FEDERAL UNIVERSITY OF SANTA CATARINA
JOINVILLE'S TECHNOLOGICAL CENTER
FACULTY OF AEROSPACE ENGINEERING

HANS HERBERT SCHULZ

DECENTRALIZED BRILLIANCE: DEPLOYING A FEDERATED LEARNING PLATFORM AND EVALUATING AGGREGATION ALGORITHMS

Joinville
2024

HANS HERBERT SCHULZ

DECENTRALIZED BRILLIANCE: DEPLOYING A FEDERATED LEARNING PLATFORM AND EVALUATING AGGREGATION ALGORITHMS

Bachelor Thesis submitted as a requirement for obtaining the bachelor's degree in Aerospace Engineering at Joinville's Technological Center of the Federal University of Santa Catarina.

Supervisor: Prof. Dr. Benjamin Grando Moreira

Co-supervisor: Eng. Gustavo Laydner de Melo Rosa

Joinville
2024

I dedicate this work to my family for their unwavering support and encouragement throughout my academic endeavor.

ACKNOWLEDGMENTS

As graduation approaches, it provides a meaningful opportunity for me to reflect on my academic journey and the people who have been a part of it.

First and foremost, I thank God for his boundless benevolence and blessings throughout my life.

I extend my gratitude to my family, Janete, Edemar, Ana Vitória, and Maria Elisa whose unconditional support proved indispensable for paving the path to this milestone.

To my supervisor Benjamin Grando who promptly accepted guiding my ideas for this thesis and my co-supervisor Gustavo Laydner, whose supportive hand and penchant for problem-solving propelled this project to its current stature.

To my internship supervisor Philip Jongebloed, whose charisma, guidance, and leadership were like a fine symphony—now sorely missed. *Herzlichen Dank für alles!*

To my colleagues from the Fraunhofer's 220 *Abteilung*, especially, Heiko Baumman, Sören Herkströter, and Tim Nagel. I look forward to our next *bierchen* at Kaktus.

To the friends I made in Germany—Berkan Akin, Berkay San, Gustavo Mello, Inés Mesquida, João Vitor Smagar, Julian Storms, Karl Ngondji, Marten Gralla, Shamil Kudukkath, and Sjoerd Hammers. Fondly reminiscing on the good times we shared.

To Lucas Ma Famg, whose priceless support and dubious sense of humor brightened even the longest workdays, leaving an indelible mark on this project.

To the friends at UFSC, especially, Pieter van Tilburg, Irisson Lima, Luiz F. Camargo, Beatriz Faga, Henrique Zschornack, and Widmark Kauê.Your support and patience have been invaluable to me and I wouldn't have gone this far without you.

To the Fraunhofer Institute for Production Technology and the Federal University of Santa Catarina (UFSC) for providing the necessary means and resources to the conclusion of this project.

To all others that somehow contributed to the making of this work.

"Aber in dem allen überwinden wir weit durch den, der uns geliebt hat." - Römer 8:37

ABSTRACT

With the exponential development in the artificial intelligence field, the concern for privacy and data acquisition has increased dramatically. Several governments have already restricted their data protection laws to better shield individuals from information leakage. The Federated Learning (FL) approach was conceived to mitigate any leakage possibility so that models could be trained without accessing one's private data. Furthermore, it allows several clients who are geographically apart to partake in the training process without sharing data. This work deployed an FL platform in two clients using container technology. Moreover, a training procedure using the two most common FL strategies was performed using the MNIST dataset. Finally, the achieved results were able to tell whether the platform was correctly deployed and which strategy performed better, both within the model and within the hardware capabilities of the clients.

Keywords: Federated learning. Docker. Artificial intelligence. IBM.

RESUMO

Com o desenvolvimento exponencial no campo da inteligência artificial, a preocupação com a privacidade e a aquisição de dados aumentou dramaticamente. Vários governos já restringiram suas leis de proteção de dados para melhor proteger o indivíduo contra vazamentos de informações. Para mitigar a possibilidade de vazamento, foi concebida a abordagem de Federated Learning, de modo que os modelos pudessem ser treinados sem acessar os dados privados de uma entidade. Além disso, ela permite que vários clientes, geograficamente separados, participem do processo de treinamento sem compartilhar dados. Para este trabalho, uma plataforma de Federated Learning foi implantada em dois clientes usando tecnologia de contêineres. Além disso, foi realizado um procedimento de treinamento usando as duas estratégias de Federated Learning mais comuns e usando o conjunto de dados MNIST. Finalmente, os resultados alcançados foram capazes de determinar se a plataforma foi implantada corretamente e qual estratégia teve um desempenho melhor, tanto dentro do modelo quanto dentro das capacidades de hardware dos clientes.

Palavras-chave: Aprendizado Federado. Docker. Inteligência Artificial. IBM.

LIST OF FIGURES

LIST OF CHARTS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
FedAvg	Federated Averaging
FedSDG	Federated Stochastic Gradient Descent
FL	Federated learning
HFL	Horizontal Federated Learning
IBM	International Business Machines Corporation
MNIST	Modified National Institute of Standards and Technology
OS	Operating System
RAM	Access Memory
ReLu	Rectified Linear Unit
RNN	Recurrent Neural Network
SVM	Support Vector Machine
VFL	Vertical Federated Learning
VM	Virtual Machine

SUMMARY

1 INTRODUCTION

After the Second World War ended, many scientists started questioning how clever machines could become (ANYOHA, 2020). Among those men was British mathematician Alan Mathison Turing, who published a paper entitled *Computing Machinery and Intelligence* in which he described the idea of creating a machine that could *learn* from experience and how to assess its results (TURING, 1950).

In the current state of affairs, artificial intelligence (AI) is being applied in several fields of science and industry. Among those fields is predictive maintenance, which seeks to find the best time to change or repair components from a production process to mitigate the costs of shutting down a machine due to unscheduled disruptions (BEHERA, 2016). However, it still presents many challenges that must be overcome. Among these challenges, we may find the ever-growing need for computational power and data privacy concerns.

A commonly used method in machine learning is to train models using centralized datasets. However, this approach poses potential risks, especially in system breaches that could expose all data used for training (LUDWIG et al., 2020).

Such risks are amplified in specific contexts, including hospital environments with sensitive patient health data and government instances with national security implications. Some examples of these breaches are the Anthem Inc. breach, considered the largest one in the United States of America so far, (UNITED STATES DEPARTMENT OF HEALTH & HUMAN SERVICES, 2020) and the U.S. Office of Personnel Management breach that affected almost 21,5 million people (NBC NEWS, 2015).

Furthermore, in recent years, several laws have been passed to limit and protect data, such as the General Data Protection Regulation(GDPR), which further challenges the scenario for machine learning applications.

Additionally, given the amount of data to be processed for the *deep learning* context, the computing costs for a centralized server are humongous. The FL approach also seeks to reduce such costs and improve the overall performance of the training procedure (SHARMA; SHAMOUT; CLIFTON, 2019).

The proposed work is part of an industrial solution entitled *AI-NET-ANIARA* (IPT, 2021), developed by the *Fraunhofer-Institut für Produktionstechnologie* (IPT) that seeks to enhance the scope of predictive maintenance in the production engineering scenario. The project intends to develop a multi-sensory platform to acquire and transmit air parameters (e.g., saturation, temperature, and flow speed) from industrial air exchange systems. With this data, train machine learning models to enhance predictive maintenance.

The deployment process is also described and comprises a series of tasks to install a pre-developed application into its intended operational environment (HEYDARNOORI; MAVADDAT, 2006). To deploy the platform, containerization through the *Docker* application is possible. After a successful deployment, performance tests are conducted utilizing the usual FL techniques for machine-learning model algorithms, and the results are discussed.

1.1 OBJECTIVES

1.1.1 General Objective

The main objective of this work is to deploy a Federated Learning platform and check its functionality by comparing the performance of machine learning models using different federated learning strategies.

1.1.2 Specific Objectives

To achieve the desired general objective, the following steps will be sequentially taken:

- Deploy an already existing model of federated learning on Raspberry Pi devices through digital containers and conclude the connection between server and client;
- Compare parameters such as GPU, network usage, and training time between different federated learning methods and the conventional centralized method;
- Explore the advantages and challenges of federated learning that resulted from this work.

2 BACKGROUND

As industries around the globe grow, more technology is being required and developed. Among those technologies, Federated Learning was born as a pioneering approach to some common issues faced in the field of machine learning, such as privacy and data storage. This chapter aims to discuss the backbone of the digital infrastructure and the main concepts used in the industrial multi-sensory platform.

2.1 FEDERATED LEARNING OVERVIEW

With the advent of deep learning (DL), the need for data acquisition in several science instances has increased. This is mainly because DL models are usually interpolators, meaning they present the best results when the input is similar to the data used for training models. Therefore, a vast amount of heterogeneous data is required to achieve decent accuracy (SARMA et al., 2021).

One way to contour this issue would be to create pooled data silos, where several institutions would upload their acquired material to the joint database. However, such a method would present a significant security and privacy risk for each party involved. The question remains: How can vast amounts of heterogeneous data be gathered and maintained in order to keep the desired privacy and security?

In light of this inquiry, Google presented in 2016 the concept of federated learning (FL), which can be described as: "[...] A machine learning technique [...] as a way of providing decentralized and collaborative learning across distributed nodes" (MANIAS, 2021, p. 3).

In other words, the FL architecture comprises multiple parties and an aggregator agent. Firstly, a global model is generated and sent to each party. Then, the model will be trained in each party with its own data set, meaning each model will have a different result by the end of the process.

An update that contains the difference between the base model and the new one is then generated by the parties and sent to the aggregator agent. The agent then merges the updates and creates a new global model, which is sent to each party, and so on (MANIAS, 2021).

2.2 HORIZONTAL AND VERTICAL DATA DISTRIBUTION

Data can be distributed and used in the FL context in different ways. The conventional way consists of having multiple parties training a model independently, and the data in all parties possess the same features. Such distribution results in the

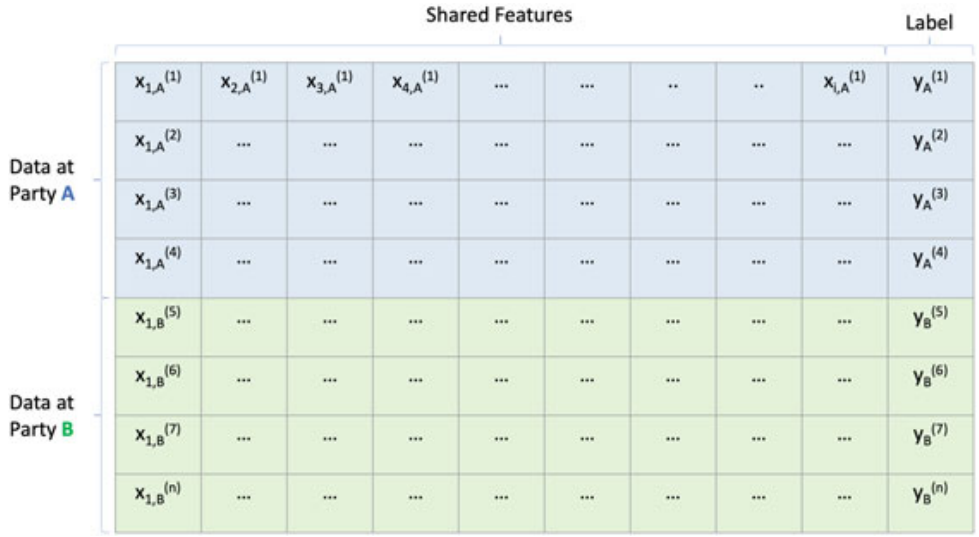
so-called Horizontal Federated Learning (HFL) approach (LUDWIG; BARACALDO, 2022).

However, most data sets in the real world will sometimes have different features. Ludwig e Baracaldo (2022) propose an example to illustrate such a scenario better.

Suppose that a patient in a medical environment requires surgery, and a model is being developed to anticipate the outcome of the operation. The medical institution will have different forms of data on him. Primary care physicians, for instance, may have records regarding the individual's health status throughout different visits. A radiologist, on the other hand, may have data on the patient's X-rays. This means that the hospital would have two different data features for the same patient, and they can both help predict the outcome of a surgery. So, the general idea would be to train a model using both features related to the same patient. This approach is called Vertical Federated Learning (VFL).

To summarize, the main difference between HFL and VFL lies in how the training data is distributed among parties. The HFL will focus on training multiple data samples that share the same features, whereas the VFL will use the same sample but with different features. Figure 1 depicts the horizontal FL since it has different subjects marked as $X_{i,A^{i+1}}$ and $X_{i,B^{i+1}}$ have the same features which are represented by X_{i+1,A^i}.

Figure 1 – Horizontally partitioned data



Source: Ludwig e Baracaldo (2022, p. 12).

On the other hand, Figure 2 represents different features being trained about the same subject, represented by the column containing the $X_{4,[A,B]^{i+1}}$.

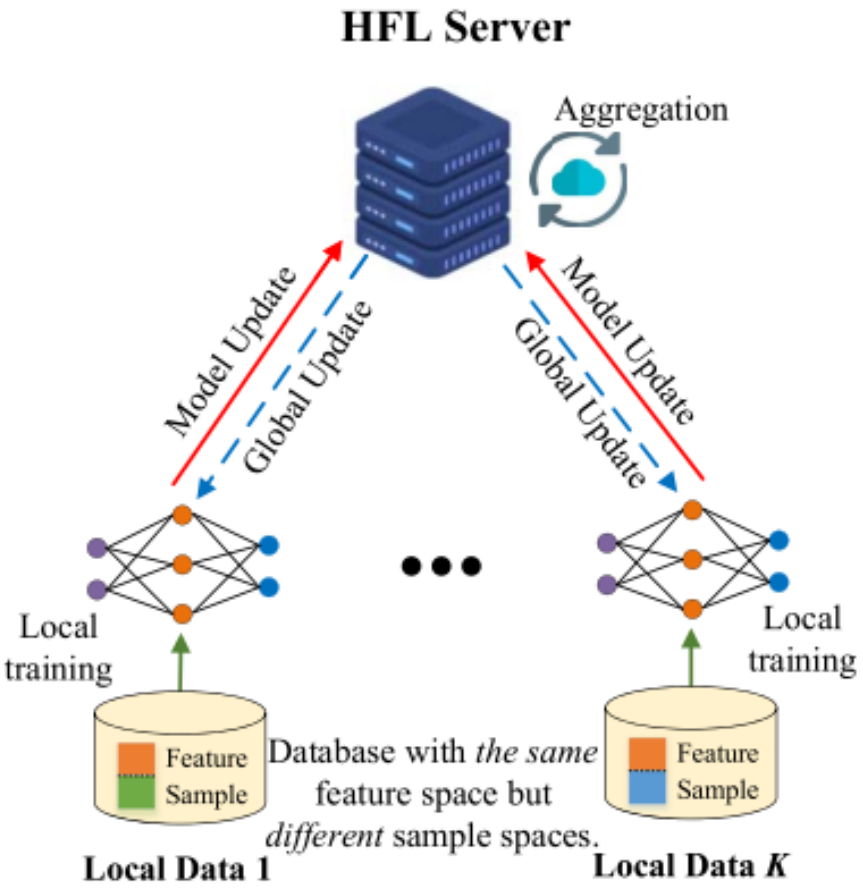
Figure 2 – Vertically partitioned data

Source: Ludwig e Baracaldo (2022, p. 12).

After comprehending the data distribution, it is worth noting that the overall FL process presents some differences for each type.

In the HFL, all parties will train a global model locally. Since the data has the same features, all parties can use the same AI model (e.g., deep learning and decision trees). Then, all newly trained models will be aggregated, and a new updated global model will be computed (NGUYEN et al., 2021). Figure 3 illustrates the procedure.

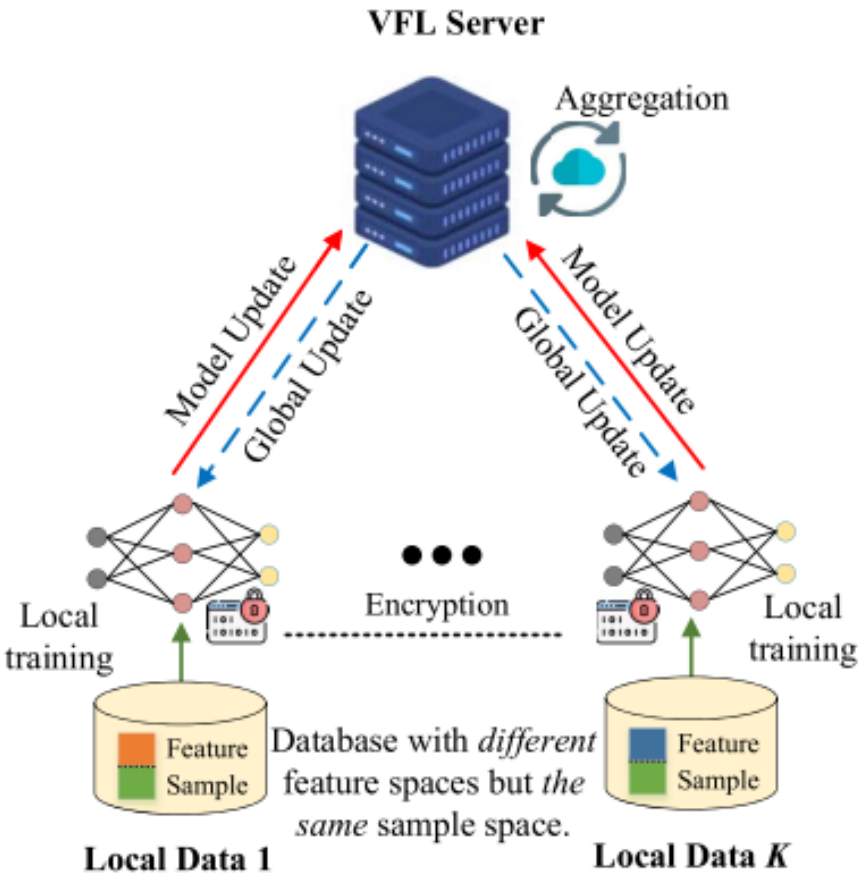
Figure 3 – Horizontal federated learning model



Source: Nguyen et al. (2021, p. 1627)

Conversely, the VFL will share an AI model with parties with the same set of data samples but different data features. Then, an alignment entity will overlap the data samples of parties. Such samples will be combined to train the standard AI model (NGUYEN et al., 2021). The process scheme is shown in Figure 4.

Figure 4 – Vertical federated learning model



Source: Nguyen et al. (2021, p. 1627).

2.3 HORIZONTAL AND VERTICAL TRAINING MODELS

From a system perspective, the FL scenarios aim to train machine learning models using disparate data to maintain privacy and overall model performance. It is imperative that data is not moved across parties and not even visualized by the central server. To achieve that, there are mainly two system designs: cross-device, usually used with the VFL data partition, and cross-silo, often used with HFL(LUDWIG; BARACALDO, 2022).

2.3.1 Cross-Device Federated Learning

According to Ludwig e Baracaldo (2022), the objective of cross-device FL is minimizing the function $F(w)$, which is defined by Equation 1.

$$F(w) = \sum_{k=1}^{n} p_k F_k(w) \tag{1}$$

In Equation 1, $F_k(w)$ is the local objective function for the device k with model weights w. The term p_k represents the importance of the contribution of the k device to the global model objective function. Such a term is described by Equation 2 as:

$$p_k = \frac{j_k}{\sum_{l=1}^{n} j_l} \tag{2}$$

Where j_k depicts the total number of data points the local model w_k was trained on in the k device. Therefore, it can be inferred that p_k is the weighted average of the number of data points per device(LUDWIG; BARACALDO, 2022).

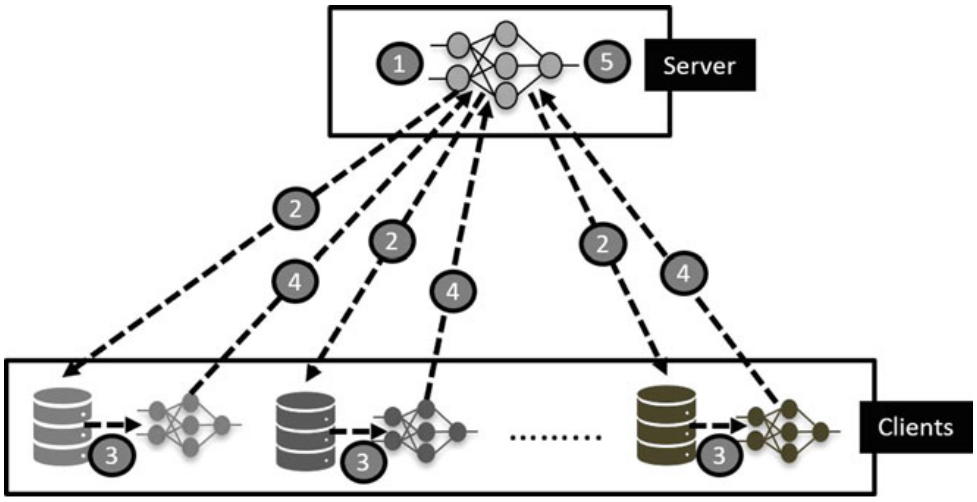
There are also additional constraints that must be fulfilled. Firstly, it is essential that no third parties can access end-user data, and it must not be associated with a particular device. Secondly, the achieved performance of a model using FL must be close enough to the performance obtained using traditional methods (LUDWIG; BARACALDO, 2022).

The global model training is executed in rounds, and each round consists of five steps, which are also described by Ludwig e Baracaldo (2022) as:

1. Initially, the model G_0 is untrained and initialized with random weights. At the beginning of each round, a subset of the available devices will be selected. The number of selected devices will significantly impact the training time, the model performance, the convergence time, and the computational cost of the system. Therefore, it is a critical point of the process. However, the basic FL implementation suggests that around 10% of the available devices should be randomly used.
2. After selecting the k devices, the initial weights w of the global model G_n will be sent to the devices. It is crucial to note that this is a network-expensive step since deep learning models can be large, and therefore, transferring them across the devices has a great bandwidth cost.
3. Training then starts for each individual dataset D^k, which will result in the model g_n^k being created in the respective k device.
4. After the generation of the local models, privacy mechanisms are put in place to ensure that none of the newly trained models can be linked to a specific device. Then, it is sent to the aggregator.
5. The aggregator then uses an aggregation algorithm to generate a new global model G_{n+1} and the process returns to Step 1. This set of procedures is repeated until a stopping condition is met, such as the maximum round number or some model convergence criteria.

Figure 5 visually represents each of the aforementioned steps.

Figure 5 – Cross-device federated learning system overview



Source: Ludwig e Baracaldo (2022, p. 200).

2.3.2 Cross-Silo Federated Learning

The main objective of Cross-Silo Federated Learning can be formally described as minimizing the $L(W)$ function described by Equation 3.

$$L(W) = \sum_{j=1}^{m} \sum_{i=1}^{n} L(w_j, x_j^i) \tag{3}$$

where $L(W)$ is the global loss function for the global model W, i represents the client, j is the data provider (i.e., silos) with the local model w_j trained from data feature x_j^i. It simply means training a model using data from different silos that refer to the same client but possess different features and labels (LUDWIG; BARACALDO, 2022).

Similarly to the Cross-Device FL, this system has two constraints. The first concerns privacy matters and states that no parties involved in the training can be associated with another party's data point ID. In practical terms, it means that one party knows that another party is involved but cannot access the detailed data of that other party. Such constraint can be met using the homomorphic encryption technique[1], which uses mathematical operations to transform data so that its original form cannot be observable.

The second constraint regards performance, and it states that the overall global performance using this system must be as close to the performance of a model that would have been generated had all the data been present in a single silo as in traditional distributed learning (LUDWIG; BARACALDO, 2022).

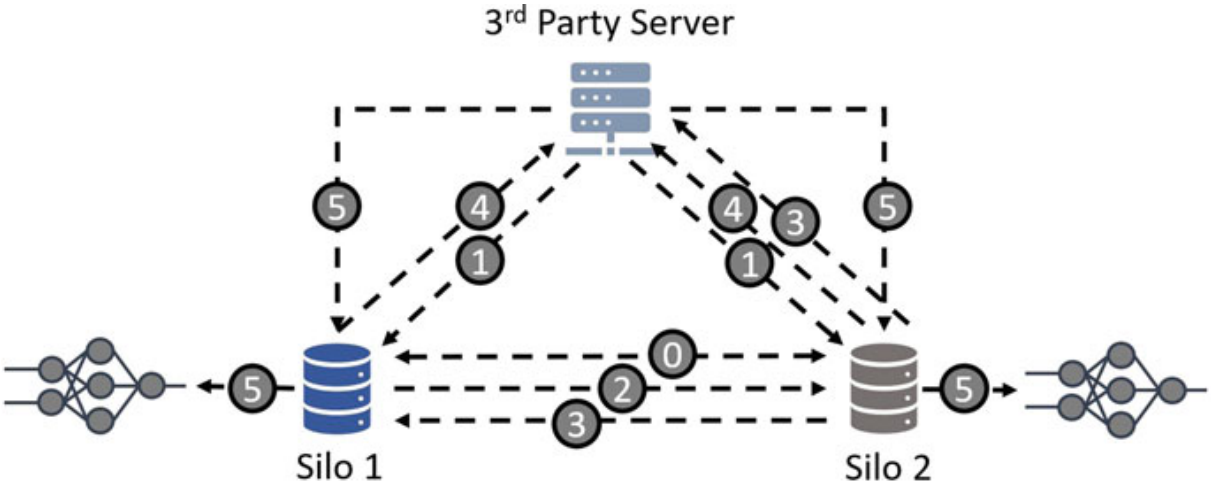
[1] The homomorphic encryption technique allows mathematical operations to be carried on a cipher text instead of the actual data. The main requirement of the technique is that the final output must be the same after operating on the encrypted text and the plain text (ROCHA; LÓPEZ, 2019).

The training procedure also happens in rounds but with one significant difference: all silos take part in every round of the training. The key steps are shown below according to Ludwig e Baracaldo (2022). For simplicity, only two data silos are being considered, but the steps can be extended to any desired number of silos.

1. Before the training, all participants must align their datasets using an anonymous data alignment technique. One could find secure multi-party communication, key-sharing, and randomized responses among those techniques. These methods can be rather computationally expensive but usually run once before the training. The output of this step is such that the data between the parties are aligned with the same ID. The unmatched ones are usually discarded.
2. A third-party aggregator will generate and send encryption key pairs to each party for secure communication, and the *partial models* that need to be trained are initialized. The term *partial models* is used because, usually, for this system, it is assumed that only the last party to be trained contains the labels. The other parties only have the training features and the part of the model that can perform a forward pass using those features.
3. The first party trains on one mini-batch of its local data and creates an output. This output is then encrypted via homomorphic encryption and is sent to the next party.
4. The second party also performs a forward pass with its own data. Since it was assumed that there were only two parties for the example, it is also possible to assume that the labels are contained in the second and final party. Therefore, the calculation of the loss function happens here. The intermediate outputs are then sent to the first party, where they will be required to update the party's model weights. The loss is sent to the third-party aggregator.
5. Both parties will then calculate their partial gradients, add another layer of encryption, and send them to the third-party aggregator. The extra layer serves so that no information can be obtained in the intermediate outputs.
6. The aggregator will then decrypt the last layer and, along with the loss values from Step 3, will be able to determine the exact gradient for each partial model in all the participants and send them back to their respective silos. These gradients will then be used to update their local models and generate new ones. After that, the process is repeated from Step 01.

The Cross-silo FL steps are depicted in Figure 6.

Figure 6 – Cross-silo federated learning system overview



Source: Ludwig e Baracaldo (2022, p. 206).

It is worth reassuring that, among the differences, the devices that act as the party are usually distinct from one system to another. Usually, in cross-device FL, the devices are user-owned IoT (Internet of Things) instruments such as cellphones, tablets, edge devices, and others. Since there are so many options in the market, the training may have parties from an array of different devices and specifications. In the context of cross-silo, the data silos used as parties tend to be of commercial grade, providing a less heterogeneous environment and generally having better computational resources than IoT devices (LUDWIG; BARACALDO, 2022).

Therefore, when implementing an FL environment, the targeted devices used as parties to train a model will play a significant role in choosing one of the aforementioned systems.

2.4 FEDERATED LEARNING ALGORITHMS

After establishing the main concepts of FL and its benefits and challenges, it becomes clear that the next step is optimization to address common issues, such as unbalanced data sets and limited communication. For such a process, algorithms have been created, which are usually applicable to any finite-sum objective (MCMAHAN et al., 2023). Hence, the problem becomes minimizing the $f(w)$ function that is defined by:

$$f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{4}$$

where w is a vector containing d model parameters.

Usually, in supervised learning, it is said that $f_i(w) = l(x_i, y_i; w)$. In other words, it represents the loss of the prediction on the labeled example (x_i, y_i) made with model parameters (w). It is also assumed that there are K clients and $\mathscr{P}$ is the set of indexes of data points on client k, and the number of training examples held by each client is

denoted as $n_k = |P_k|$ (MCMAHAN et al., 2023). Therefore, Equation 4 can be rewritten as:

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \tag{5}$$

where

$$F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(w) \tag{6}$$

Each client will then hold a part P_k of all training examples and compute $F_k(w)$, representing the average loss on client k (NILSSON et al., 2018).

Among the many available algorithms, it is worth noting the Federated Stochastic Gradient Descent (FedSGD) and the Federated Averaging (FedAvg) as they are the most common in FL(LUDWIG; BARACALDO, 2022).

2.4.1 Federated Stochastic Gradient Descent

According to McMahan et al. (2023), the latest successful applications of deep learning have heavily relied on the variants of the stochastic gradient descent (SGD) for optimization. Therefore, the most straightforward approaches for FL algorithms can be conceived by having the SDG as its backbone.

The SDG can be naively applied to an FL problem where a single batch gradient calculation is done per round of communication. Such a method is computationally efficient, but it usually requires several training rounds to produce reasonably good models (MCMAHAN et al., 2023).

To apply this technique in FL, a C fraction of clients is chosen in each round, and the gradient of the loss over all the data held by the clients is computed. Therefore, C refers to the global batch size with $C = 1$ representing the full batch setup (MCMAHAN et al., 2023).

A typical implementation is proposed by McMahan et al. (2023) with $C = 1$, a fixed learning rate η. In this scenario, each client k will compute $g_k = \nabla F_k(w_t)$ (the average gradient on its local data for model w_t). Then, the central server aggregates these gradients and applies the update $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$ since that $\sum_{k=1}^{K} \frac{n_k}{n} g_k = \nabla f(w_t)$.

An equivalent update is found by $\forall k, w_{t+1}^k \leftarrow w_t - \eta g_k$ followed by $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$. This means each client will locally take one step of gradient descent on the current model using its local data. The server then takes a weighted average of the resulting models (MCMAHAN et al., 2023).

2.4.2 Federated Averaging

The Federated Averaging algorithm, FedAvg, is an extension of the FedSDG. The central concept follows the same steps motioned above but also iterates the local update $w_k \leftarrow w^k - \eta \nabla F_k(w^k)$ in each client multiple times before the averaging step (MCMAHAN et al., 2023).

Besides the previous hyperparameters C and η, the FedAVG will count with two more. B represents the local mini-batch size, and E is the number of iterations through the local data before the global model is updated (NILSSON et al., 2018).

A pseudo-code is proposed by McMahan et al. (2023) and is presented within Algorithms 1 and 2.

Algorithm 1: Federated Averaging. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate (MCMAHAN et al., 2023).

Data: Initialize w_0
for *each round* $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$;
  $S_t \leftarrow$ random set of m clients;
  for *each client* $k \in S_t$ ***in parallel*** **do**
    $w_k^{t+1} \leftarrow \texttt{ClientUpdate}(k, w_t)$;
  end
  $m_t \leftarrow \sum_{k \in S_t} n_k$;
  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$;
end

Algorithm 2: `ClientUpdate`(k, w) (MCMAHAN et al., 2023).

$B \leftarrow (split\ P_k\ into\ batches\ of\ size\ B)$
foreach *local epoch* i *from 1 to* E **do**
  for *batch* b *in* B **do**
    $w \leftarrow w - \eta \nabla l(w, b)$;
  end
end
return w to server

2.5 CONTAINERIZING

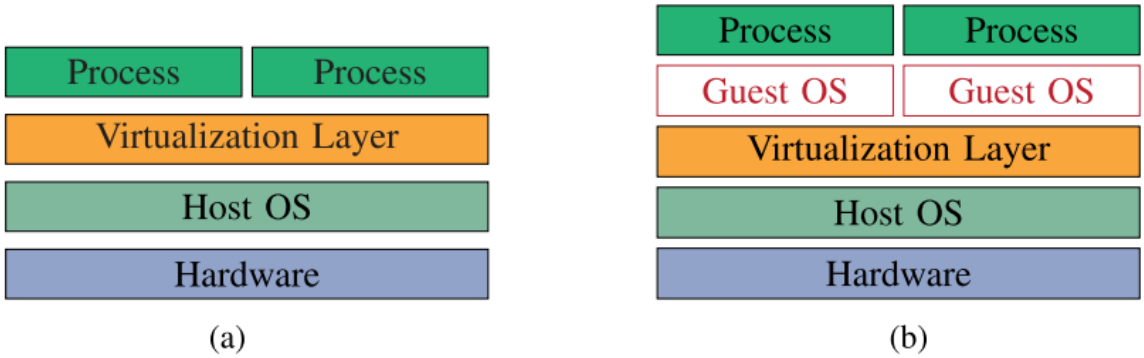
Since the central concept of FL is enforcing data privacy by not centralizing all data in one major center, it is evident that the silos or devices used will be physically apart, which can generate a compatibility issue. Some of those devices may have different operating systems (OS) or may be in a different hardware version for instance. To overcome this challenge, the concept of virtualization is introduced.

In its essence, computer virtualization allows a single physical machine to be split into several virtual machines (VM) (SHEA; LIU, 2012). More recently, however,

the container technology was created as a lightweight alternative to VMs (SULTAN; AHMAD; DIMITRIOU, 2019). A container is a group of processes isolated from other groups through different kernel namespaces and resource allocation quota (WAN et al., 2019).

Unlike the VMs, containers share only the operating system kernel, not a full copy of the OS. This reduces a container's startup time to milliseconds and the computational resources needed, making them a sensible pick for usage in edge devices (SULTAN; AHMAD; DIMITRIOU, 2019). Figure 7 depicts both container and VM architectures.

Figure 7 – Simplified architecture. (a) Container. (b) Virtual Machine



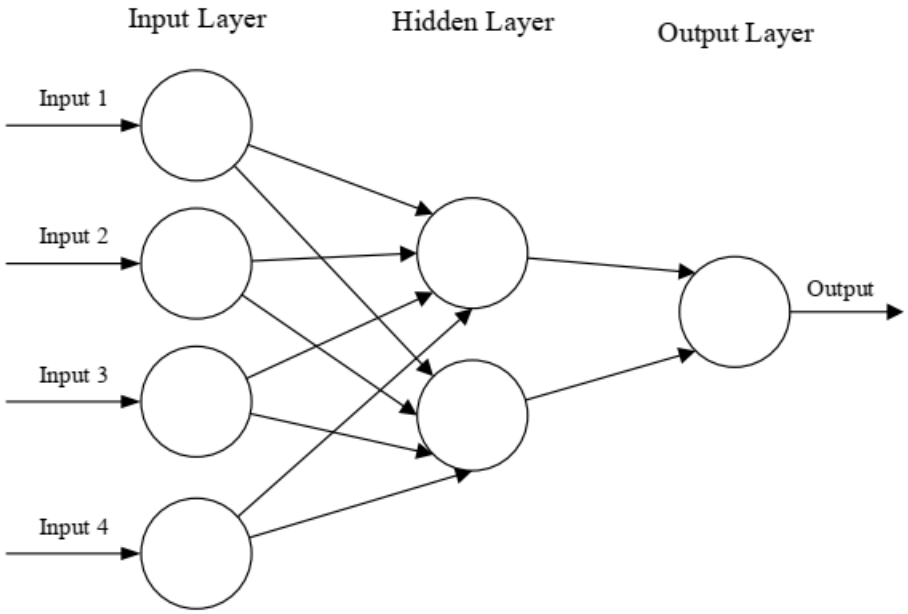
Source: Sultan, Ahmad e Dimitriou (2019, p. 52977).

2.6 NEURAL NETWORK ARCHITECTURE OVERVIEW

Artificial Neural Networks (ANNs) are a series of computational systems that attempt to mimic the behavior of a biological neural system. The ANNs usually comprise a series of interconnected computational nodes that are referred to as *neurons* whose primary function is to receive an input and collectively assist the learning and optimization of the final output by the entire network (O'SHEA; NASH, 2015).

A generic ANN model is shown in Figure 8. Usually, input is given in the form of a multidimensional vector and forwarded to the hidden layer. The hidden layer will make decisions based on previous layers and evaluate whether the final output improves or worsens. If a model has multiple hidden layers, it is commonly referred to as deep learning (O'SHEA; NASH, 2015).

Figure 8 – Generic ANN model



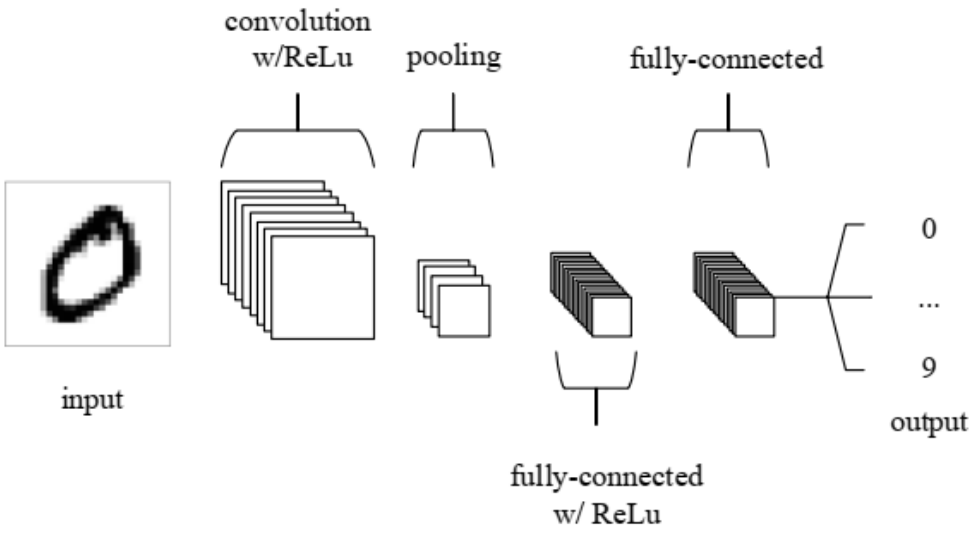
Source: O'Shea e Nash (2015, p.2)

One of the many variations of ANNs is the Convolutional Neural Networks (CNNs), which are designed mostly for image pattern recognition and are the project's main scope. The term *convolutional* refers to the algebraic operation of convolution, which consists of an operation that combines two different functions and creates a third one. Mathematically, it can be described as (HERMAN, 2015):

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t) \cdot g(x - t)dt \tag{7}$$

The CNN architecture is composed of three-layer categories. A convolutional layer, a pooling layer, and fully-connected layers. A basic architecture designed for the MNIST dataset is shown in Figure 9 containing one input, one convolutional, one pooling, and two fully-connected layers.

Figure 9 – Simplified CNN architecture with five layers



Source: O'Shea e Nash (2015, p.4)

The convolutional layer determines the output of neurons connected to local regions through the convolution operations. The rectified linear unit (ReLu) aims to apply an activation function, such as the sigmoid, to the output of the activation produced by the previous layer (O'SHEA; NASH, 2015).

The pooling layer, on the other hand, performs down-sampling along the spatial dimensionality of the given input. Finally, the fully-connected layers attempt to produce class scores from the activations so that they can be classified (O'SHEA; NASH, 2015).

3 METHODOLOGY

This project is part of an industrial solution led by Ericsson AB that aims to develop a multi-sensory platform that monitors air parameters (e.g. saturation, temperature, and flow speed) from industrial exchange systems (IPT, 2021).

Since this project has many partners across Europe, the exchange systems are installed in several locations. Hence, an FL platform is needed to acquire and use data provided by these facilities.

This endeavor comprises several sequential steps. Initially, the appropriate FL library is picked. The next step is understanding its operating mechanism to implement it within the project. Further on, the deployment mechanism must be selected alongside a deployment platform. Then, the setup FL platform will be tested using model training with the MNIST dataset. Finally, the results will be evaluated, and the performances achieved with different FL strategies will be compared.

3.1 FEDERATED LEARNING LIBRARIES

One of the critical concepts of the project is deploying an FL platform to perform the necessary training in the developed models. Therefore, the choice of the primary library is paramount. The key aspects considered for a thorough analysis were the accepted ML frameworks, data partitioning type, privacy and security, FL strategies, FL paradigms, and ML models. Chart 1 presents the most prominent libraries and their characteristics.

Chart 1 – Libraries feature comparison.

Features/ Framework		Pysyft	Flower	IBM FL	TFF	FedML
General	**Contributor**	Open - Minded	Adap Gmbh	IBM	Google	FedML Inc.
	Environment	Windows, Mac, Linux, Docker	Windows, Mac, Linux, Docker	Windows, Mac, Linux, Docker	Windows, Mac, Linux, Docker	Windows, Mac, Linux, Docker
	ML Framework	Pytorch	Pytorch, TF, Libtorch, JAX...	Pytorch, TF, Scilearn, Keras	TF	Pytorch, TF
Architecture	**Data Partitioning**	Vertical Horizontal	Vertical Horizontal	Horizontal	Horizontal	Vertical Horizontal
	Data Types	Numbers, Text, Image, Time-Series	Numbers, Text, Image, Time-Series	Numbers, Text, Image, Time-Series	Numbers, Text, Image, Time-Series	Numbers, Text, Image, Time-Series
	Privacy and Security	HE, MPC, DP	SecAgg	Multiple cryptographic methods	DP	Secret sharing key agreement
	FL Strategy	FedAVG, FedSGD	FedAVG, FedSGD, QF FedAvg ...	FedAVG, FedSGD, FedAVG+ ...	FedAVG, FedSGD	FedAVG, FedNOV, FedNAS...
Paradigm	**Vertical FL**	yes	yes	no	no	yes
	FTL	no	yes	no	no	yes
	Cross-Device	yes	yes	no	no	yes
	Cross-Silo	yes	yes	yes	yes	yes
	Hetero-task learning	no	yes	yes	no	yes
	Edge computing	yes	yes	no	yes	yes
ML Models	**Regression**	yes	yes	yes	no	yes
	Clustering	no	yes	yes	no	no
	Trees	no	yes	yes	no	no
	SVM	no	no	yes	no	no
	Bayes networks	no	no	yes	no	no
	NN	yes	yes	yes	yes	yes
	DNN	yes	yes	yes	yes	yes
	CNN	yes	yes	yes	yes	yes
	RNN	yes	yes	yes	yes	yes

Source: Adapted from Saidani (2023, p. 67).

Since the application is enterprise-oriented, the matters of privacy, security, and fast model specification stand above all else. Moreover, a horizontal data partitioning approach was selected for the project. Additionally, only Flower and IBM FL are production-ready libraries (SAIDANI, 2023). After examining Chart 1, the IBM FL library emerges as the optimal choice since it has multiple cryptographic methods, a wide range of FL strategies, and several supported ML models. A comprehensive exposition of the chosen library is made below.

3.1.1 IBM FL General Concepts

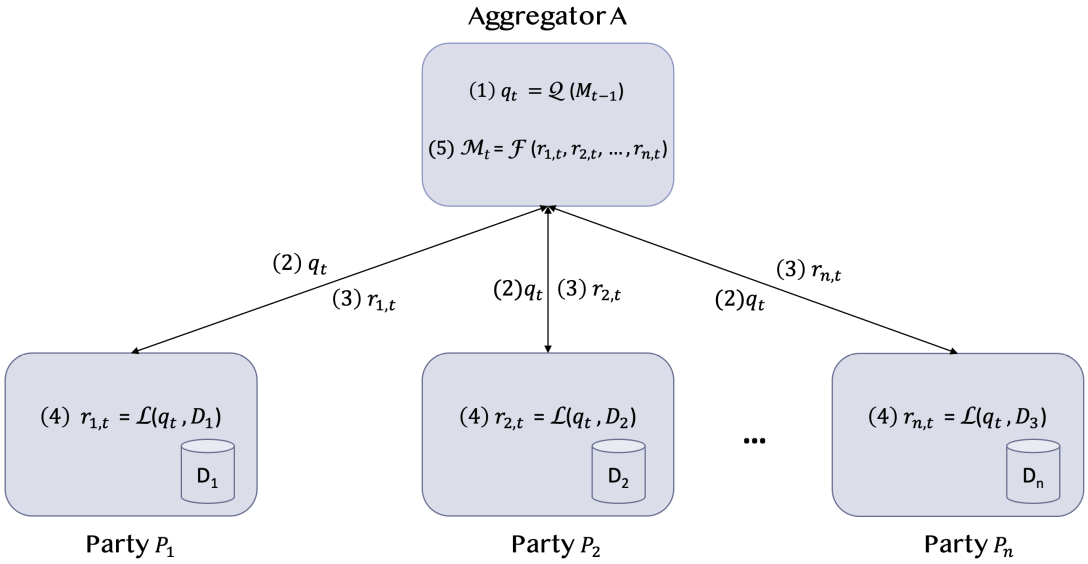
For a practical understanding, it is essential to go over the main functionalities of the IBM FL library.

The platform trains a model M using a data set D. However, the data set is split into n parties. It is important to note that each party P_i owns a unique section of the data set D_i, and no party can access the other party's data set. Additionally, the

Federated Learning process also includes an aggregator A that cannot access any of the party's data set. The process steps below are described by Ludwig et al. (2020) and shown in Figure 10:

1. To train M_G, the aggregator uses a function Q that takes as input the current model or state of the training M_t at round t and generates a next query q_{t+1}.
2. One such query, q_t, requests information about a local model or aggregated information about each party's data set. Example queries include requests for gradients or model weights of a neural network or count for decision trees.
3. The local training process applies a function $\mathscr{L}$ that takes query q_t and the local dataset D_i and outputs a model update $r_{i,t}$. Usually, the query, q_t, contains information that the party can use to initialize the local training process, for example, model weights to start with local training or candidate feature values and/or class labels to compute counts for.
4. $r_{i,t}$ is sent back from party P_i to the aggregator A, which collects all the $r_{i,t}$ from parties P_i.
5. When parties model updates $r_{1,t}$, $r_{2,t}$, ..., $r_{n,t}$, where $r_{i,t}$ refers to the model update of party i at round t, are received by the aggregator forming set $R_t = \{r_{1,t}, r_{2,t}, \ ..., r_{n,t}\}$, they are aggregated by applying fusion function F that takes as input R_t and returns M_t. (LUDWIG et al., 2020, p.4)

Figure 10 – Conceptual scheme of IBM's federated learning

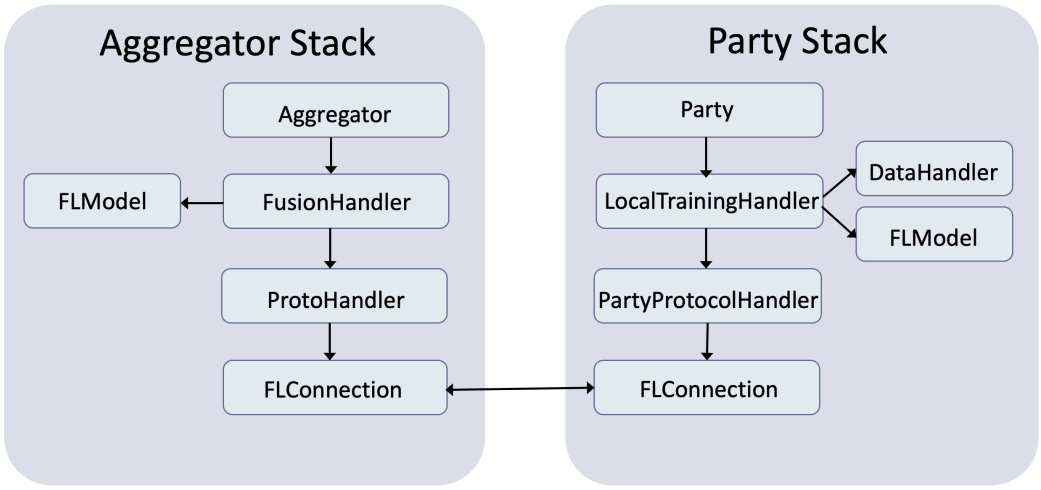


Source: Ludwig et al. (2020, p.5)

3.1.2 General Architecture

The framework is built modularly to make the communication infrastructure independent of the machine learning library and the federated learning algorithm. The main modules are the aggregator, hence the name Aggregator Stack, and the parties P_i stack. Additionally, this modular division allows the parties to read and pre-process from different locations, which is the platform's ultimate goal Ludwig et al. (2020). Figure 11 provides a representation of the platform's architecture.

Figure 11 – Architecture stack of IBM's federated learning



Source: Ludwig et al. (2020, p.6)

The main components described by Ludwig et al. (2020) are:

1. **Connection**: Given that the platform's primary goal is a decentralized learning model, a proper connection method is imperative to maintain the quality and continuity of the process. It is important to note that all connectivity procedure is handled by the `FLConnection` component in the Aggregator Stack. IBM's platform supports several connection methods, such as Flask web framework and WebSockets, which is the chosen method for this work's connection handling;
2. **Protocol handler**: For the parties to be able to communicate, message exchange is necessary, and this component is responsible for controlling this message. The message set comprises a query q, a model update r, and other sub-components such as party registration. There is a difference between each handler for each stack, given that each stack has different messages to be exchanged;
3. **Data handler**: In several corporate environments, it is possible that the FL training must be executed within different data structures, and therefore, the data handler

component is needed for accessing and pre-processing the dataset D_i for each respective party P_i.

4. **FL training modules**: IBM's FL has embedded machine learning algorithms that can be selected by configuring a `FusionHandler` in the aggregator and `LocalTrainingHandler` in the parties. The remaining components are:
   a) `FusionHandler`: This particular component is responsible for two key functions: generating queries via Q and fusing model updates using F. To facilitate message exchange and interpret party replies, the `ProtocolHandler` provides a set of APIs for sending and receiving messages. Ultimately, the goal is to aggregate the model updates and produce a final result using the F function;
   b) `LocalTrainingHandler`: Specifies the $\mathscr{L}$ function that will generate the model updates and ship them to the aggregator;
   c) `FLModel`: The purpose of this module is to create a standardized API for model training, saving, evaluation, updating, and generating model updates that can be used with various machine learning libraries, such as *Keras* or *scikit-learn*.

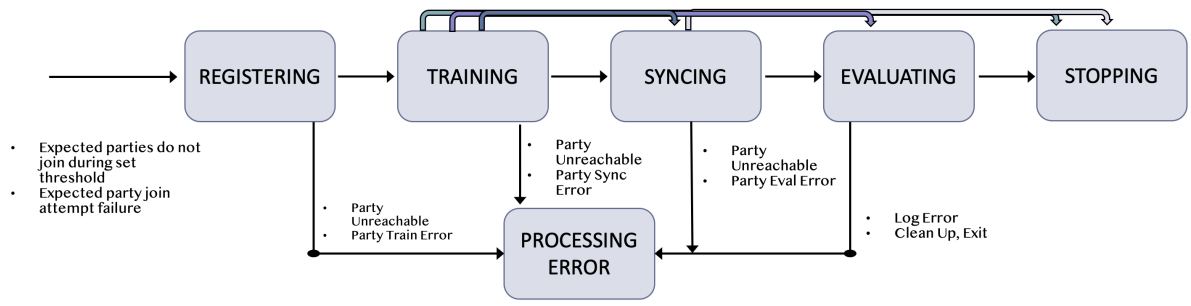
3.1.3 Aggregator Stack

When treating the stack as a whole, the aggregator stack possesses the required elements for the aggregator A to operate. Its main functions are coordinating the FL process, executing the function F, the fusion algorithm, and forwarding the meta-data of the FL process (LUDWIG et al., 2020).

Additionally, the aggregator offers an interface with the commands that the user can enter. It has six main phases, which are *REGISTERING, TRAINING, SYNCING, EVALUATING, STOPPING* and *PROCESSING ERROR*.

Figure 12 depicts the aggregator and all its possible phases:

Figure 12 – Architecture stack of IBM's federated learning



Source: Ludwig et al. (2020, p.8).

3.1.4 Party Stack

Similar to the aggregator stack, the party stack has a unique connection, protocol handler, data handler, model, and local training. However, in this case, the `DataHandler` is mandatory for all parties (LUDWIG et al., 2020).

For a party to join the FL, it must issue a *REGISTER* command provided in the user interface. As soon as the registration is complete, the party waits for a message from the aggregator with a query q to start running the `LocalTrainingHandler`.

Subsequently, $\mathscr{L}$ is executed based on the query q and produces a reply r. The r is sent back to the aggregator. The process is repeated until the aggregator sends a stop request to the FL process (LUDWIG et al., 2020).

3.2 DEPLOYMENT

The central concept of FL revolves around establishing communication among several silos or devices with a central aggregator. However, the selected FL library has several dependencies (that often require other external dependencies) that need to be previously installed for it to function correctly. Additionally, the party devices may be running in different software versions or hardware.

Initially, the proposition is to use edge devices, such as Raspberry Pi devices, as parties. Therefore, a very lightweight deployment solution is required. The main options discussed were virtual machines and containers. Containers were created to be lighter than virtual machines since they only share the OS (Operating System) kernel instead of a full copy of the OS (SULTAN; AHMAD; DIMITRIOU, 2019). Hence, for the desired application, container deployment is the fit alternative.

Several tool managers like Docker, LXC, and RKT are available to manage and deploy containers. Docker is consolidated as the predominant container runtime environment and, therefore, has the most documentation and research available (SULTAN; AHMAD; DIMITRIOU, 2019). Consequently, it is the chosen runtime environment for the project.

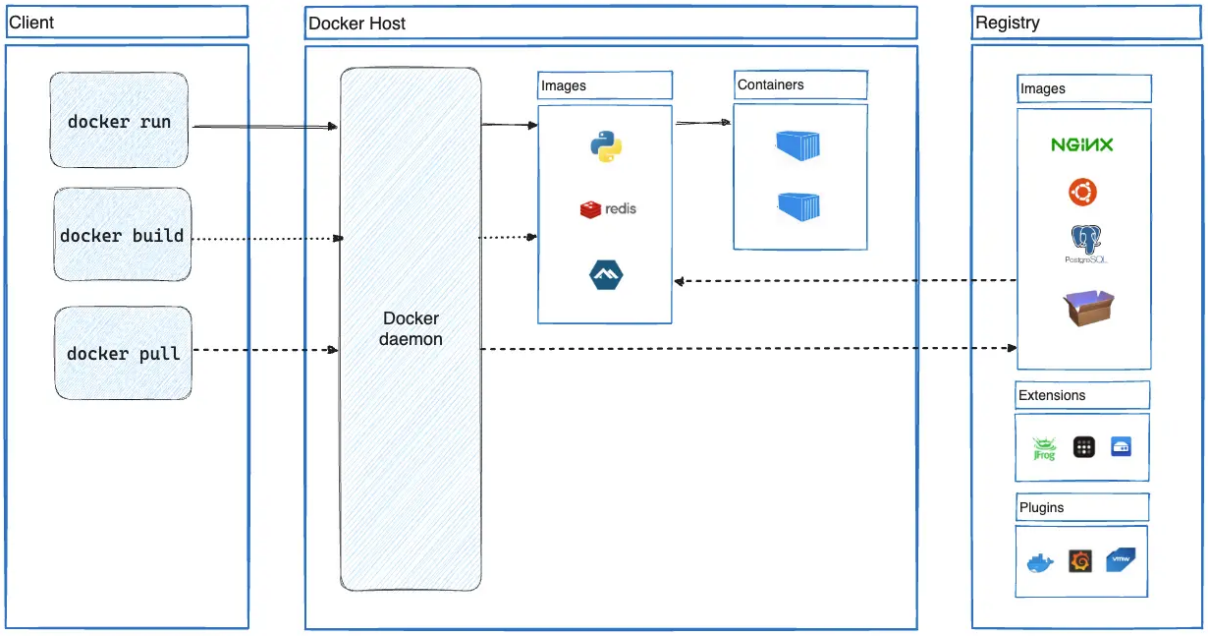
3.2.1 Docker

Docker is an open platform for developing, shipping, and deploying an application. It is written in the Go programming language [1] and uses several Linux features to perform its tasks. It also uses a technology named `namespaces` that isolates the container from the host system (DOCKER INC., 2023).

[1] The Go programming language was created around 2008 by Rob Pike, Robert Griesemer, and Ken Thompson at Google. The project was conceived due to the dissatisfaction with the C++ and Java programming languages that were being used in Google at that time (MEYERSON, 2014).

Docker uses a client-server architecture that is depicted in Figure 13. The client talks to the Docker daemon [2], who is responsible for building, running, and distributing the containers. The client and daemon communicate through REST API over UNIX sockets or a network interface (DOCKER INC., 2023).

Figure 13 – Docker architecture



Source: Docker Inc. (2023)

3.3 THIRD-PARTY EVALUATION TOOLS

The Keras library was utilized to acquire the desired metrics for the final model. Keras is a high-level neural networks API that was originally developed in Python and can run on top of several frameworks such as TensorFlow or Theano (ARNOLD, 2017). It is one of the most popular neural network APIs because it supports all available neural network models and is built to operate in modules, granting flexibility and speed in research.

3.4 EXPERIMENT

The experiment itself consisted of training a model using a subset of the MNIST dataset in each of the containerized silos and, afterward, evaluating the updated model results.

[2] The daemon is a background process that handles Docker API requests and Docker objects such as images, containers, networks, and volumes 13.

3.4.1 Hardware

The main components of a FL platform are the main server, which contains the aggregator and its respective clients' devices. As seen in 1, the IBM FL library supports horizontal data partitioning and a cross-silo device. Therefore, two devices were selected as silos. Chart 2 illustrates the relevant hardware configuration for each silo.

Chart 2 – Hardware comparison.

Hardware Description	**IPT-N-3011**	**IPT-N-0007**
HDD [TB]	1	1
RAM Memory [GB]	4	8
Processing Unit	4-core @ 2.9 GHz	4-core @ 2.9 GHz
Linux Distribution	Ubuntu 18.04	Ubuntu 18.04

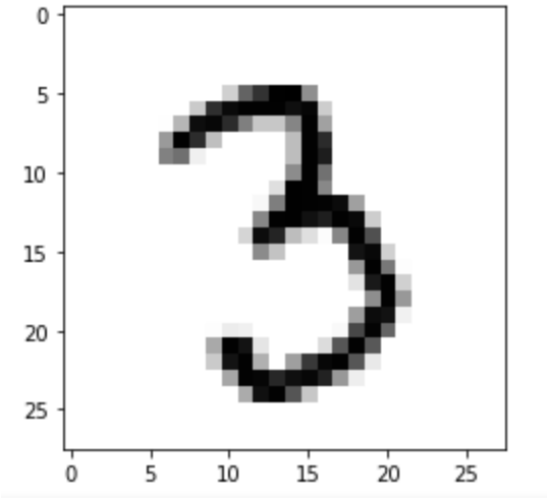
Source: Author (2024)

Due to limited availability, the aggregator was run on several different computers within the Fraunhofer network. Therefore, it is fruitless to describe each hardware configuration individually. However, since the training is done locally in each client, the model is transmitted through an internet connection and the dataset used is relatively small, the aggregator hardware will not significantly impact the final results.

3.4.2 Dataset

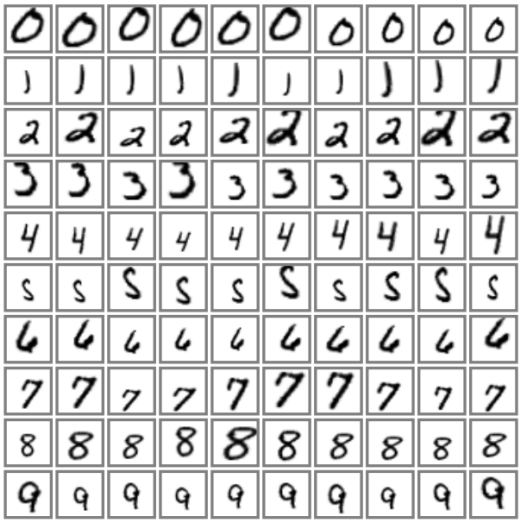
The objective of this work is to deploy the platform. Therefore, a common and easy-to-use dataset is ideal to test the platform. The Modified National Institute of Standards and Technology (MNIST) dataset comprises a collection of 28x28 pixel grayscale images of handwritten digits (0 through 9) and corresponding labels indicating the digit they represent. It is commonly used as a test case for several machine learning algorithms (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, 2017). Figure 14a depicts one image of the handwritten digit *3*, whereas Figure 14b represents several examples for all digit classes.

Figure 14 – MNIST digits



(a) MNIST digit *3* in the pixel scale.

Source: Chris Albon (2019)



(b) Subset of handwritten digits for all classes.

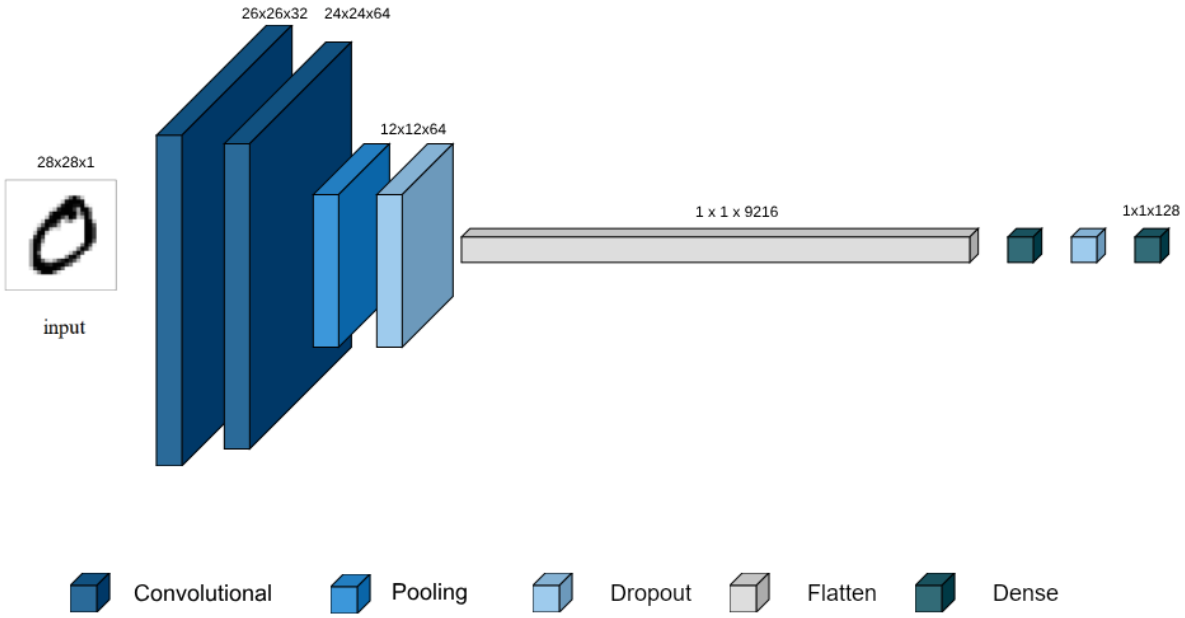
Source: Lecun et al. (1998, p. 11)

The IBM FL library provides an algorithm that selects a random subset of images for each party with a desired number of data points. The generated file is then ready to be used in the training and testing procedure. Furthermore, 2000 images were used for training, and 5000 were used for testing.

3.4.3 Training Model

The training model used in the experiment is a Keras CNN classifier, a built-in feature in the FL library. Since this model aims to test the platform, the suggested model in the IBM guide was used. It was configured to possess two convolutional layers with activated ReLu, one pooling layer, one dropout, flatten, and two dense layers to compose the fully-connected layers segment. Its architecture is depicted in Figure 15.

Figure 15 – Custom keras CNN detailed architecture



Source: Author (2024)

3.4.4 FL Parameters

To explore the platform's capabilities, two categories of tests were conducted and each test was repeated five times. The main difference between the tests was the FL strategy, both FedSDG and FedAvg, used to get the results. Therefore, the constants are model, data points, data set, epochs, rounds, and termination accuracy. Table 1 shows the training parameters and their values. It is worth noting that both parties had the same amount of data points from different random subsets of the MNIST data set.

Table 1 – Parameters used

Parameters	**Values**
Model	Custom Keras CNN
Dataset	MNIST
Training data points	2000
Testing data points	5000
Parties	2
Learning Rate	0.01
Rounds	20
Epochs	20
Termination Accuracy	0.9

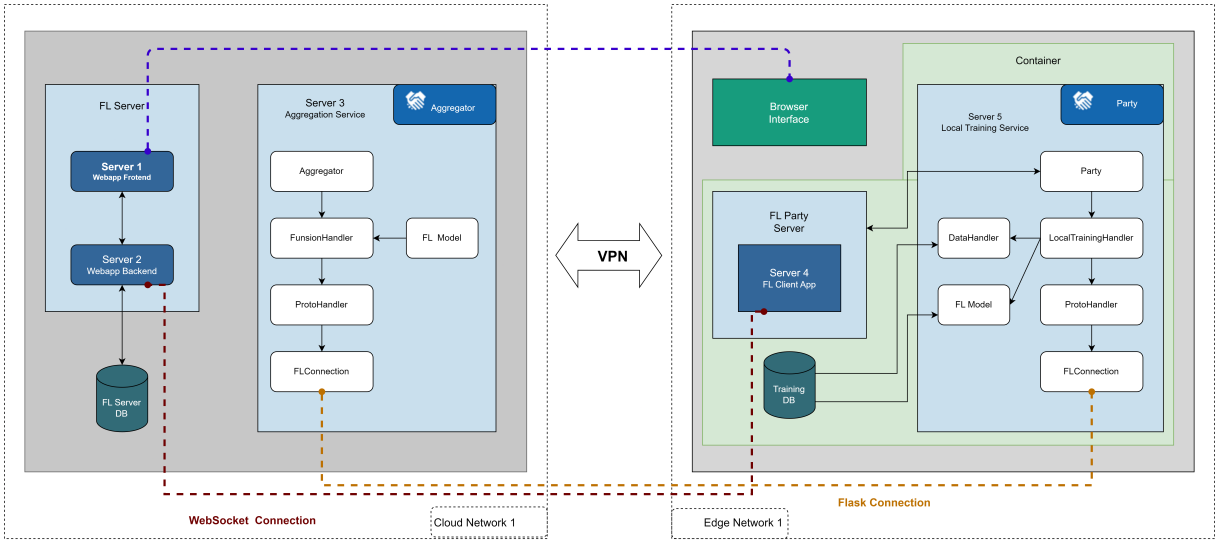
Source: Author (2024)

3.4.5 Network Configuration

Since the party devices are meant to be geographically apart, and the aggregation service runs in the Fraunhofer cloud network, the parties must connect to the cloud to send the trained model. Therefore, a private VPN connection creates a tunnel from the local network to the edge network.

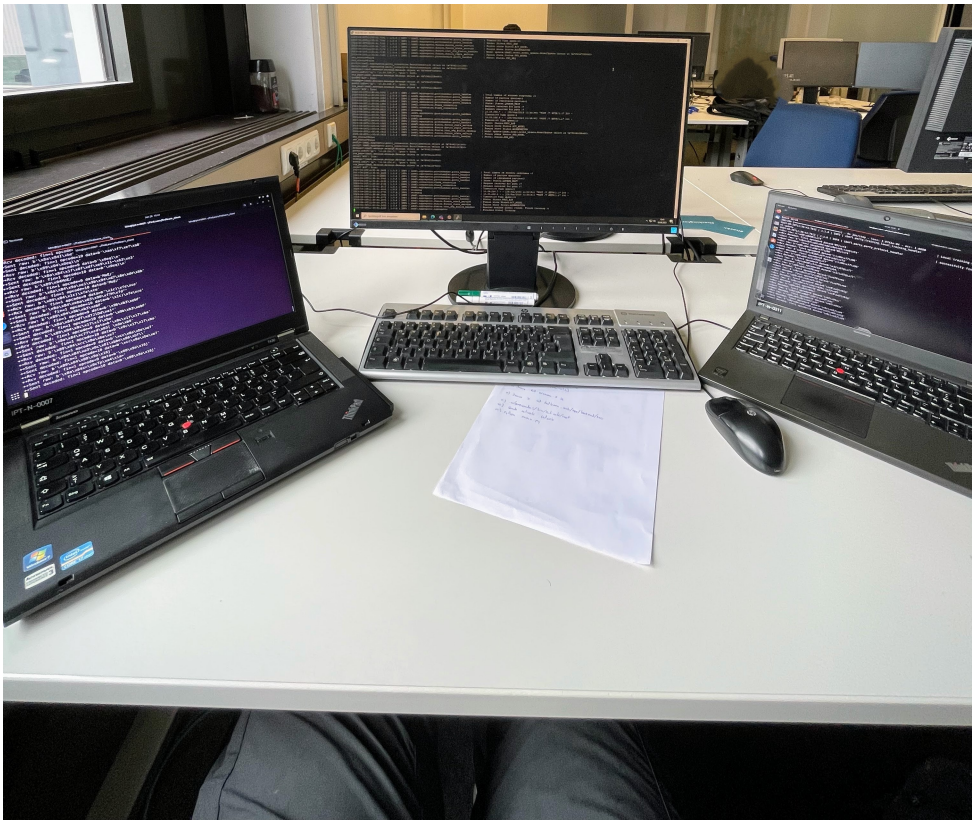
A WebSocket connection between the FL Client App and the Web App backend is also established. Furthermore, a Flask connection is established between the aggregation and local training services. Figure 16 depicts these connections and the final project architecture.

Figure 16 – Overall FL architecture



Source: Adapted from Rosa et al. (2023, p.7)

Figure 17 – Physical devices setup



Source: Author (2023)

3.4.6 Evaluation Criteria

A few criteria were set to be analyzed to test whether the platform was working. In terms of hardware, it was chosen to monitor the CPU and RAM usage since the initial application envisioned using edge devices. Additionally, the network traffic was also studied to evaluate whether the parties were sending the updates accordingly.

In terms of model training, threshold metrics were used. A threshold metric quantifies classification prediction errors, meaning it is a fraction or ratio of a class classification and its expected classification. They are usually used when it is expected to minimize the number of errors (BROWNLEE, 2021). The chosen ones were accuracy, loss, precision, recall, F1 score, and the confusion matrix for each global model.

Accuracy is the most common threshold metric and is described by Brownlee (2021) as Equation 8:

$$Accuracy = \frac{CorrectPredictions}{TotalPredictions} \tag{8}$$

Precision is the rate of positively assigned cases divided by the number of positive examples labeled by the system whereas recall measures the positively assigned cases divided by the number of positive examples in the data set and the F1 Score is the combination of precision and recall through a harmonic mean of both metrics (SOKOLOVA; LAPALME, 2009).

Although the accuracy and recall metrics are important, it is convenient to use precision as the guideline when it is desired to minimize the number of false positive errors. On the other hand, it is useful to use recall as a guideline when false negatives are more critical (BROWNLEE, 2021).

However, both precision and recall can be misleading when analyzed by themselves. Therefore, the F1 Score metric was also chosen, which is one of the most common metrics for imbalanced classification problems (BROWNLEE, 2021). Brownlee (2021) describes these metrics for multiclass classification according to Equations 9, 10 and 11.

$$Precision = \frac{\sum^{c \epsilon C} TruePositives_c}{\sum^{c \epsilon C} TruePositives_c + FalsePositives_c} \tag{9}$$

$$Recall = \frac{\sum^{c \epsilon C} TruePositives_c}{\sum^{c \epsilon C} TruePositive_c + FalseNegative_c} \tag{10}$$

$$F1_{score} = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{11}$$

Therefore, it can be stated that an ideal model would have a value of 1 for precision, recall, and F1 Score. However, since the training was executed as a testing method of the platform, an arbitrary threshold of 0.8 or higher was set for those three metrics to be considered a successful training.

4 RESULTS AND DISCUSSIONS

This chapter addresses the achieved results and discusses how they compare to the expected output from the platform setup. For the chapter, data was collected from the first experiment from each FL strategy. The collection was conducted through a Python script using the `psutil` and `time` libraries. The remaining results can be found in Appendix A and Appendix B.

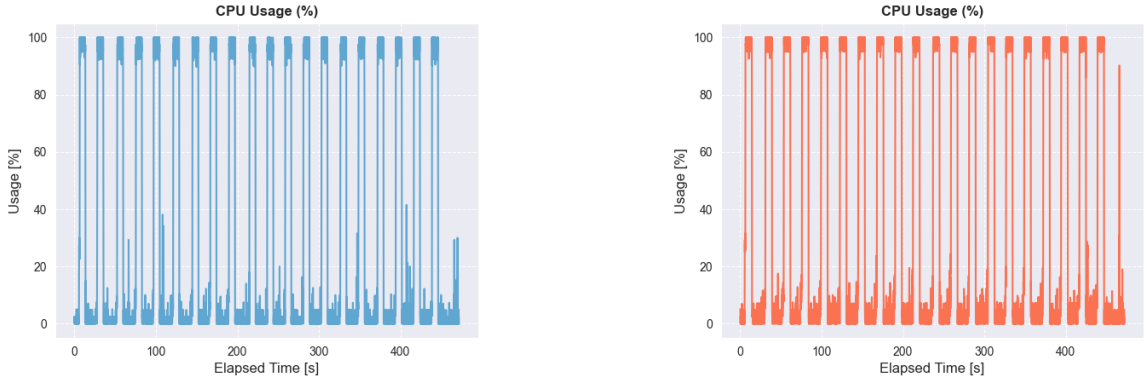
4.1 RESOURCE USAGE

The project's initial proposition was to use edge devices such as Raspberry Pis to perform the role of parties. Therefore, the main concern with the deployment was the resources required to execute a full FL round since edge devices are known for having limited hardware capabilities.

4.1.1 Evaluation of FedAvg Strategy

Figure 18 shows the CPU usage for devices IPT-N-0311 and IPT-N-0007, respectively, while the FedAvg strategy is used during Experiment 1.

Figure 18 – CPU usage for FedAvg strategy in Experiment 1



(a) CPU usage from device IPT-N-0311 while in FedAvg

(b) CPU usage from device IPT-N-0007 while in FedAvg

Source: Author (2024)

Each CPU presents the same 20-peak behavior. Each peak represents one round of the training stage of the model that happens locally in each silo. Therefore, it is possible to conclude that the model training occurred as expected. The training phase lasted 471 seconds for both devices, for the data set containing 5000 testing points, and it used 100% of the CPU power. Hence, it is possible to assume that a regular edge device would take a longer time since it is usually less powerful than a regular CPU.

The RAM has also been analyzed. Analogously, Figures 19 and 20 show the RAM usage for both devices during Experiment 1.

Figure 19 – RAM usage on device IPT-N-0311 with FedAvg in Experiment 1

Source: Author (2024)

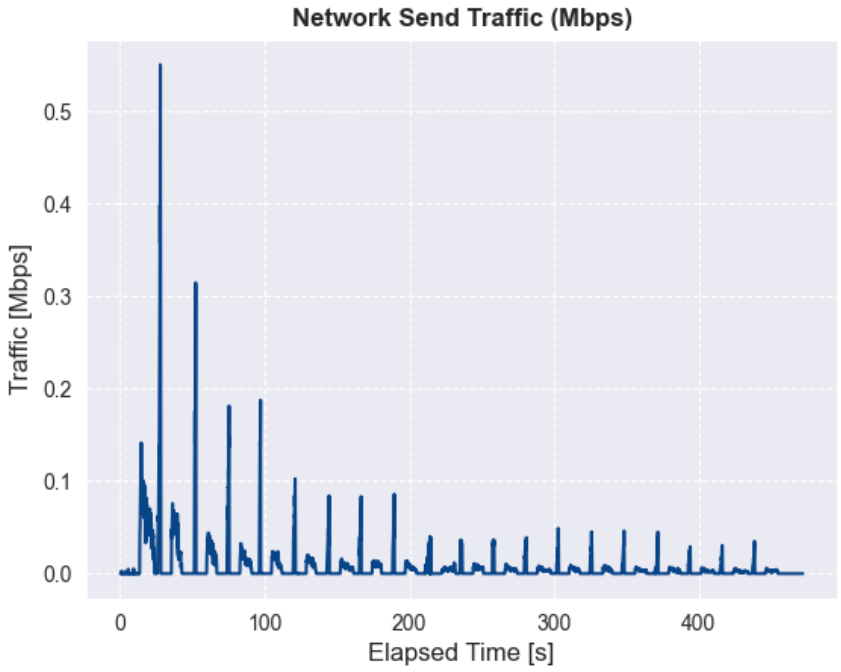
Figure 20 – RAM usage on device IPT-N-0007 with FedAvg in Experiment 1

Source: Author (2024)

Although the behavior in both figures is similar, the scale shows that device IPT-N-0311 uses almost 600 MB/s more RAM than its counterpart party. That can be attributed to device IPT-N-0311 having four extra gigabytes available, hence it can allocate more memory for the training procedure. This behavior, however, may jeopardize the initial proposition of using edge devices as the parties since they have limited RAM resources.

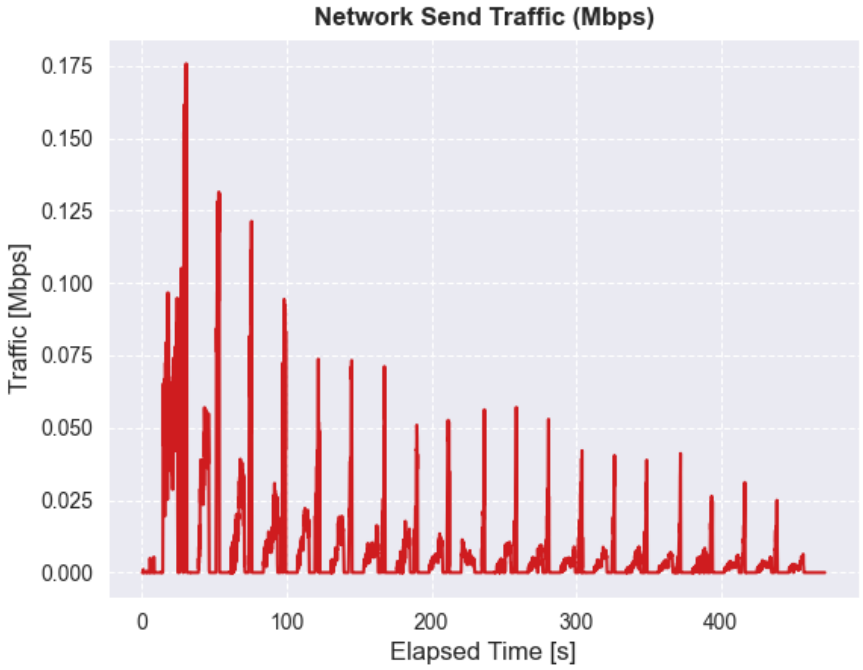
The following criterion to analyze is the network traffic from both parties. Figures 21 and 22 depicts both devices network traffic. Since the training procedure encloses receiving and sending the model parameters, peak patterns like CPU usage are also expected.

Figure 21 – Network traffic on device IPT-N-0311 with FedAvg in Experiment 1



Source: Author (2024)

Figure 22 – Network traffic on device IPT-N-0007 with FedAvg in Experiment 1



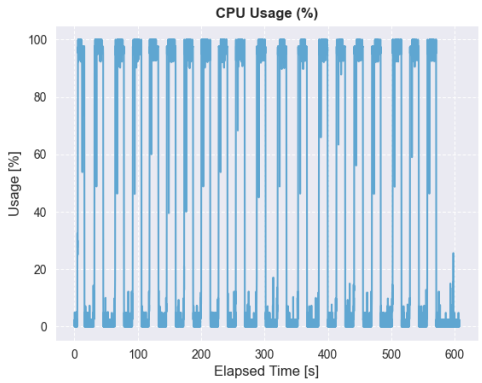
Source: Author (2024)

Both figures show a very distinctive exponential-shaped curve from the network traffic. That can happen due to a phenomenon entitled *communication overhead*, which means that, after every round, the model gets more accurate, which means that some parameters will no longer be altered. Therefore, the size of the exchanged communication keeps getting smaller after every round.

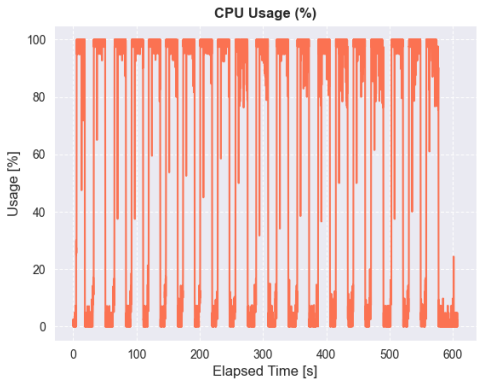
4.1.2 Evaluation of FedSDG Strategy

The same analysis is made when changing the FL strategy to FedSDG. Figure 23 represents the CPU usage from IPT-N-0311 and IPT-N-0007 devices, respectively. The training procedure in Experiment 1 lasted 607 seconds on both IPT-N-0311 and IPT-N-0007 devices.

Figure 23 – CPU usage for FedSDG strategy in Experiment 1



(a) CPU usage on device IPT-N-0311 with FedSDG

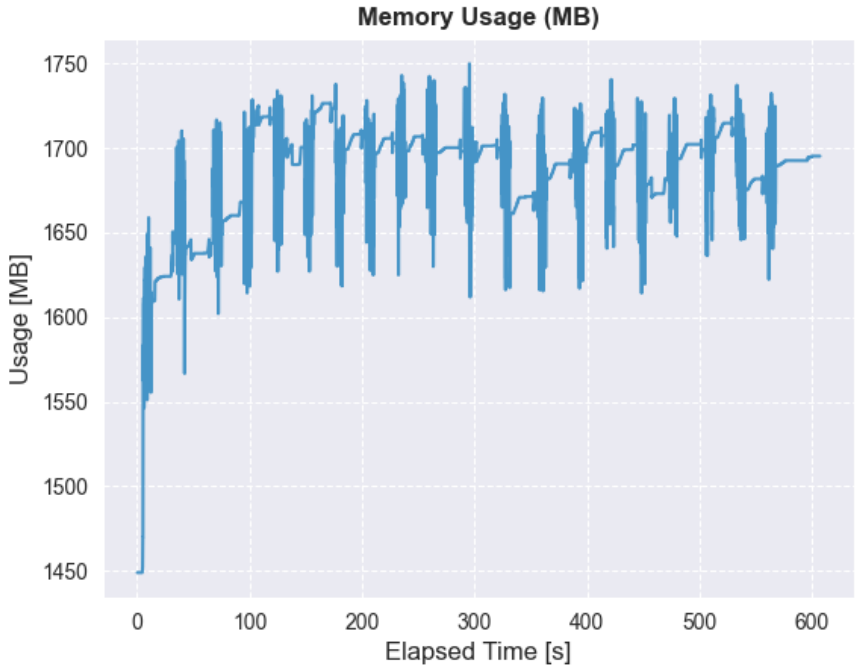


(b) CPU usage on device IPT-N-0007 with in FedSDG

The presented behavior is almost identical to the one observed using the FedAvg strategy. However, since the FedAvg algorithm is a variation of the FedSDG, a similar behavior was expected.

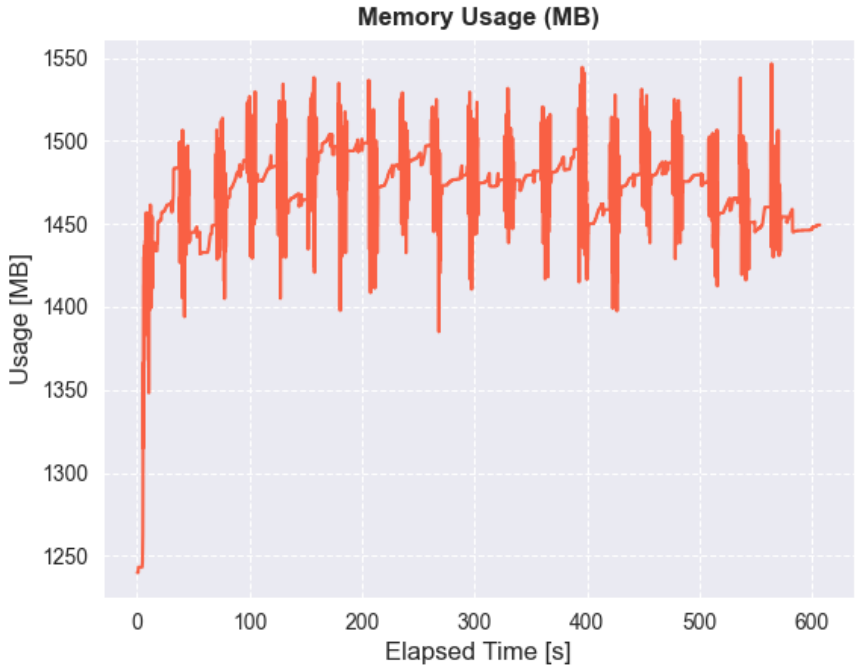
RAM usage in Experiment 1 has also been tracked and is shown in Figures 24 and 25 from devices IPT-N-0311 and IPT-N-0007, respectively.

Figure 24 – RAM usage from device IPT-N-0311 while in FedSDG



Source: Author (2024)

Figure 25 – RAM usage from device IPT-N-0007 while in FedSDG

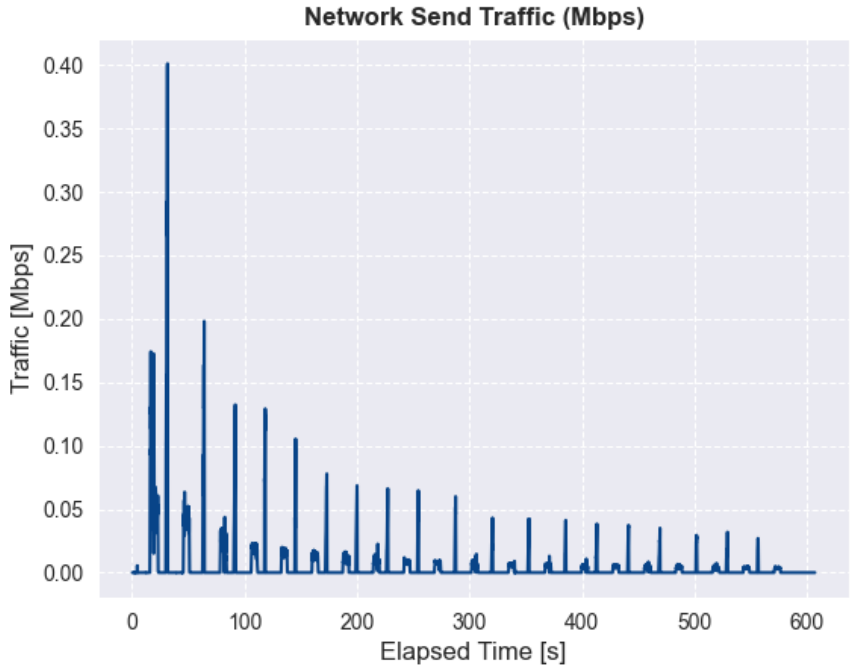


Source: Author (2024)

Once again, the observed behavior is similar to the one observed within the FedAvg strategy. However, compared to Figure 19, the first FedSDG experiment required approximately 650 MB less RAM for device IPT-N-3011 and 300 MB less for device IPT-N-0007. All other FedSDG experiments are within the same margin of RAM usage and exhibit a similar pattern to the ones shown in Figures 24 and 25 as seen in Appendix B.

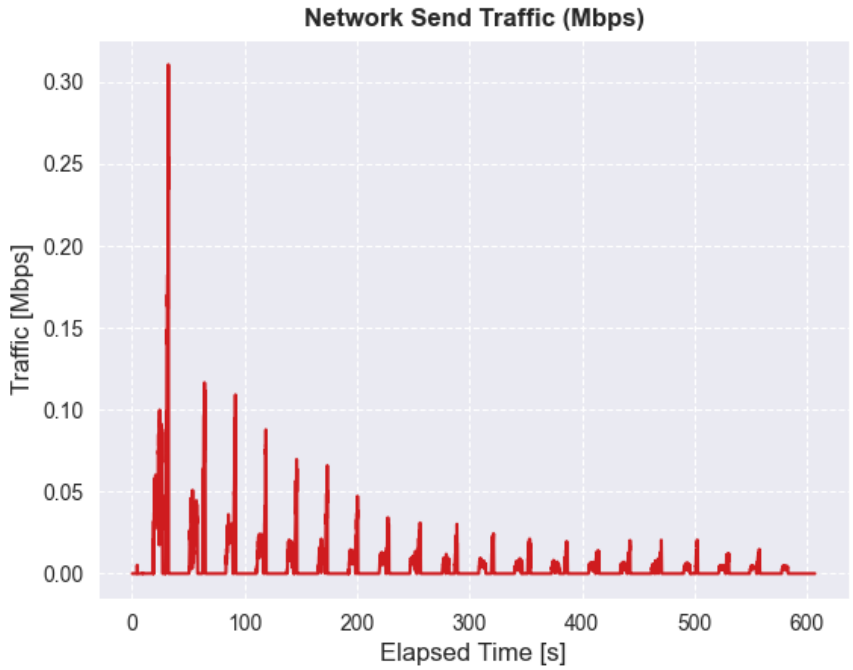
Ultimately, the network traffic for Experiment 1 is analyzed, and the results are shown in Figures 26, 27 for both IPT-N-0311 and IPT-N-0007 parties, respectively.

Figure 26 – Network traffic from device IPT-N-0311 while in FedSDG



Source: Author (2024)

Figure 27 – Network traffic usage from device IPT-N-0007 while in FedSDG



Source: Author (2024)

The network traffic again represents a similar pattern to the one observed in the FedAVg. For device IPT-N-0311, the Mbps value is close for both FL strategies, whereas for device IPT-N-0007, the value observed using the FedSDG is almost double

that observed using FedAvg.

Among the hypotheses is that all models are randomly initialized. Therefore, they can already have some optimization during the weight randomization process, which could explain why less information was changed from party IPT-N-0007. Additionally, all other experiments had different traffic values sent, which supports the model randomization hypothesis. Appendix B contains the remaining network traffic results from the other experiments using the FedSDG strategy.

4.2 TRAINING RESULTS

To test the platform, it is imperative to evaluate whether the model was properly trained on each device. Therefore, the final global model was exported and evaluated with a different subset of 5000 data points randomly selected by the IBM sorting algorithm.

4.2.1 Evaluation of FedAvg Strategy

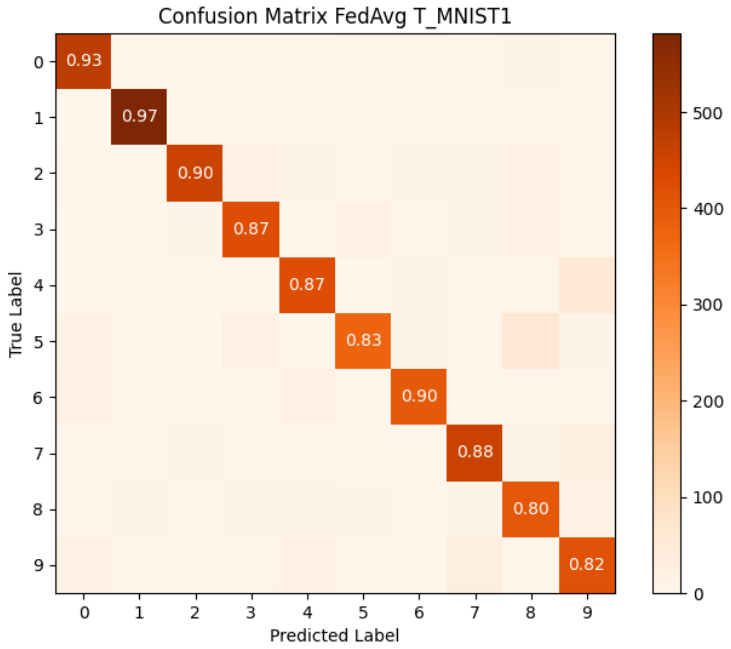
The desired metrics are shown in Table 2, and the confusion matrix for Experiment 1 is exhibited in Figure 28.

Table 2 – Global model metrics achieved with FedAvg strategy

Metrics	**Experiment 1**	**Experiment 2**	**Experiment 3**	**Experiment 4**	**Experiment 5**
Accuracy	0.8840	0.8782	0.8828	0.8806	0.8774
Loss	0.7848	0.8463	0.8284	0.8835	0.8283
Precision	0.8817	0.8728	0.8763	0.8771	0.8713
Recall	0.8734	0.8696	0.8706	0.8698	0.8662
F1 Score	0.8820	0.8755	0.8808	0.8781	0.8747

Source: Author (2024)

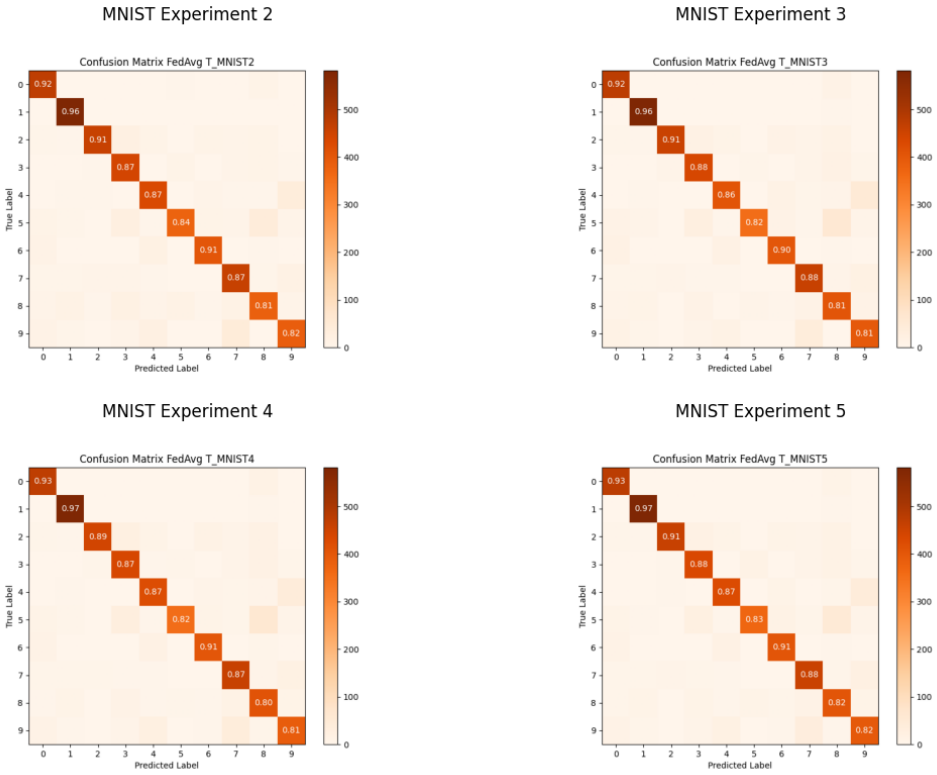
Figure 28 – Experiment 1 confusion matrix using FedAvg with F1-Scores



Source: Author (2024)

Figure 29 shows the confusion matrices of the remaining training experiments with the F1-Score in its diagonals.

Figure 29 – Confusion matrices using FedAvg with F1-Scores



Source: Author (2024)

When observing Table 2, it can be seen that the F1 Score and Accuracy for all five experiments were over 0.87, which is within the designated threshold and, therefore, can be considered a good classification model for the project. Most models training with the MNIST dataset achieve an accuracy of 98% (GUPTA, 2020). However, since a small fragment of the dataset was used and a simple CNN architecture was implemented, the concern for accuracy improvement was neglected.

In Figure 28, since the diagonal squares are clearly defined, it visually represents the quality of the model achieved in Experiment 1 and that classes 0 and 1 had the best predictions. Figure 29 shows that the other models behave almost identically to the one observed in Figure 28, especially in some incorrectly classified classes, such as class 8 and class 9. That is expected since all experiments were performed in the same way.

Furthermore, Table 3 shows the maximum variation in percentage for all metrics during all experiments. It can be observed that the variation for the F1 Score was less than 1%, which demonstrates the model's and platform's consistency.

Table 3 – Maximum difference for global model Metrics with FedAvg strategy

Metrics	Max Difference (%)
Accuracy	0.75 %
Loss	6.39 %
Precision	1.18 %
Recall	0.83 %
F1 Score	0.83 %

Source: Author (2024)

The individual values of precision, recall, and F1 Score for each class can be found in Appendix C.

4.2.2 Evaluation of FedSDG Strategy

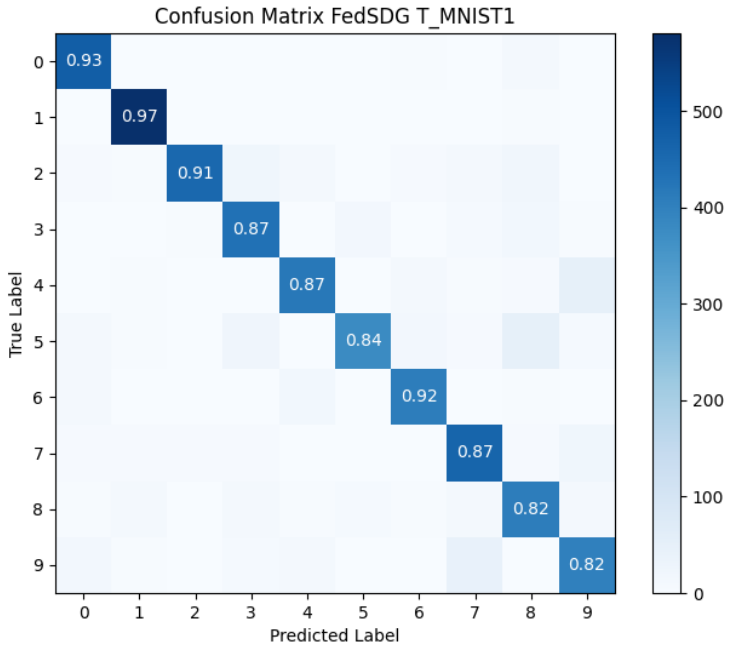
The exported metrics are shown in Table 4, and the confusion matrix for Experiment 1 is depicted in Figure 30.

Table 4 – Global model metrics achieved with FedSDG strategy

Metrics	Experiment 1	Experiment 2	Experiment 3	Experiment 4	Experiment 5
Accuracy	0.8802	0.8814	0.8772	0.8768	0.8838
Loss	0.7647	0.9040	1.0133	1.0645	1.0040
Precision	0.8813	0.8814	0.8793	0.8796	0.8838
Recall	0.8781	0.8792	0.8751	0.8750	0.8819
F1 Score	0.8781	0.8788	0.8746	0.8744	0.8813

Source: Author (2024)

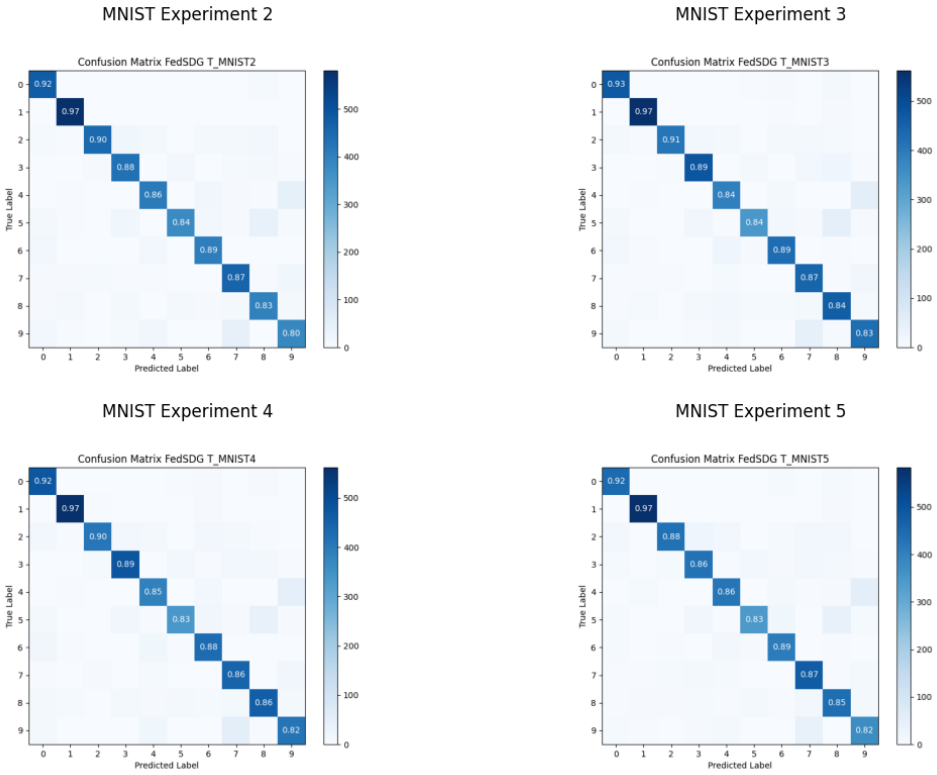
Figure 30 – Experiment 1 confusion matrix using FedSDG with F1-Scores



Source: Author (2024)

Figure 31 shows the confusion matrices of the remaining training experiments with the F1-Score in its diagonals.

Figure 31 – Confusion Matrices using FedSDG with F1-Scores



Source: Author (2024)

Similarly to the FedAvg experiment, when observing Chart 4, it can be stated that all metrics are also within the established threshold. When looking at Figure 30, the diagonal squares are visibly stated, and classes 1 and 3 had the best predictions. It can also be seen that classes 5, 7, 8, and 9 had the most errors. Figure 31 depicts the same pattern observed in Experiment 1, even for the best and worst predictions, which is expected since all experiments were performed identically.

Furthermore, Table 5 depicts the maximum difference in the percentage of all metrics for all experiments. As observed for the FedAvg, the FedSDG also shows a maximum v variation of less than 1%, solidifying the model and the fusion algorithm capabilities.

Table 5 – Maximum difference for global model Metrics with FedSDG strategy

Metrics	**Max Difference (%)**
Accuracy	0.79 %
Loss	31.41 %
Precision	0.51 %
Recall	0.78 %
F1 Score	0.78 %

Source: Author (2024)

Appendix D provides the individual values of precision, recall, and F1 Score for

each class.

4.2.3 Runtime

When considering the scaling options for industry applications, the training runtime is a great feasibility indicator since the goal in any industrial process is to minimize the time spent in production. Table 6 depicts the training time for all training procedures using the FedAvg strategy.

Table 6 – Training time with FedAvg strategy

Experiment #	**Elapsed Time [s]**	
	IPT-N-0007	IPT-N-0311
1	471.67	471.86
2	467.69	467.68
3	473.12	473.11
4	475.45	475.54
5	487.48	487.54

Source: Author (2024)

It is possible to see that the training intervals for both parties were almost identical. Two main factors can explain that. The first is the party's device similarities. Therefore, the training is expected to exhibit similar behavior in both clients. Moreover, the second factor is that the tests were all performed in the same geographic environment, meaning that both of them had the same internet connection available and, therefore, very similar communication feedback. Analogously, Table 7 shows the time for the FedSDG strategy.

Table 7 – Training time with FedSDG strategy

Experiment #	**Elapsed Time [s]**	
	IPT-N-0007	IPT-N-0311
1	607.15	607.17
2	576.46	574.54
3	578.30	578.34
4	591.14	591.10
5	590.04	590.10

Source: Author (2024)

For the FedSDG strategy, the same time similarity pattern is observed among the clients, which is expected. However, there is a difference between the training time using the FedSDG strategy that is, on average, 23% longer than the FedAvg strategy. Since the FedSDG works with gradients instead of weights, that kind of information implies slightly larger data to be transmitted to and from the aggregator, resulting in this delay.

4.2.4 Takeaways

To illustrate a direct comparison between both strategies, the metrics' mean average for all experiments is shown in Table 8.

Table 8 – Metrics mean average

Metrics	FedAvg	FedSDG
Accuracy	0.8806	0.8799
Loss	0.8343	0.9501
Precision	0.8758	0.8811
Recall	0.8699	0.8778
F1 Score	0.8782	0.8774

Source: Author (2024)

Table 8 shows that FedSDG yielded a slightly better result than FedAvg. However, all metrics are close to their counterparts. Therefore, it can be stated that both strategies yielded a relatively similar model after the training procedure. It is worth noting that the only plausible difference is seen within the loss parameter. Since FedAvg differs from FedSDG in aggregating weights versus aggregating gradients, this algorithm change can impact the convergence behavior of each model.

When comparing the experiments individually from both Table 2 and Table 4, the same pattern of slightly better performance and also a greater loss value from FedSDG is observed in each experiment.

5 CONCLUSIONS

This work aimed to deploy a federated learning library into an industrial platform and test its functionalities so that it can later be adapted to train with data acquired and transmitted from multi-sensory devices. To achieve the main objective, the library and several other features, such as the dataset itself, were containerized and deployed in each device that worked as a client in the training process through a Docker container.

Afterward, the Websocket and Flask connections were established, and minor configurations, such as export model function buttons, were added. With the deployment process completed, a set of five different global model training was executed using two different FL strategies, the FedAvg and FedSDG.

When the project was first conceived, the main suggestion was to use edge devices as clients, specifically two 3rd-generation Raspberry Pis, since they are cheaper than regular computers and, therefore, more attractive to the industry and its everyday struggle to minimize cost. However, after two months of testing and research, the conclusion came that the IBM FL library (version 1.1.0) was not fully compatible with the ARM architecture present in most edge devices. The same conclusion was found within IBM's GitHub repository issues tab [1].

Additionally, the library suggested using the Cent OS image to run it inside the container due to its strong encryption and cybersecurity capabilities. Unfortunately, the image also lacked support for ARM architectures and was discontinued in 2021, which dashed hopes for any further assistance.

To contour this challenge, two ThinkPads with Arch x86 architectures were replaced as the clients. The deployment ran successfully for them and the training results proved that the platform works according to the initial expectations.

When comparing the employed FL strategies, both yielded a model with similar classifying characteristics, showing hits and errors in the same classes. Hence, regarding the final global quality, it is clear that both FedAvg and FedSDG strategies can be used in the platform to provide a good standard for the model.

Nevertheless, when comparing the strategies hardware-wise, the results have shown that the FedSDG strategy consumes higher network bandwidth, ranging from 30% to almost 70% more, depending on the analyzed experiment. Moreover, the FedSDG took an average of 1,8 more minutes to finish the training procedure than the FedAvg. For a test training with only two parties, two minutes does not impact the final results. However, the whole concept of FL is to be able to train on dozens or hundreds

[1] The issues can be found at https://github.com/IBM/federated-learning-lib/issues/97 and https://github.com/IBM/federated-learning-lib/issues/74

of devices, meaning this network consumption can exceed the available network traffic and impact on several more minutes of delay.

Consequently, since the output model yields similar results for both strategies and the FedAvg consumes fewer network resources and finishes the process faster, for this classification scenario, the FedAvg strategy is ideal for the application.

For future studies, a revision of this approach using more parties, such as five clients, would prove useful in testing the network traffic and any possible bottlenecks that may arise. For instance, multiple nodes may want to connect simultaneously to the aggregator, slowing down the training process. This is one of the main challenges for the federated learning approach (ROSA et al., 2023).

Furthermore, testing with more complex models using real-case scenario datasets would also be suggested as a future study, to better evaluate the platform's conditions and training strategies in a possible industrial context.

Although the testing was a proof of concept made with only two parties, when extending to dozens of clients, some other challenges may appear such as the heterogeneity of data. Since each company is physically apart, the way data is sampled can vary from party to party. However, achieving a good performance model requires similar data sets in all parties (ROSA et al., 2023). Therefore, creating a pipeline that sorts and feeds data to the platform would also be a future project worth working on.

As the project unfolded, the federated learning approach seemed even more fitting since one company could train a model with data from multiple branches, enhancing the predictive maintenance schedule and mitigating production costs. Moreover, this approach allows the company to retain all data within its storage infrastructure, thereby ensuring the privacy and governance of each client's data. Annex A contains an article derived from this work.

REFERENCES

ANYOHA, R. **The history of artificial intelligence**. Harvard University, 2020. Disponível em: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/. Acesso em: 11 mai. 2023.

ARNOLD, T. kerasr: R interface to the keras deep learning library. **J. Open Source Softw.**, v. 2, p. 296, 2017. Disponível em: https://www.theoj.org/joss-papers/joss.00296/10.21105.joss.00296.pdf. Acesso em: 15 mar. 2024.

BEHERA, P. K. Leverage of multiple predictive maintenance technologies in root cause failure analysis of critical machineries. **Procedia Engineering**, 2016.

BROWNLEE, J. (Ed.). **Imbalanced Classification with Python**: Choose better metrics, balance skewed classes, and apply cost-sensitive learning. Machine Learning Mastering, 2021. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=jaXJDwAAQBAJ&oi=fnd&pg=PP1&dq=A+gentle+introduction+to+machine+learning+performance+metrics+brownlee&ots=CgHzfOPYYP&sig=ZXDVXS7c6N6vDhG3nhM5Zg051fc#v=onepage&q=A%20gentle%20introduction%20to%20machine%20learning%20performance%20metrics%20brownlee&f=false. Acesso em: 22 abr. 2018.

CHRIS ALBON. **MNIST Dataset**. 2019. Disponível em: https://chrisalbon.com/Data/MNIST+Dataset. Acesso em: 14 mai. 2024.

DOCKER INC. **Docker overview**. United States of America, 2023. Disponível em: https://docs.docker.com/get-started/overview/. Acesso em: 08 mar. 2024.

FRAUNHOFER INSTITUTE FOR PRODUCTION TECHNOLOGY. **AI-NET-ANIARA**: Beschleunigung des digitalen wandels in europa durch intelligente netzautomatisierung – automatisierung von netzrandinfrastrukturen und -anwendungen mit künstlicher intelligenz. Aachen, 2021. Disponível em: https://www.ipt.fraunhofer.de/de/projekte/ai-net-aniara.html. Acesso em: 18 abr. 2023.

GUPTA, J. **Going beyond 99% — MNIST Handwritten Digits Recognition**. United States of America, 2020. Disponível em: https://towardsdatascience.com/going-beyond-99-mnist-handwritten-digits-recognition-cfff96337392. Acesso em: 01 jul. 2024.

HERMAN, R. L. **Introduction to Partial Differential Equations**. R.L. Herman, 2015. 391-392 p. Disponível em: https://math.libretexts.org/Bookshelves/Differential_Equations/Introduction_to_Partial_Differential_Equations_(Herman)/09%3A_Transform_Techniques_in_Physics/9.06%3A_The_Convolution_Operation. Acesso em: 15 apr. 2024.

HEYDARNOORI, A.; MAVADDAT, F. Reliable deployment of component-based applications into distributed environments. **Third International Conference on Information Technology: New Generations (ITNG'06)**, p. 52–57, 2006. Disponível em: https://ieeexplore.ieee.org/document/1611570. Acesso em: 06 mai. 2024.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998. Disponível em: https://ieeexplore.ieee.org/document/726791. Acesso em: 14 mai. 2024.

LUDWIG, H.; BARACALDO, N. (Ed.). **Federated learning**: a comprehensive overview of methods and applications. Springer International Publishing, 2022. Disponível em: https://doi.org/10.1007/978-3-030-96896-0.

LUDWIG, H. et al. **IBM federated learning**: an enterprise framework white paper. 2020. Disponível em: https://doi.org/10.48550/arXiv.2007.10987. Acesso em: 01 mar. 2023.

MANIAS, D. Making a case for federated learning in the internet of vehicles and intelligent transportation systems. **IEEE Network**, 2021.

MCMAHAN, H. B. et al. **Communication-Efficient Learning of Deep Networks from Decentralized Data**. 2023. Disponível em: https://doi.org/10.48550/arXiv.1602.05629. Acesso em: 10 out. 2023.

MEYERSON, J. H. The go programming language. **IEEE Softw.**, v. 31, p. 100–103, 2014. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6898707. Acesso em: 08 mar. 2024.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **The EMNIST Dataset**. 100 Bureau Drive, Gaithersburg, MD 20899, 2017. Disponível em: https://www.nist.gov/itl/products-and-services/emnist-dataset. Acesso em: 06 mai. 2024.

NBC NEWS. **OPM: 21.5 million people affected by background check breach**. United States of America, 2015. Disponível em: https://www.nbcnews.com/tech/security/opm-hack-security-breach-n389476. Acesso em: 01 mai. 2023.

NGUYEN, D. C. et al. Federated learning for internet of things: a comprehensive survey. **IEEE Communications Surveys Tutorials**, v. 23, n. 3, p. 1622–1658, 2021.

NILSSON, A. et al. A performance evaluation of federated learning algorithms. **Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning**, 2018. Disponível em: https://dl.acm.org/doi/pdf/10.1145/3286490.3286559. Acesso em: 13 out. 2023.

O'SHEA, K.; NASH, R. **An Introduction to Convolutional Neural Networks**. 2015. Disponível em: https://arxiv.org/abs/1511.08458. Acesso em: 15 apr. 2024.

ROCHA, V. F. d.; LÓPEZ, J. **An overview on homomorphic encryption algorithms**. 2019. Disponível em: https://ic.unicamp.br/~reltech/PFG/2018/PFG-18-28.pdf. Acesso em: 15 fev. 2024.

MELO ROSA, G. Laydner de et al. **Architecture for Edge-Based Predictive Maintenance of Machines Using Federated Learning and Multi Sensor Platforms**. 2023. 11 p. Disponível em: https://www.researchgate.net/publication/370986976_Architecture_for_Edge-Based_Predictive_Maintenance_of_Machines_Using_Federated_Learning_and_Multi_Sensor_Platforms. Acesso em: 16 apr. 2024.

SAIDANI, A. A systematic comparison of federated machine learning libraries. Faculty of Information Systems. Technische Universität München, Munich, January 2023. Disponível em: https://wwwmatthes.in.tum.de/file/xjrmj55uaqem/Sebis-Public-Website/-/Master-s-Thesis-Ahmed-Saidani/230115%20Saidani%20Masters%20Thesis.pdf. Acesso em: 05 mar. 2024.

SARMA, K. V. et al. Federated learning improves site performance in multicenter deep learning without data sharing. **Journal of the American Medical Informatics Association: JAMIA**, v. 28, p. 1259–1264, 2021.

SHARMA, P.; SHAMOUT, F. E.; CLIFTON, D. A. Preserving patient privacy while training a predictive model of in-hospital mortality. 2019.

SHEA, R.; LIU, J. Network interface virtualization: challenges and solutions. **IEEE Network**, v. 26, n. 5, p. 28–34, 2012. Disponível em: https://ieeexplore.ieee.org/document/6308072. Acesso em: 27 feb. 2024.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Information Processing Management**, v. 45, p. 427–437, 07 2009. Disponível em: https://www.researchgate.net/publication/222674734_A_systematic_analysis_of_performance_measures_for_classification_tasks. Acesso em: 23 abr. 2018.

SULTAN, S.; AHMAD, I.; DIMITRIOU, T. Container security: Issues, challenges, and the road ahead. **IEEE Access**, v. 7, p. 52976–52996, 2019. Disponível em: https://ieeexplore.ieee.org/document/8693491.

TURING, A. M. Computing machinery and intelligence. **Mind**, Oxford University Press, v. 59, n. October, p. 433–60, 1950.

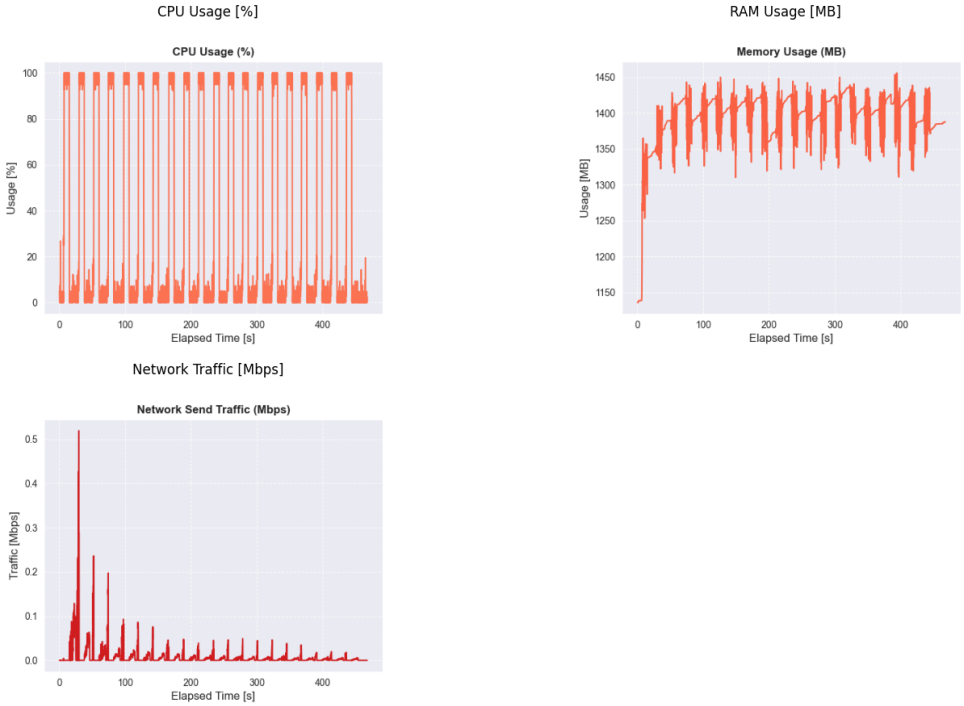
UNITED STATES DEPARTMENT OF HEALTH & HUMAN SERVICES. **Anthem pays OCR $16 Million in record HIPAA settlement following largest U.S. health data breach in history**. United States of America, 2020. Disponível em: https://www.hhs.gov/guidance/document/anthem-pays-ocr-16-million-record-hipaa-settlement-following-largest-us-health-data-breach. Acesso em: 01 mai. 2023.

WAN, Z. et al. Practical and effective sandboxing for linux containers. **Empirical Software Engineering**, v. 24, p. 4034 – 4070, 2019. Disponível em: https://link.springer.com/article/10.1007/s10664-019-09737-2.

APPENDIX A - HARDWARE TESTS RESULTS FOR FEDAVG

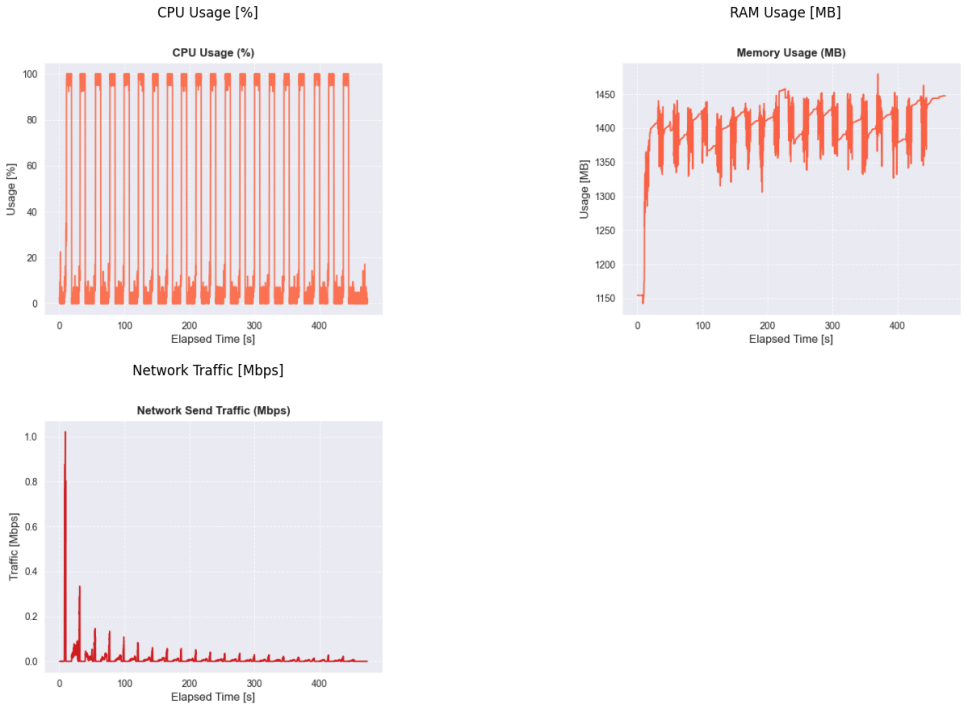
This appendix discloses the remaining hardware test results for CPU and RAM usage and detected network traffic. The obtained results from experiment 1 are similar to the ones found in the remaining experiments. Figures 32, 33, 34 and 35 display experiments 2, 3, 4 and 5 hardware metrics for device IPT-N-0007 while using the FedAvg strategy respectively.

Figure 32 – Hardware metrics for party IPT-N-0007 in experiment 2



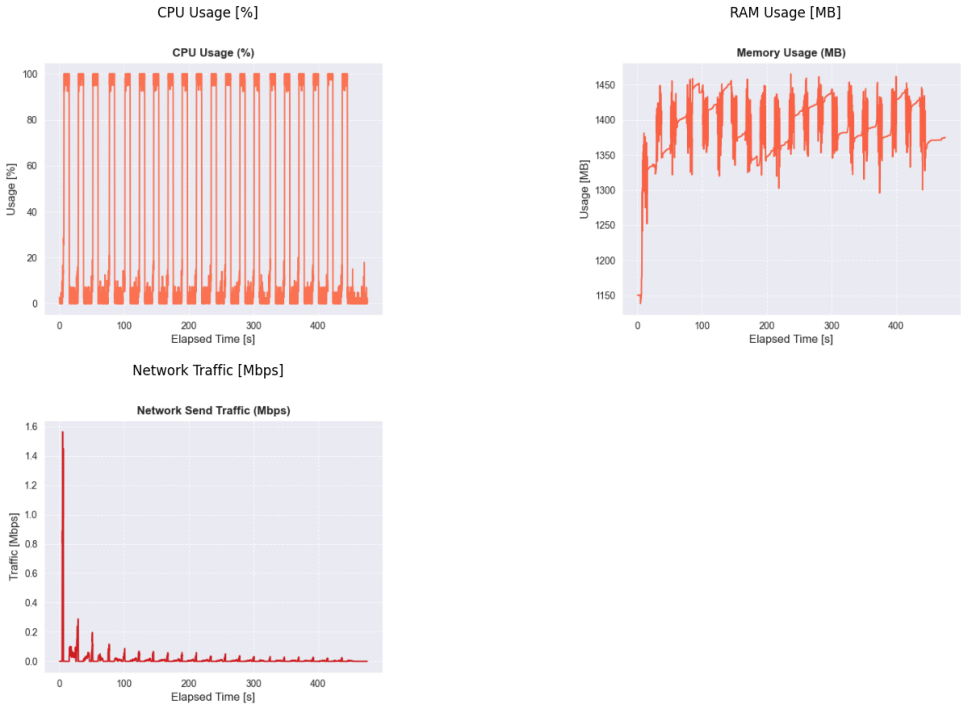
Source: Author (2024)

Figure 33 – Hardware metrics for party IPT-N-0007 in experiment 3



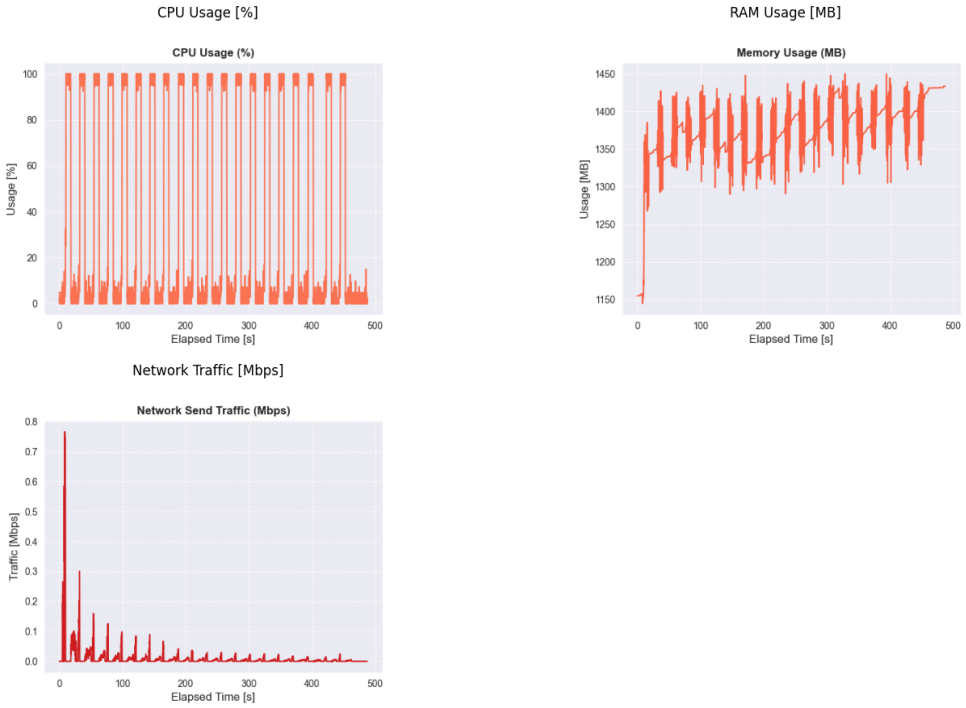
Source: Author (2024)

Figure 34 – Hardware metrics for party IPT-N-0007 in experiment 4



Source: Author (2024)

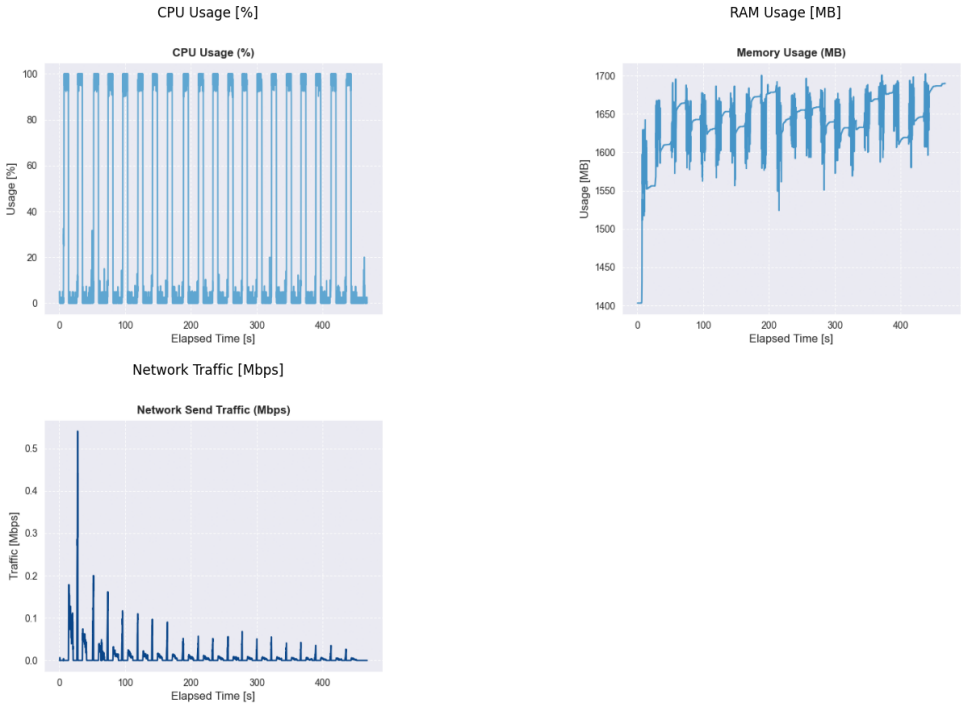
Figure 35 – Hardware metrics for party IPT-N-0007 in experiment 5



Source: Author (2024)

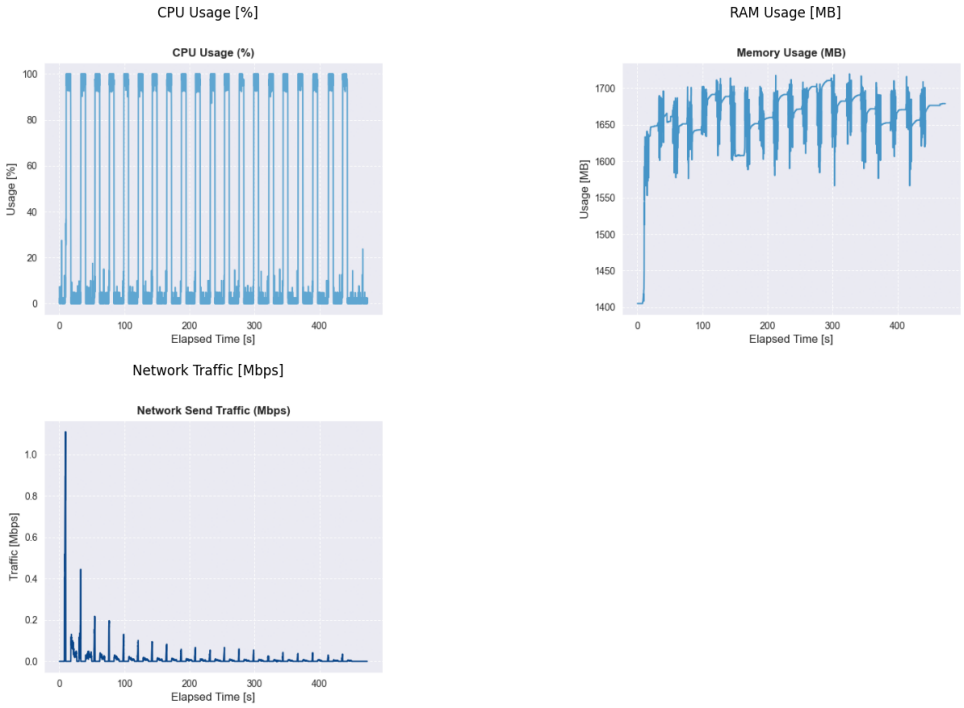
Analogously, Figures 36, 37, 38 and 39, depict the hardware metrics from experiments 2, 3, 4 and 5 for device IPT-N-0311 while using the FedAvg strategy.

Figure 36 – Hardware metrics for party IPT-N-0311 in experiment 2



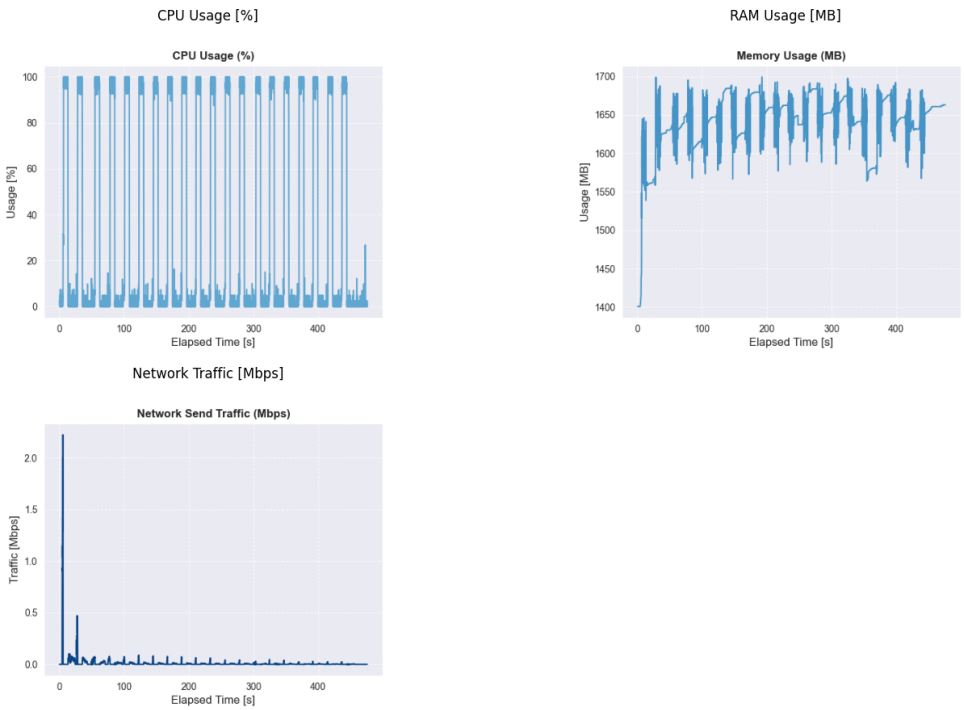
Source: Author (2024)

Figure 37 – Hardware metrics for party IPT-N-0311 in experiment 3



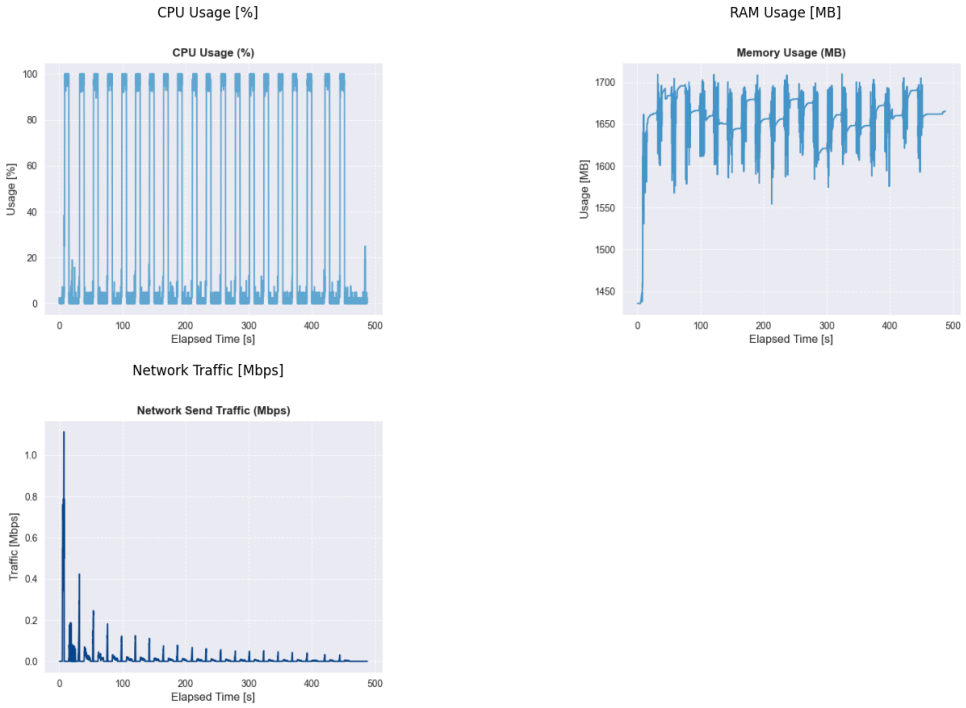
Source: Author (2024)

Figure 38 – Hardware metrics for party IPT-N-0311 in experiment 4



Source: Author (2024)

Figure 39 – Hardware metrics for party IPT-N-0311 in experiment 5

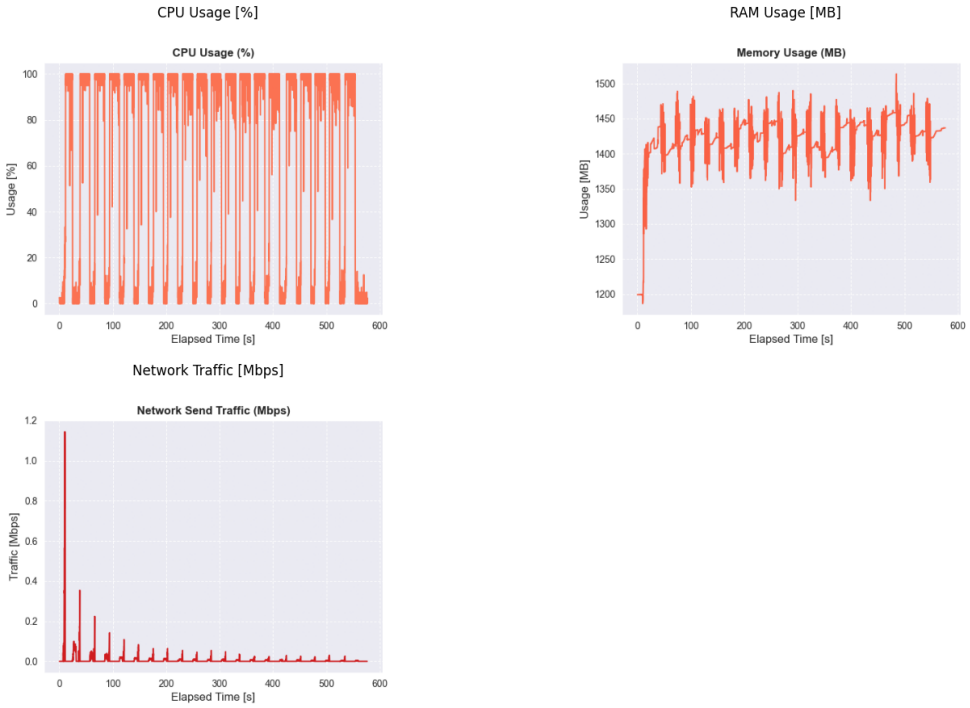


Source: Author (2024)

APPENDIX B - HARDWARE TESTS RESULTS FOR FEDSDG

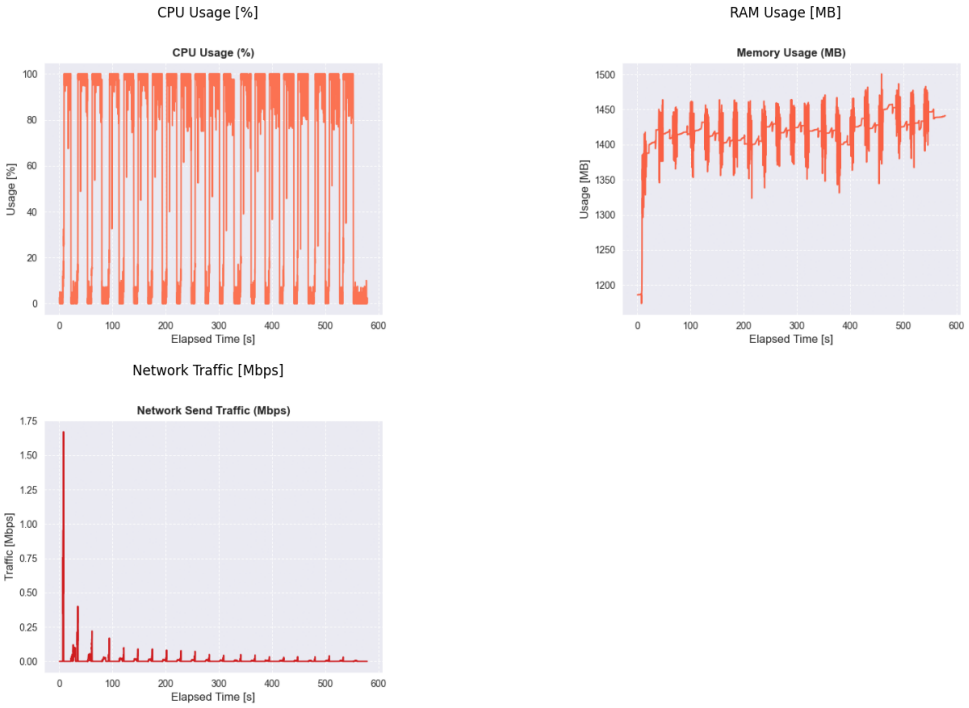
This next set of results aims to present the metrics achieved while adopting the FedSDG strategy. Figures 40, 41, 42 and 43 depict the metrics obtained in device IPT-N-0007.

Figure 40 – Hardware metrics for party IPT-N-0007 in experiment 2



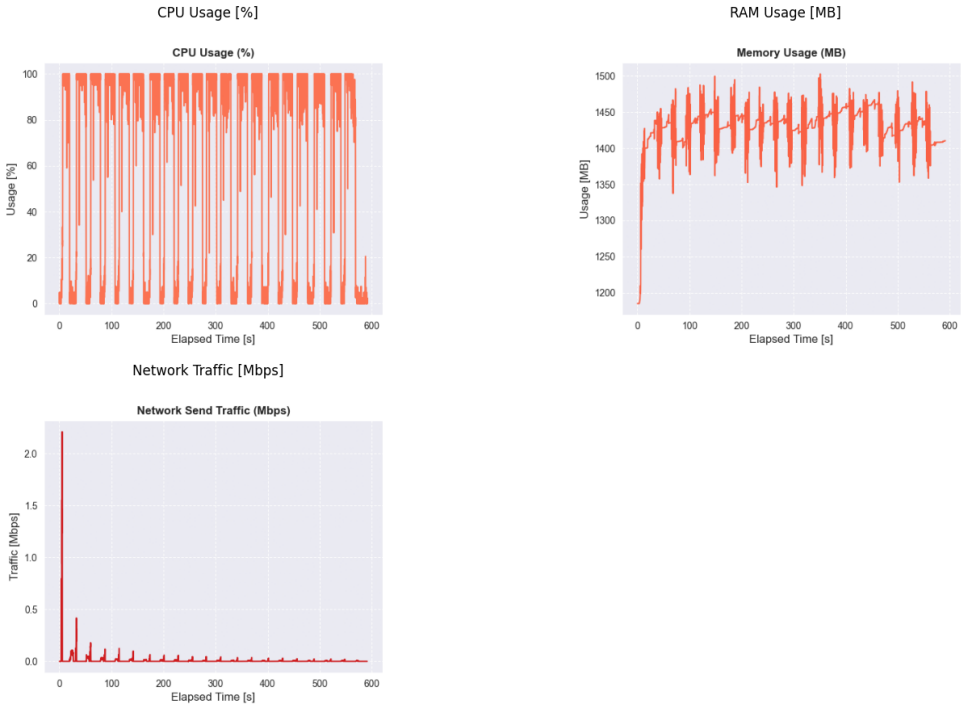
Source: Author (2024)

Figure 41 – Hardware metrics for party IPT-N-0007 in experiment 3



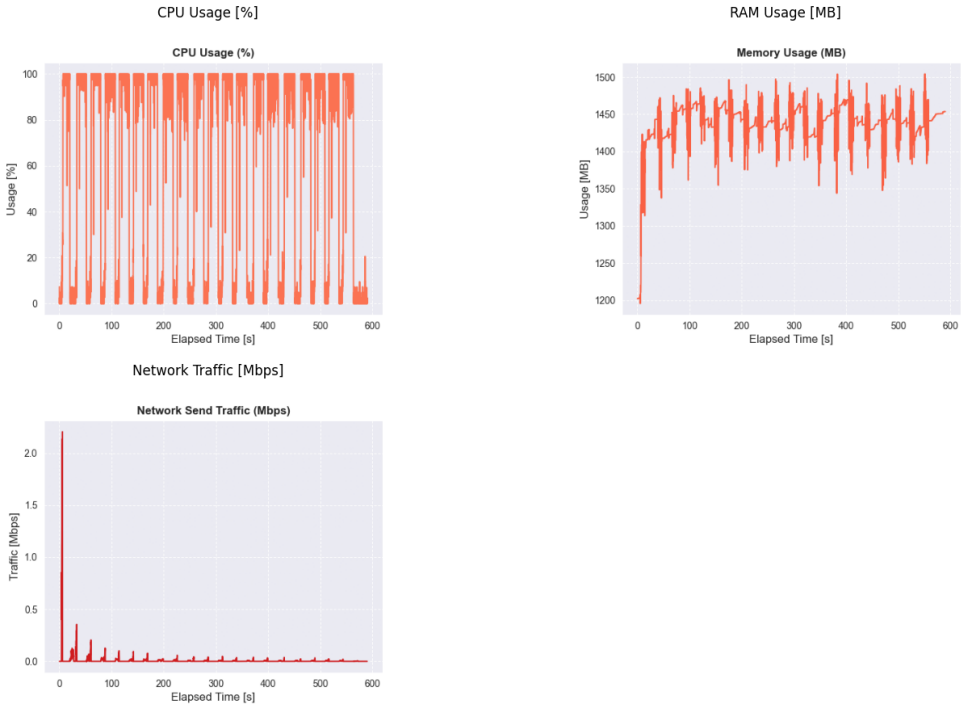
Source: Author (2024)

Figure 42 – Hardware metrics for party IPT-N-0007 in experiment 4



Source: Author (2024)

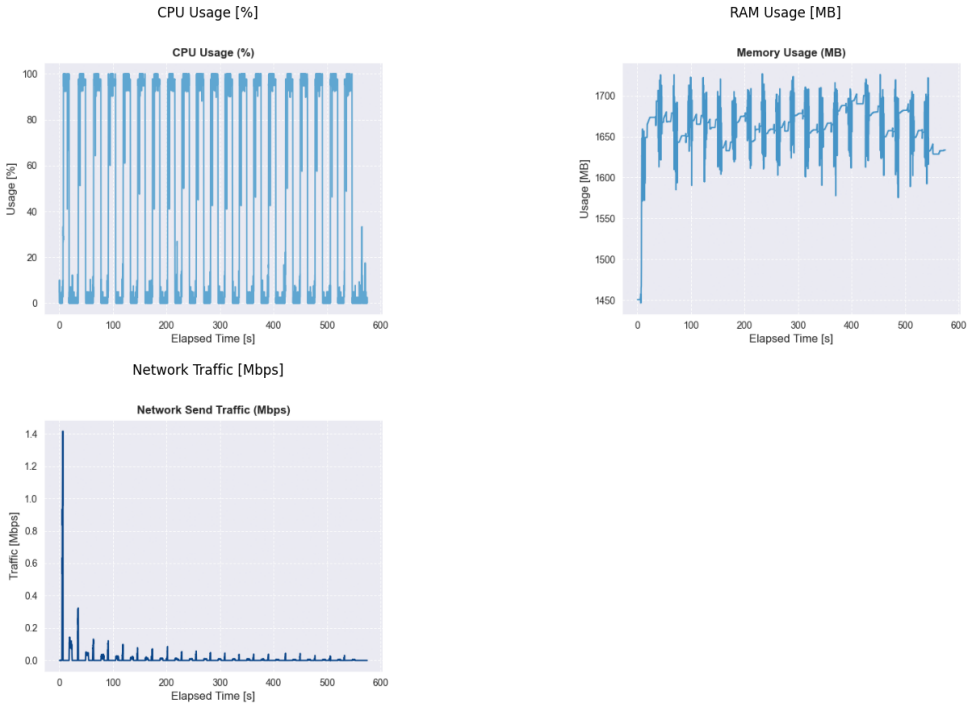
Figure 43 – Hardware metrics for party IPT-N-0007 in experiment 5



Source: Author (2024)

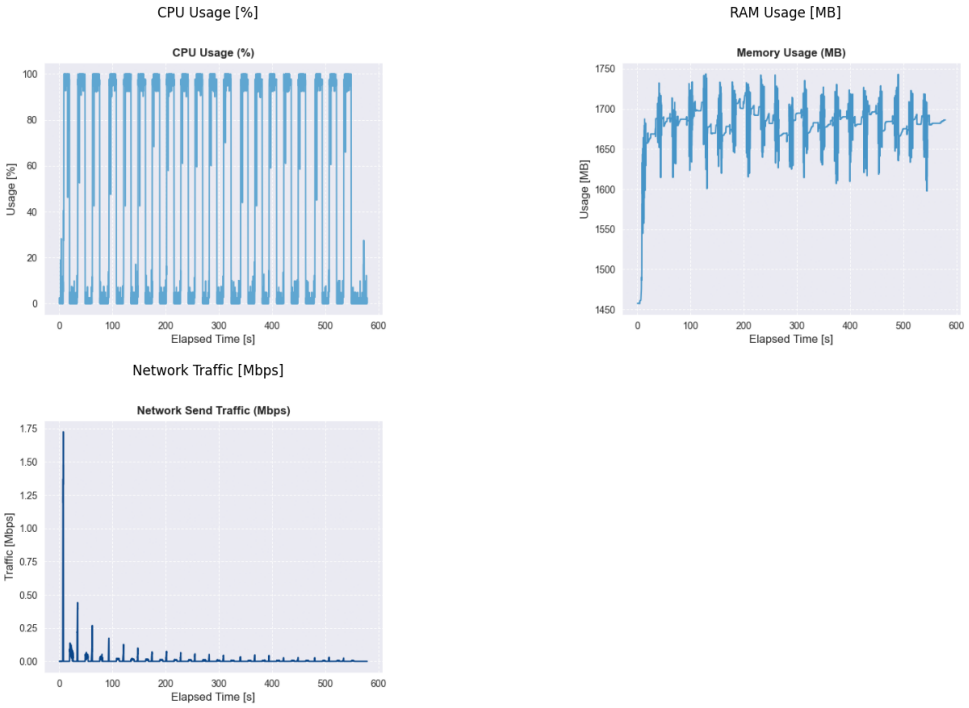
Analogously, the set containing Figures 44, 45, 46 and 47, depicts the hardware metrics for device IPT-N-0311 when adopting the FedSDG strategy.

Figure 44 – Hardware metrics for party IPT-N-0311 in experiment 2



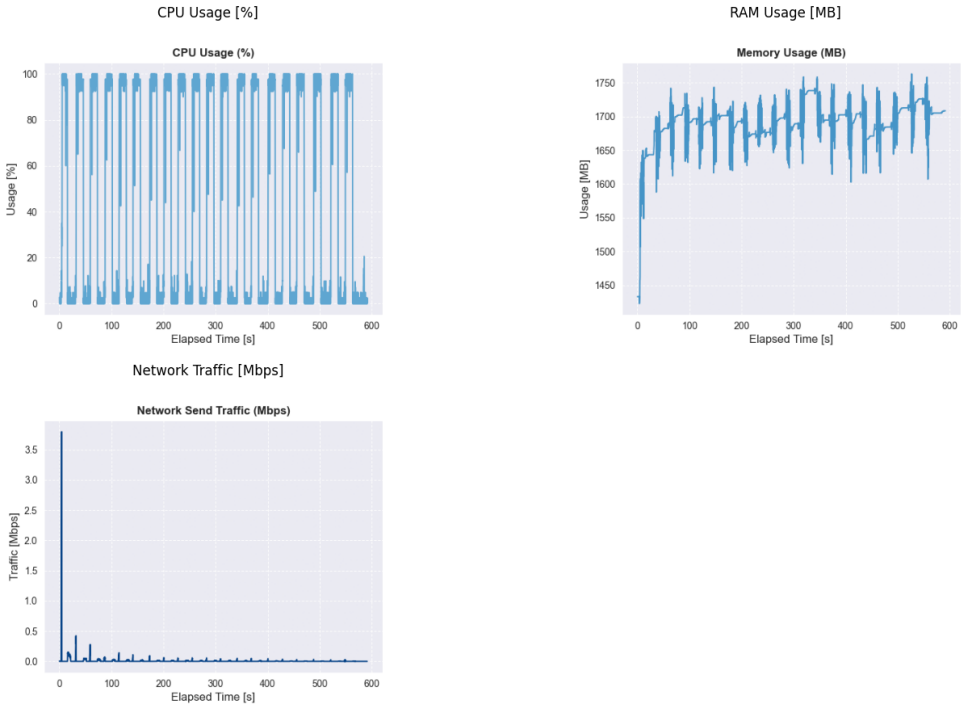
Source: Author (2024)

Figure 45 – Hardware metrics for party IPT-N-0311 in experiment 3



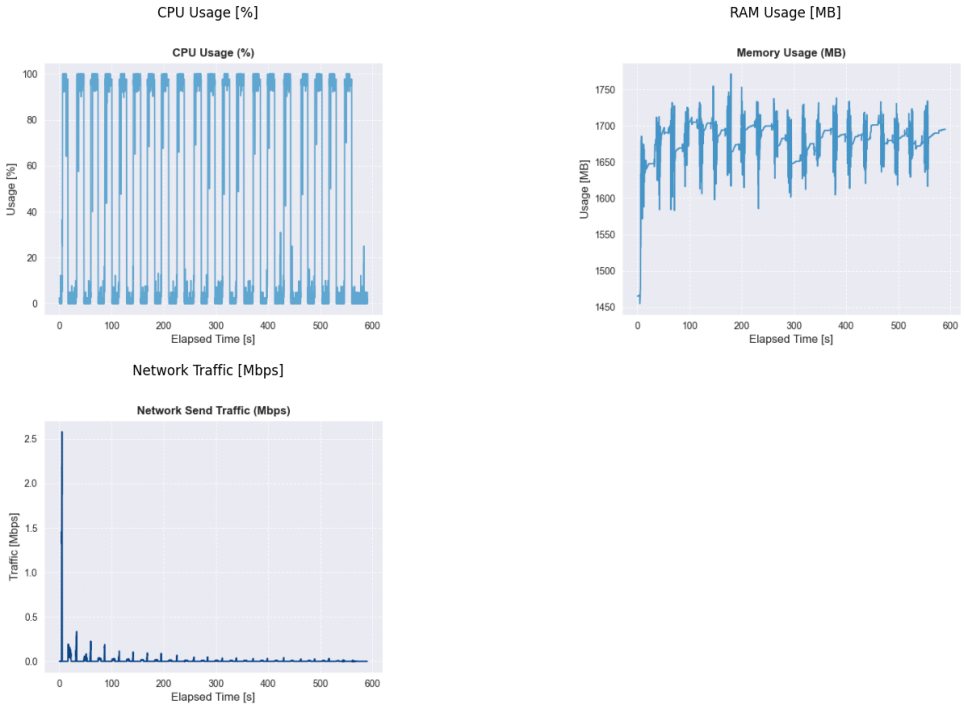
Source: Author (2024)

Figure 46 – Hardware metrics for party IPT-N-0311 in experiment 4



Source: Author (2024)

Figure 47 – Hardware metrics for party IPT-N-0311 in experiment 5



Source: Author (2024)

APPENDIX C - TRAINING METRICS FOR FEDAVG

This appendix displays the individual values for precision, recall and F Score for each classification class during the experiments using both strategies. Initially, Table 9 depicts the metrics for experiment 1 while using the FedAvg strategy.

Table 9 – Experiment 1 model metrics with FedAvg strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8931	0.9525	0.9560	0.8826	0.8792	0.9212	0.9103	0.8707	0.7556	0.7916
Recall	0.9597	0.9915	0.8555	0.8627	0.8577	0.7633	0.8980	0.8911	0.8463	0.8554
F1 Score	0.9252	0.9716	0.9030	0.8725	0.8683	0.8348	0.9041	0.8808	0.7984	0.8222

Source: Author (2024)

Next, Tables 10, 11, 12 and 13, depict the metrics for the remaining experiments still using the FedAvg strategy.

Table 10 – Experiment 2 model metrics with FedAvg strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8929	0.9372	0.9563	0.8396	0.8640	0.9284	0.9120	0.8403	0.8097	0.8341
Recall	0.9577	0.9915	0.8630	0.9119	0.8780	0.7673	0.9161	0.9008	0.8063	0.7996
F1 Score	0.9241	0.9636	0.9073	0.8743	0.8710	0.8402	0.9140	0.8695	0.8080	0.8165

Source: Author (2024)

Table 11 – Experiment 3 model metrics with FedAvg strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8782	0.9357	0.9563	0.8596	0.8889	0.9593	0.8936	0.8553	0.7632	0.8032
Recall	0.9597	0.9915	0.8612	0.9037	0.8293	0.7224	0.9138	0.8969	0.8547	0.8182
F1 Score	0.9171	0.9628	0.9062	0.8811	0.8580	0.8242	0.9036	0.8756	0.8064	0.8106

Source: Author (2024)

Table 12 – Experiment 4 model metrics with FedAvg strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.9013	0.9433	0.9612	0.8410	0.8673	0.9619	0.9018	0.8506	0.7468	0.8207
Recall	0.9577	0.9915	0.8368	0.8996	0.8638	0.7204	0.9161	0.8969	0.8632	0.8037
F1 Score	0.9286	0.9668	0.8947	0.8693	0.8656	0.8238	0.9089	0.8731	0.8008	0.8121

Source: Author (2024)

Table 13 – Experiment 5 model metrics with FedAvg strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8968	0.9479	0.9563	0.8685	0.8770	0.9296	0.8923	0.8642	0.7834	0.8216
Recall	0.9637	0.9915	0.8630	0.8934	0.8699	0.7551	0.9206	0.8911	0.8526	0.8182
F1 Score	0.9291	0.9692	0.9073	0.8808	0.8735	0.8333	0.9062	0.8774	0.8165	0.8199

Source: Author (2024)

APPENDIX D - TRAINING METRICS FOR FEDSDG

Analogously, the metrics for the individual classes were also recorded for the model using the FedSDG strategy. Table 14 depicts the metrics achieved in experiment 1.

Table 14 – Experiment 1 model metrics with FedSDG strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8966	0.9463	0.9660	0.8580	0.8991	0.9332	0.9067	0.8474	0.7857	0.8101
Recall	0.9617	0.9898	0.8518	0.8914	0.8516	0.7694	0.9252	0.8969	0.8568	0.8285
F1 Score	0.9280	0.9675	0.9053	0.8744	0.8747	0.8434	0.9158	0.8715	0.8197	0.8192

Source: Author (2024)

Following the same pattern, Tables 15, 16, 17 and 18, refers to the individual metrics for the other four experiments using the FedSDG strategy.

Table 15 – Experiment 2 model metrics with FedSDG strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8810	0.9447	0.9574	0.8628	0.8803	0.9267	0.8672	0.8363	0.8193	0.8038
Recall	0.9698	0.9898	0.8443	0.8893	0.8374	0.7735	0.9184	0.9047	0.8400	0.7955
F1 Score	0.9232	0.9667	0.8973	0.8759	0.8583	0.8432	0.8921	0.8692	0.8295	0.7996

Source: Author (2024)

Table 16 – Experiment 3 model metrics with FedAvg strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8960	0.9607	0.9761	0.9021	0.8435	0.9256	0.8811	0.8290	0.8123	0.8259
Recall	0.9576	0.9825	0.8450	0.8838	0.8453	0.7636	0.8940	0.9158	0.8687	0.8422
F1 Score	0.9258	0.9715	0.9059	0.8928	0.8444	0.8369	0.8875	0.8702	0.8395	0.8340

Source: Author (2024)

Table 17 – Experiment 4 model metrics with FedSDG strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8723	0.9607	0.9555	0.9040	0.8518	0.9135	0.8637	0.8131	0.8614	0.8211
Recall	0.9657	0.9825	0.8430	0.8856	0.8388	0.7682	0.8960	0.9199	0.8630	0.8146
F1 Score	0.9166	0.9715	0.8957	0.8947	0.8452	0.8346	0.8796	0.8632	0.8622	0.8178

Source: Author (2024)

Table 18 – Experiment 5 model metrics with FedSDG strategy

Metrics	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Precision	0.8929	0.9620	0.9459	0.8611	0.8745	0.9105	0.8671	0.8368	0.8222	0.8013
Recall	0.9574	0.9848	0.8184	0.8560	0.8366	0.7672	0.9113	0.9129	0.8723	0.8333
F1 Score	0.9240	0.9733	0.8775	0.8586	0.8551	0.8327	0.8886	0.8732	0.8465	0.8170

Source: Author (2024)