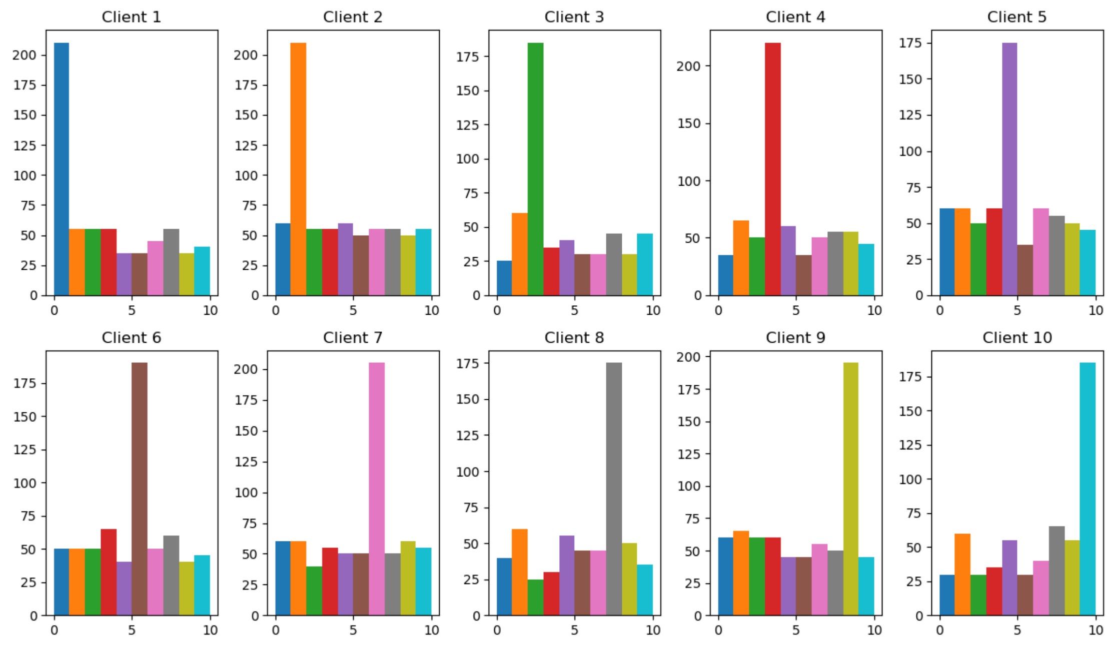


Figure 3.7: Clients data distribution

Results, depicted in Figure 3.8 are similar to previous test. This means, that client with unbalanced data achieved better accuracy.

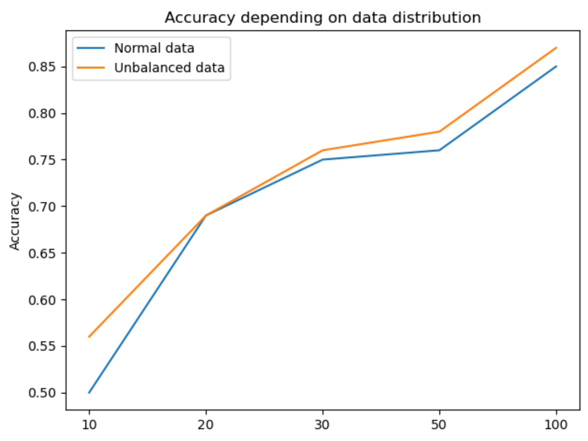


Figure 3.8: Unbalanced data

3.2.4. Fake clients

In the Section 1.1.3 we talked about malicious clients updates. There have been shown cases where the user intentionally wants to ”harm” the model. Now, using the platform, we want to test how such attacks affect the accuracy of the model.

Fake clients were simulated by labeling data in a random way. To achieve this, a special dataset has been created for some clients. Such a collection had incorrectly labeled pictures – randomly.

For example, a picture with the number 5 was marked as a picture with the number 8. Each client was assigned an appropriate set of data needed for the experiments.

The plot 3.9 shows how accuracy depends on the number of fake clients. By fake clients we mean that the client has bad intentions and does not evaluate the model correctly. Fake client intentionally harm the model.

The total number of clients is ten and the more fake clients we have the worse results we achieve. Specifically, for 100 learning rounds, case without fake clients achieved 81%, case with one fake client achieved 73%, with two fake clients – 65%, and with 3 fake clients – 59%. The total number of clients participating in the one round was ten.

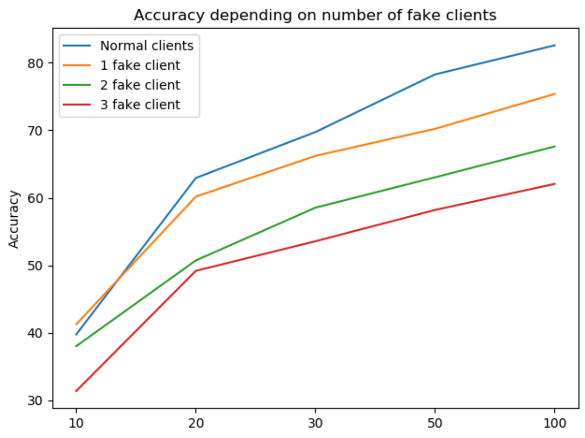


Figure 3.9: Fake clients

3.2.5. Accuracy depending on model aggregation

Presented example shows learning when the model is not shared (Figure 3.10). In this example the learning process looks like that client does not send the model to the server. Model is sent only at the end of the learning process.

For this test a special module was used which don't aggregate model. The model was aggregated only at the end of the training. This means that each client trained initial model, which was merged at the and.

Achieved accuracy is about 10% so it means that model is comparable with the model which selects digit in the random way.

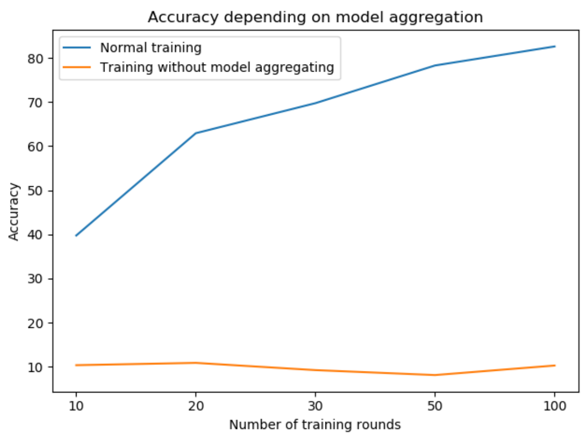


Figure 3.10: Accuracy depending on model aggregation

3.3. Tests showing platform flexibility

In this Section, tests are presented that show the flexibility of the platform to perform various types of experiments. The main purpose of these tests is *not* to show the highest accuracy of the model, but to show that different scenarios can be run easily, using the platform. The datasets used are MNIST – database of handwritten digits, CIFAR-10 – an established computer-vision dataset used for object recognition, and Iris dataset, which consists of 3 different types of irises.

3.3.1. Increasing accuracy

Due to the fact that modular architecture was used, it is possible to flexibly approach the attempts to increase the effectiveness of learning. One of such methods is presented in the article [28]. It tested ways to increase the effectiveness of learning. One is to extend the data by slightly modifying the original images. It consists in creating a new image by shifting the original image in a given direction. In the experiment, for each primary image, 4 extended ones were added, which were created by shifting the original image to 4 different sides (top, bottom, left, right).

Another proven improvement was the appropriate grouping of clients, so that in a given round the sum of classes of participants was as little varied as possible. This was done using the Kullback–Leibler divergence as described by the following algorithm. Here, K_{KL} means Kullback – Leibler divergence function.

Results of experiments are presented in Table 3.1.

The basic method was 72.92% effective. The best result was obtained by the method using data augmentation – almost 80%. Client grouping was practically the same as the basic method –

Algorithm 3 RESCHEDULING algorithm

```
function RESCHEDULING
    $S_{mediator} \leftarrow \emptyset$
    $S_{client} \leftarrow \emptyset$
    do
        Create mediator $m$
        for $|S_{client}| > 0$ and $|m| < \gamma$ do
            $k \leftarrow argmin_i\ K_{KL}(P_m + P_i || P_u)\ i \in S_{client}$
            Mediator $m$ add client $k$
            $S_{client} \leftarrow S_{client} - k$
        $S_{mediator} \leftarrow S_{mediator} \cup m$
    while $S_{client}$ is not $\emptyset$
```

Table 3.1: Results

Used method	Result
Base	72.92 %
Data augmentation	79.99 %
Grouping clients	73.08 %
Both	76.06 %

73.08%. Interestingly, the combination of two methods – client grouping and data augmentation gave a worse result than data augmentation itself. The conclusion is that grouping clients does not improve efficiency, and may even worsen it. Perhaps this is because the classes for the customer data were not diverse enough.

Mediators were simulated by preparing a module, which meets described assumptions.

3.3.2. Clients with different data

Consider a scenario where different devices have datasets of different types. For example, Client 1 has MNIST and CIFAR-10 dataset fragment, Client 2 has a CIFAR-10 and Iris Dataset fragment, and Client 3 MNIST and Iris Dataset fragments. In this way, we can perform 3 machine learning processes – on MNIST, CIFAR-10, and Iris datasets. The test was carried out in such a way that first learning was carried out on the MNIST dataset with the help of clients 1 and 3, then on the CIFAR-10 dataset with the help of clients 1 and 2, and finally on the Iris dataset with the help of clients 2 and 3. To test scenario one round was performed with ten epochs. To conduct such training was only necessary to set the appropriate parameter in the application configuration file and then run it. Customers automatically downloaded the appropriate modules needed for training. The achieved accuracy is 82% for the MNIST dataset, 76% for the CIFAR-10 dataset, and 73% for the Iris dataset.

3.3.3. Testing different parameters

Now let us imagine a scenario where multiple clients have the same data type (in this case, MNIST). To achieve the best possible accuracy of the model, we want to test its different versions with different parameters. In such a scenario, let us consider 2 types of neural networks. One simple with 2 hidden layers and the other a little more complicated – with 7 hidden layers. In addition, it is worth checking the operation of various optimisers. In this test, we will check 2 – SGD and Adam. As in the previous scenario, such tests are easily executable. The tests was conducted using four clients and one round was performed with ten epochs. All we need is the right configuration to conduct the experiment.

The obtained results are presented in Table 3.2. The research has shown that the best accuracy was achieved with neural network with 2 hidden layers and Adam optimiser. This is probably because a more complex network needs more examples to learn.

Table 3.2: Results

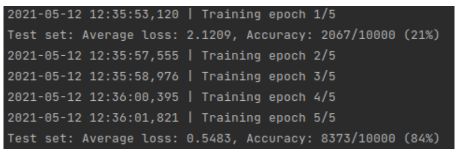
Used method	Result
Neural network with 2 hidden layers + SGD	82 %
Neural network with 2 hidden layers + Adam	84 %
Neural network with 7 hidden layers + SGD	78 %
Neural network with 7 hidden layers + Adam	79 %

3.4. Running platform

As mentioned before platform can be run locally – within single machine and remotely – when different parts are running on different machines.

3.4.1. Running platform locally

Screenshot depicted in Figure 3.11 shows result of running the platform locally. It shows that the final accuracy was 84%. Communication via WebSockets was used in this test

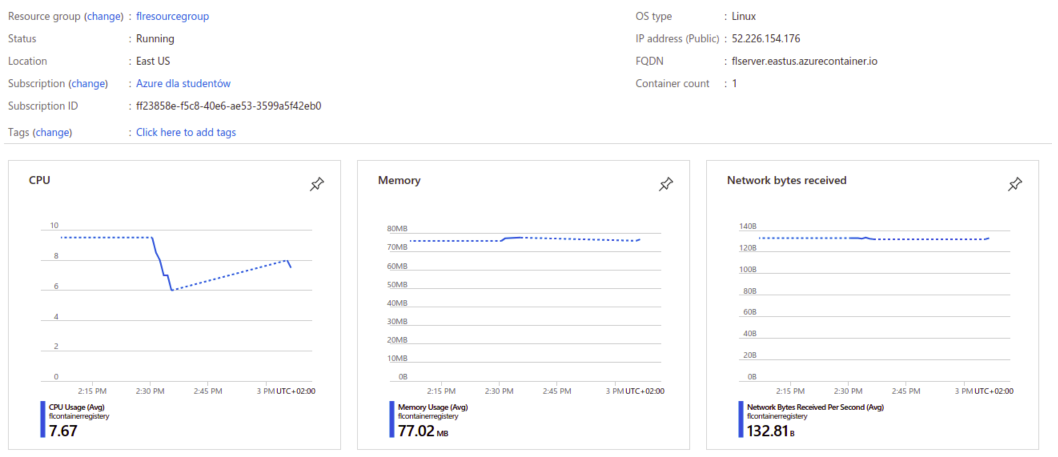


```
2021-05-12 12:35:53,120 | Training epoch 1/5
Test set: Average loss: 2.1209, Accuracy: 2067/10000 (21%)
2021-05-12 12:35:57,555 | Training epoch 2/5
2021-05-12 12:35:58,976 | Training epoch 3/5
2021-05-12 12:36:00,395 | Training epoch 4/5
2021-05-12 12:36:01,821 | Training epoch 5/5
Test set: Average loss: 0.5483, Accuracy: 8373/10000 (84%)
```

Figure 3.11: Accuracy depending on model aggregation

3.4.2. Running platform remotely

To be able to test how the remotely platform works, the server was deployed to cloud (as an Azure container), while the clients were running on two different computers. Screenshot depicted in Figure 3.12 shows a management board on Azure cloud. Due to the fact that platform can be easily containerised it can run on any platform.



Resource group (change)	: flresourcegroup
Status	: Running
Location	: East US
Subscription (change)	: Azure dla studentów
Subscription ID	: ff23858e-f5c8-40e6-ae53-3599a5f42eb0
Tags (change)	: Click here to add tags
OS type	: Linux
IP address (Public)	: 52.226.154.176
FQDN	: flserver.eastus.azurecontainer.io
Container count	: 1

Figure 3.12: Running server on Azure cloud

4. Concluding remarks

Federated learning attracts a lot of attention. A lot of research, concerning its potential application, is done in various fields (e.g. medicine and IoT). Moreover, existing literature deals with various issues, such as ways to increase the accuracy of the model, or ways to defend against malicious attacks. However, analysis of literature indicates that, as usually in the early stage of development, missing are "general purpose" tools that would support research. Specifically, there exist platforms that can be used to apply federated learning to various datasets. These tools are robust and supported by "big players" in the field. However, they are not easily extendable to include "modules" needed to explore various scenarios to be able to answer open research questions. As a result, researchers that study federated learning build their own "local-FL-platforms" and perform "local simulations". Furthermore, even though there are multiple FL platforms under development, their use implies reliance on existing tools (e.g. well-known ML libraries). They also have disadvantages that make research difficult. One of them is the difficulty to experiment with using remote communication. To respond to these challenges, the proposed approach aims at development of a flexible modular FL platform that can be used, extended, and customised depending on user needs. The platform is scalable and the number of participants can take part in training as needed. Making changes is very easy – we just need create the appropriate module(s). The platform also allows to test remote communication, which is very important when testing with various devices types.

Currently, the platform is available from the GitHub repository [5]. Obviously, during research that led to this Thesis, it was established that the platform should be extended in a number of directions. The following ones are the most crucial:

- Ways of describing module content should be extended, possibly by using ontologies and semantic technologies. This, while introducing a lot of desired flexibility and robustness, involves substantial development of the core of the platform.
- There is also a need to add negotiations; mainly between clients and manager(s). For instance, possibility of (a) joining/leaving the process, or (b) negotiating the conditions for

starting the training. Such negotiations require specification of a "negotiation language" and may, again, involve use of semantic technologies.

- Ability of instantiating complex workflows, based on combining (orchestrating use of) multiple modules needs to be added. This can involve also GUI-based tool(s) that would support workflow definition. Observe that workflow definition would have to be translated to series of actions on the side of the clients, but also involving training management infrastructure.
- The platform currently only runs on the Linux operating system. It should be possible to run also on other – Windows, macOS, as well as various mobile systems (e.g. Android)

Nevertheless, the platform is already being used (at last) in the following research: (1) study of federated learning in agriculture (N. Kumar, Wyższa Szkoła Menedżerska, Warszawa), (2) dealing with heterogeneity of data across nodes (A. Danilenka, Politechnika Warszawska), and (3) defence against attacks (D. Kolasa, Politechnika Warszawska). As a consequence, the platform is expected to be further evolved and used.

Based on the work, a article (”Sunday-FL – Developing Open Source Platform for Federated Learning") was written and it will be published soon.

5. Acknowledgements

I would especially like to thank a few people for their help. In particular, I am very grateful to Maria Ganzha and Marcin Paprzycki for supervising my work, numerous suggestions, and ideas. Additionally, I would like to thank Kumar Nalinaksh, for his ideas for work development and review of the Master's Thesis.

Bibliography

[1] 5g iot industry. https://www.prnewswire.com/news-releases/the-worldwide-5g-iot-industry-is-expected-to-grow-to-40-2-billion-by-2026-at-a-cagr-of-73-from-2021--301275202.html.

[2] Ai in healthcare. https://www.healtheuropa.eu/exploring-the-benefits-of-ai-and-data-in-healthcare/107683/.

[3] Drug discovery. https://venturebeat.com/2020/09/17/major-pharma-companies-including-novartis-and-merck-build-federated-learning-platform-for-drug-discovery/.

[4] Fedvision. https://www.fedai.org/cases/computer-vision-platform-powered-by-federated-learning.

[5] Github repository. https://github.com/pioek11111/Federated-learning.

[6] Google blog. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[7] Ibm and china's digital bank webank jointly held workshop on federated learning. https://syncedreview.com/2020/02/07/ibm-and-chinas-digital-bank-webank-jointly-held-workshop-on-federated-learning.

[8] An industrial grade federated learning framework. available online. https://fate.fedai.org/.

[9] Intel. intel movidius myriad 2 vpu specifications. available online:. https://newsroom.intel.com/wpcontent/uploads/sites/11/2017/06/Myriad-2-VPU-Fact-Sheet.pdf.

[10] Iot connected devices. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[11] Iot connected devices worldwide. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide.

[12] Iot in smart cities. https://statetechmagazine.com/article/2021/04/researchers-eye-machine-learning-secure-iot-data.

[13] Mobile users don't want to share data. https://www.mediapost.com/publications/article/178739/mobile-users-dont-want-to-share-data-with-apps-or.html.

[14] Number of photos. https://focus.mylio.com/tech-today/how-many-photos-will-be-taken-in-2021.

[15] Nvidia. https://blogs.nvidia.com/blog/2019/12/01/clara-federated-learning.

[16] Processors for ai. https://www.eetimes.eu/top-10-processors-for-ai-acceleration-at-the-endpoint.

[17] Pysyft, a python library for secure and private deep learning. https://github.com/OpenMined/PySyft.

[18] Sensor market. https://www.globenewswire.com/news-release/2021/02/08/2171162/0/en/Sensor-Market-to-Hit-USD-228-Billion-by-2026-to-Attain-CAGR-of-6-22-APAC-Region-to-Spearhead-the-Global-Sensors-Market.html.

[19] Tensorflow federated. (2016). tensorflow. retrieved may 3, 2021. https://www.tensorflow.org/federated.

[20] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[21] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[22] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.

[23] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H Brendan McMahan,

et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[24] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[25] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2020.

[26] Erick Cantú-Paz. *Efficient and accurate parallel genetic algorithms*, volume 1. Springer Science & Business Media, 2000.

[27] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[28] Moming Duan, Duo Liu, Xianzhang Chen, Yujuan Tan, Jinting Ren, Lei Qiao, and Liang Liang. Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 246–254. IEEE, 2019.

[29] Chenyou Fan and Ping Liu. Federated generative adversarial learning. In *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*, pages 3–15. Springer, 2020.

[30] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[31] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidyalingam S Sunderam. *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing.* MIT press, 1994.

[32] Yiyuan Gong and Alex Fukunaga. Distributed island-model genetic algorithms using heterogeneous parameter settings. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 820–827. IEEE, 2011.

[33] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[34] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[35] Andrew Hard, Kurt Partridge, Cameron Nguyen, Niranjan Subrahmanya, Aishanee Shah, Pai Zhu, Ignacio Lopez Moreno, and Rajiv Mathews. Training keyword spotting models on non-iid data with federated learning. *arXiv preprint arXiv:2005.10406*, 2020.

[36] Thomas Hiessl, Daniel Schall, Jana Kemnitz, and Stefan Schulte. Industrial federated learning–requirements and system design. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 42–53. Springer, 2020.

[37] Ce Ju, Dashan Gao, Ravikiran Mane, Ben Tan, Yang Liu, and Cuntai Guan. Federated transfer learning for eeg signal classification. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 3040–3045. IEEE, 2020.

[38] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[39] Renuga Kanagavelu, Zengxiang Li, Juniarto Samsudin, Yechao Yang, Feng Yang, Rick Siow Mong Goh, Mervyn Cheah, Praewpiraya Wiwatphonthana, Khajonpong Akkarajitsakul, and Shangguang Wang. Two-phase multi-party computation enabled privacy-preserving federated learning. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 410–419. IEEE, 2020.

[40] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Blockchained on-device federated learning. *IEEE Communications Letters*, 24(6):1279–1283, 2019.

[41] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[42] Mohamed Kurdi. An effective new island model genetic algorithm for job shop scheduling problem. *Computers & operations research*, 67:132–142, 2016.

[43] GeunHyeong Lee and Soo-Yong Shin. Reliability and performance assessment of federated learning on clinical benchmark data. *arXiv preprint arXiv:2005.11756*, 2020.

[44] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1544–1551, 2019.

[45] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *arXiv preprint arXiv:1907.09693*, 2019.

[46] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211*, 2020.

[47] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[48] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[49] Gan Sun, Yang Cong, Jiahua Dong, Qiang Wang, and Ji Liu. Data poisoning attacks on federated machine learning. *arXiv preprint arXiv:2004.10020*, 2020.

[50] Ahmet Ali Süzen, Burhan Duman, and Betül Şen. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–5. IEEE, 2020.

[51] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *ACM Computing Surveys (CSUR)*, 53(2):1–33, 2020.

[52] Darrell Whitley, Soraya Rana, and Robert B Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of computing and information technology*, 7(1):33–47, 1999.

[53] Qiong Wu, Kaiwen He, and Xu Chen. Personalized federated learning for intelligent iot applications: A cloud-edge based framework. *IEEE Open Journal of the Computer Society*, 1:35–44, 2020.

[54] Binhang Yuan, Song Ge, and Wenhui Xing. A federated learning framework for healthcare iot devices. *arXiv preprint arXiv:2005.05083*, 2020.

[55] Wenyu Zhang, Xiumin Wang, Pan Zhou, Weiwei Wu, and Xinglin Zhang. Client selection for federated learning with non-iid data in mobile edge computing. *IEEE Access*, 9:24462–24474, 2021.

[56] Xinwei Zhang, Mingyi Hong, Sairaj Dhople, Wotao Yin, and Yang Liu. Fedpd: A federated learning framework with optimal rates and adaptivity to non-iid data. *arXiv preprint arXiv:2005.11418*, 2020.

[57] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

List of symbols and abbreviations

FL	federated learning

List of Figures

List of Tables