

الجمهورية الجزائرية الديمقراطية الشعبية

République algérienne démocratique et populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة عين تموشنت بلعاج بوشايب

Université Ain Temouchent Belhadj Bouchaïb

Faculté des Sciences et de Technologie

Département des Mathématiques et Informatique



End of Cycle Project

To obtain the Master's degree in: Computer Science

Field: Mathematics and Computer Science

Sector: Computer Science

Specialty: Networks and data engineering

Theme

**Federated Learning in Distributed Networks: Leveraging Collaborative
Machine Learning without Compromising Data Privacy**

Presented by :

MLLE. SMAHI FAIROUZ

MLLE.SAYAH ASMAA

Defended on: 24/06/2024

Before the jury composed of

Supervisor Dr Benaribi Fethi

Chairman Dr Berrakem .F

Examiner Dr Bouchakour Errahmani .H

Academic year: 2023/2024

Remerciement

We thank God for giving us the courage and passion to complete this work. We wish to express our sincere gratitude to our supervisors, Mr. Benaribi Imad. Our thanks also go to the jury members for the honor they have given us by agreeing to evaluate this work and participate in the defense. We are grateful to all the teachers at Belhadj Bouchaib University for their contribution to our education and their expertise in furthering our studies. We thank our parents, who have supported us greatly throughout our lives and will continue to help us in all our future projects. We also thank our brothers and sisters who have encouraged and helped us, whether directly or indirectly, in our lives. Finally, we thank everyone who has helped us complete this work, whether directly or indirectly.

Contents

General Introduction.....	1
1 Background.....	3
1.1 Introduction.....	3
1.2 Machine Learning.....	3
1.3 Deep Learning.....	4
1.3.1 Single-layer neural network	4
1.3.2 Multi Layer neural network	5
1.3.3 Activation functions	6
1.3.4 Deep Learning Approaches	8
1.3.5 Table of approach	9
1.3.6 Loss Functions	9
1.3.7 Deep Learning Optimizers.....	10
1.3.8 Deep Learning Types	12
1.4 Transfer Learning	17
1.5 Federated Learning.....	18
1.5.1 Federated Learning Approaches	18
1.5.2 The Life cycle of a Model in Federated Learning.....	21
1.5.3 Federated Optimization Algorithms.....	21
1.5.4 Federated Learning Challenges.....	23
1.6 ICPS.....	24
1.7 IoT	24
1.7.1 IoT Architecture.....	25
1.8 Cloud.....	26
1.8.1 Characteristics.....	27
1.8.2 Service Models	27
1.8.3 Cloud Solutions.....	28
1.9 Fog.....	29
1.9.1 Characteristics.....	29
1.9.2 Fog Node Attributes	30
1.9.3 Fog Node Architectural Service and Deployment Models	31
1.9.4 Architecture.....	31
1.10 Edge.....	33
1.10.1 Architecture.....	33
1.10.2 Features	34
1.11 Conclusion	35

2 State of the art	37
Contents	
<hr/>	
2.1 Introduction.....	37
2.2 Cloud-based FL.....	37
2.3 Fog-based FL.....	39
2.4 Edge-based FL.....	40
2.5 Discussion and Comparison	42
2.6 Conclusion	45
3 FedGA-ICPS Framework.....	46
3.1 Design Architecture	46
3.1.1 IoT Layer	47
3.1.2 Edge Layer.....	48
3.1.3 Fog Layer.....	48
3.1.4 Cloud Layer	48
3.2 FedGA-ICPS Framework.....	49
3.2.1 Learning.....	53
3.2.2 Election	54
3.2.3 Federation	56
3.2.4 Broadcasting.....	57
3.3 Conclusion	57
4 FedGA-ICPS Implementation and Experiments.....	59
4.1.1 Dataset Fashion MNIST	65
4.1.2 Dataset EMNIST	65
4.1.3 Dataset MNIST	66
4.1.4 Dataset Cifar-10	66
4.1.5 Neural Network Models.....	66
4.1.6 Transfer Learning.....	67
4.1.7 Election.....	68
4.1.8 Federated With Genetic Algorithm (FedGA).....	69
4.1.9 Optimization Algorithms	70
4.2.9.1 Particle Swarm Optimization.....	70
4.2.9.2 Ant Colony Optimization	72
4.2 Experiments and Results	73
4.2.1 FL Evaluation.....	73
4.3 Discussions and comparison	77
4.4 Conclusion	78
General Conclusion	79
Bibliography	80

List of Figures

1.1	Deep Learning neural network [60].	4
1.2	Single Layer Perceptron representation	5
1.3	Overview of a Multi-Layer Perceptron	6
1.4	Sigmoid Activation Function	7
1.5	Hyperbolic Tangent Activation Function	7
1.6	ReLU Activation Function	8
1.7	ANN Architecture	13
1.8	CNNs Architecture	14
1.9	RNN Architecture	14
1.10	LSTM Architecture	15
1.11	GRU Architecture	16
1.12	DenseNet Architecture	16
1.13	Difference between Traditional ML and TL	17
1.14	Structure of federated learning	18
1.15	Classification of federated learning	19
1.16	Centralized Federated Learning Architecture	20
1.17	Peer to peer decentralized Federated Learning Architecture	20
1.18	FedPer illustration	23
1.19	Example of a proposed IoT-Edge-Fog-Cloud architecture [14].	25
1.20	IoT SOA Architecture [6].	25
1.21	Cloud Computing Models [43].	28
1.22	Fog Computing Architecture [62].	32
1.23	Edge Computing Architecture.	34
3.1	The Proposed Architecture Design for FedGA-ICPS framework	47
3.2	The Interoperability and Integrity Validation and Evaluation [33]	49
3.3	FedGA-ICPS Class Diagram.	52
3.4	TL Registry Structure	53
3.5	Learning Sequence Diagram	54
3.6	A Configuration of the Election Process at a slot of time	55
3.7	Election Sequence Diagram	55
3.8	FedGA Schema	56
3.9	FL Sequence Diagram	58
4.1	Taken from the Fashion MNIST dataset	65
4.2	Taken from EMNIST dataset	66
4.3	Taken from MNIST dataset	66

List of Figures

4.4	Taken from Cifar-10 dataset.....	66
4.5	CNN model.....	67
4.6	ANN model.....	67
4.7	RNN model.....	68
4.8	GRU model.....	68
4.9	densenet model.....	69
4.10	Election attributes	69
4.11	FedGA Configuration	70
4.12	FedGA Fitness Function	70
4.14	FedAVG(1)	71
4.15	FedPer(1)	72
4.16	FedGA(1).....	72
4.17	FedAVG(2).....	73
4.18	FedGA(2)	73
4.19	FedPer(2).....	74
4.20	FedAVG(3).....	74
4.21	FedGA(3)	75
4.22	FedPer(3)	75
4.23	FedAVG(4).....	76
4.24	FedGA(4).....	76
4.25	FedPer(4)	77

List of Tables

1.1: Deep Learning Approaches.....	11
2.1 Comparison of the State of The Art	44
4.1 Application Domains and Tasks	68
4.2: Comparing graph representations between the Standard Genetic Algorithm and Particle Swarm Optimization.....	71
4.3: Comparing graph representations between the Standard Genetic Algorithm and Ant Colony Optimization.....	72

List of Algorithms

1	Federated Averaging (FedAVG) algorithm [58]	22
2	Federated Genetic Algorithm (FedGA) [32]	57

List of abbreviations and acronyms

AI	<i>Artificial intelligence</i>
ML	<i>Machine Learning</i>
FL	<i>Federated Learning</i>
TL	<i>Transfer Learning</i>
SLP	<i>Single Layer Perceptron</i>
MLP	<i>Multi Layers Perceptron</i>
ReLU	<i>Rectified Linear Unit</i>
RNN	<i>Recurrent Neural Network</i>
CNN	<i>Convolutional Neural Network</i>
FFNN	<i>Feed Forward Neural Network</i>
LSTM	<i>Long Short Terme Memory</i>
BPTT	<i>Backpropagation Through Time</i>
CPS	<i>Cyber physical System</i>
ICPS	<i>Industrial Cyber physical System</i>
IoT	<i>Internet Of Things</i>
TCP	<i>Transmission Control Protocol</i>
UDP	<i>User Datagram Protocol</i>
WAN	<i>Wide Area Network</i>
LAN	<i>Local Area Network</i>
LPWAN	<i>Low-power WAN</i>
RFID	<i>Radio Frequency Identification</i>
NFC	<i>Near Field Communication</i>

GSM	<i>Global System for Mobile Communications</i>
GPS	<i>Global Positioning System</i>
LTE	<i>Long Term Evolution</i>
QoS	<i>Quality of Service</i>
HTTP	<i>Hypertext Transfer Protocol</i>
MQTT	<i>MQ Telemetry Transport</i>
DDS	<i>Data Distribution Service</i>
AMQP	<i>Advanced Message Queuing Protocol</i>
XMPP	<i>Extensible Messaging and Presence Protocol</i>
JMS	<i>Java Messaging Services</i>
REST	<i>Representational state transfer</i>
CoAP	<i>Constrained Application Protocol</i>
OPC UA	<i>Open Platform Communications United Architecture</i>
M2M	<i>Machine To Machine</i>

General introduction

The Internet of Things (IoT) has developed rapidly in recent years, offering ubiquitous sensing and computing capabilities that enable a wide range of objects to be connected to the Internet [54]. There are multiple domains such as the Internet of Industrial Things (IoIT), referring the use of connected devices, sensors, intelligent systems, fast communication protocols and effective cybersecurity mechanisms to enhance and optimize various industrial processes and applications [11]. Managing the profusion of data generated by intelligent devices within cyber-physical systems necessitates sophisticated and secure big data analysis approaches. The intricacy of decision-making continually intensifies, given the ever-growing volume of data. In this context, the effectiveness of artificial intelligence (AI), machine learning (ML) and deep learning (DL) techniques in particular has been demonstrated, due to their sophisticated learning and processing capabilities, particularly within network-based systems [29].

The conventional approach to implementing AI in cyber-physical systems (CPS) was centralized, with a central server driving the model using data from all connected end devices. However, transmitting such a large volume of data from these end devices to the cloud leads to bandwidth congestion, delays in data processing and a potential risk of privacy breach [44].

We conducted research to understand the deployment of FL across Cloud, Fog, and Edge computing environments, as well as its integration with other methodologies like deep learning and transfer learning. Furthermore, we evaluated the innovative solution, FedGA-ICPS, developed by Mademoiselle Guendouzi Badra Souhila. Following Guendouzi's work, we conducted tests on various datasets to assess its performance in heterogeneous environments. We conducted extensive tests on multiple benchmarks and topologies, including datasets such as Fashion MNIST, EMNIST, MNIST, and CIFAR-10, as well as models like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs), Artificial Neural Networks (ANNs), and DenseNet.

Additionally, we tested two optimization algorithms, Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), with the EMNIST and MNIST datasets. These tests were conducted to evaluate the effectiveness of these optimization techniques in improving model performance. The results underscore the versatility and efficacy of PSO and ACO in various scenarios, demonstrating their potential to enhance the accuracy and

efficiency of our models within the FedGA-ICPS framework.

In this context, our project report is structured into four chapters: In Chapter 1, we discuss the generalities of Machine Learning (ML) and Integrated Cyber -Physical Systems (ICPS). In Chapter 2, we evaluate existing architectures that utilize Federated Learning (FL), Deep Learning (DL), and Transfer Learning (TL) methods to facilitate decision-making in the cloud, fog, and edge layers, while also identifying their flaws. In Chapter 3, we detail the architecture of our FedGA-ICPS framework. Following that, in Chapter 4, we test it with a chosen use case. Finally, we conclude our work with a general conclusion.

Chapter 1

Background

1.1 Introduction

In today's world, data is predominantly stored in digital format, thanks to advancements in computer programs capable of handling large volumes of information. Data isn't just about numbers; it's more than just digits. It has evolved beyond being a tool solely for analyzing past events; it now serves as a crucial resource for informing decisions and predicting future outcomes.

Quality data forms the cornerstone of any efficient operation. Without it, businesses cannot effectively forecast, plan, or monitor long-term performance, which directly impacts their success. In recent years, the volume of data has surged, leading to increasingly complex decision-making processes. This complexity has prompted the adoption of machine learning (ML) techniques, which streamline decision-making and provide valuable insights into customer behavior and business operations. Furthermore, there exists a symbiotic relationship between AI techniques and ICPS (Integrated Cyber -Physical Systems). This chapter offers an overview of machine learning (ML), describing its methodologies including Deep Learning (DL), Transfer Learning (TL), and Federated Learning (FL), as well as an introduction to ICPS layers: IoT, Cloud, Fog, and Edge.

1.2 Machine Learning

Machine learning is an automated process that occurs through the analysis of typically large datasets. Past procedures, known as "symbolic artificial intelligence," relied on algorithms consisting of a logical sequence of instructions to encode a given output (often referred to as the target) for all possible inputs. In contrast, modern machine learning systems "learn" directly from data, estimating mathematical functions that expose representations of some input, or learn to connect one or more inputs to one or more outputs to make predictions on new data [78].

According to Boris Katz, a principal research scientist and head of the InfoLab Group at CSAIL, the objective of AI is to develop computer models capable of demonstrating "intelligent behaviors" related to humans. This necessitates creating machines proficient

in tasks such as visual scene recognition, comprehension of natural language text, and execution of actions in the physical realm.

Machine learning serves as a methodology within AI, initially defined by AI pioneer Arthur Samuel in the 1950s as "the field of study that empowers computers to learn autonomously without explicit programming."

Algorithms of Machine Learning (ML) are employed across various domains, encompassing medicine, email filtering, speech recognition, and computer vision. This technology proves indispensable in scenarios where conventional algorithms struggle or fail to accomplish the designated tasks [39].

1.3 Deep Learning

Deep learning, a subset of machine learning, utilizes algorithms to identify patterns within complex datasets and forecast outcomes. Unlike traditional machine learning methods, which rely on labeled datasets, deep learning networks can be trained using unsupervised learning, reducing the need for labeled data and human input. The term "deep learning" originates from the incorporation of numerous hidden layers within its models. While a basic neural network consists of input, output, and hidden layers, deep neural networks feature multiple hidden layers for more intricate processing as it is shown in Figure 1.1.

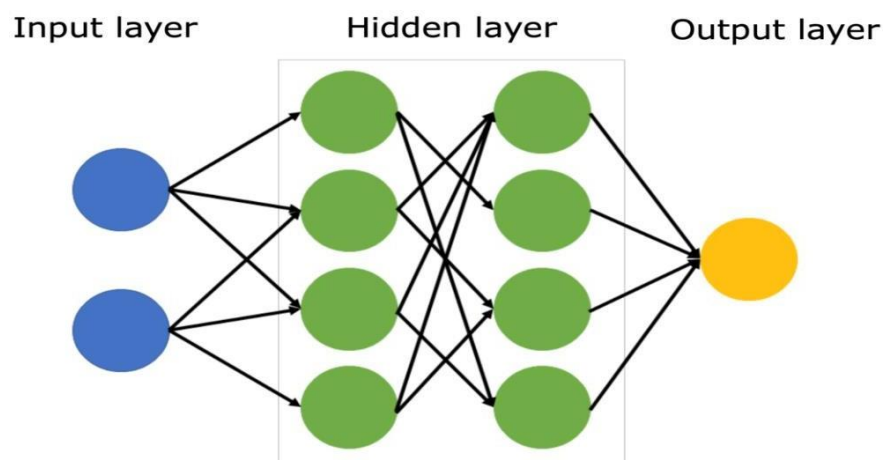


Figure 1.1: Deep Learning neural network [60].

These additional layers enhance the accuracy of predictions made by deep learning systems. However, achieving this level of accuracy requires millions of data points and hundreds of hours of training, surpassing the requirements of simpler neural networks.

1.3.1 Single-layer neural network

A single-layer neural network, often referred to as a "perceptron," is the simplest form of an artificial neural network. It comprises just one layer of artificial neurons or perceptrons, known as the "output layer," which produces the final output of the network. It uses

Chapter 1. Background

high-dimensional vector $X = x_1, x_2, \dots, x_n$ as the feature input and performs binary classification on it by multiplying it with the weight vector $W = w_1, w_2, \dots, w_n^T$ and adding a bias b :

$$Y = W^T \times X + b \quad (1.1)$$

The final output is obtained by applying an activation function, given by:

$$\text{Output}(Y) = \text{Sign}(Y) = \begin{cases} 1 & \text{if } Y > 0 \\ -1 & \text{otherwise} \end{cases} \quad (1.2)$$

A representation of the single-layer neural network is shown in Figure 1.2.

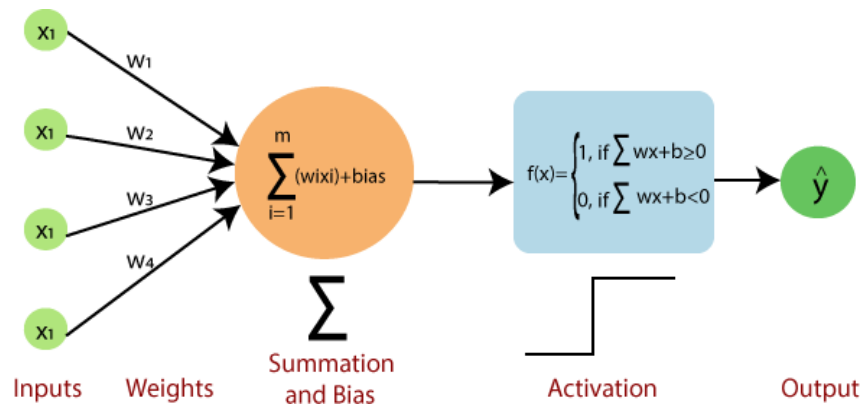


Figure 1.2: Single Layer Perceptron representation.

single-layer neural networks are rarely used for real-world applications because of their limitations. Multi-layer neural networks, such as feedforward neural networks (FFNNs), are more capable of handling complex tasks and are commonly used in tasks like image recognition, natural language processing, and many other machine learning applications.

1.3.2 Multi Layer neural network

An MLP, or multilayer perceptron, is a modern type of feedforward artificial neural network, consisting of three layer types illustrated in Figure 1.3: the input layer, output layer, and hidden layer(s) [5]. The input layer receives the input data to be processed, while the output layer is responsible for tasks such as prediction and classification. The true computational center of the MLP lies within its hidden layers, which can vary in number and are situated between the input and output layers.

In an MLP, data progresses from the input to the output layer in a forward direction, similar to a feedforward network. The training of MLP neurons is accomplished through the backpropagation learning algorithm. This approach enables MLPs to tackle problems

that lack linear separability and are engineered to approximate any continuous function. Common applications of MLP include pattern categorization, recognition, prediction, and approximation.

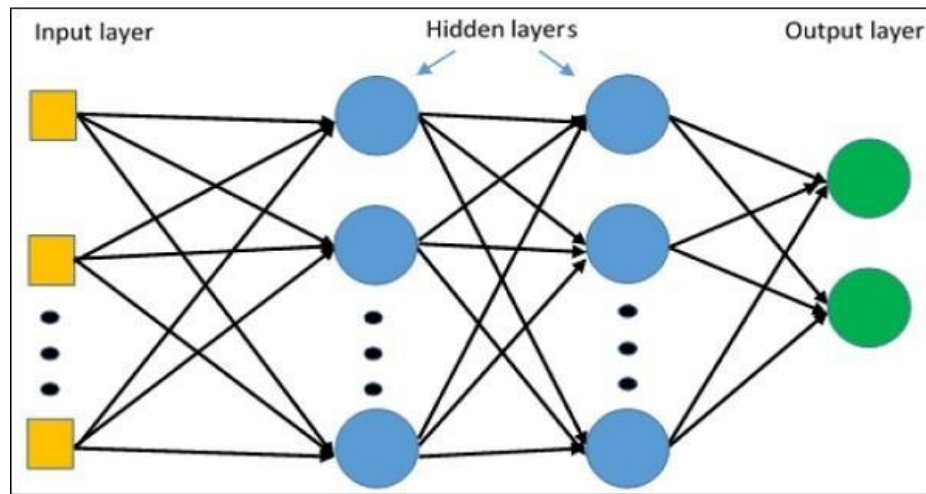


Figure 1.3: Overview of a Multi-Layer Perceptron.

1.3.3 Activation functions

In a neural network, each layer consists of nodes, and every node possesses a weight that influences the input processing from one layer to the next. Without an activation function, the output signal of a neural network remains a simple linear function, related to a straightforward one-degree polynomial. To effectively interpret complex, high-dimensional, and nonlinear datasets, especially when employing models with multiple hidden layers like deep learning (DL). These functions enable the network to capture complex patterns and relationships within the data, facilitating more robust and subtle analysis [79].

Sigmoid Activation Function.

The sigmoid activation function, also known as the logistic function shown in Figure 1.4, is typically applied in the output layer of binary classification tasks, where the outcome is either 0 or 1. This function constrains its output between 0 and 1. Consequently, predictions are straightforward: if the sigmoid function's value exceeds 0.5, the result is predicted as 1; otherwise, it is predicted as 0. The formula for the sigmoid function is as follows:

$$\text{Sig}(x) = \frac{e^x}{1 + e^x} \quad (1.3)$$

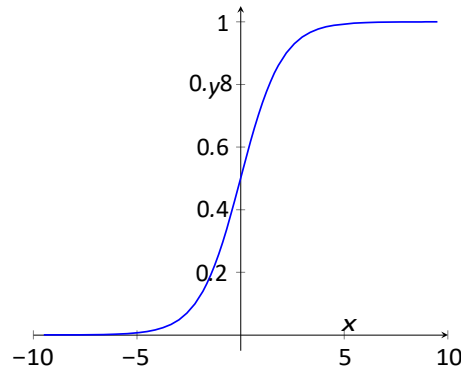


Figure 1.4: Sigmoid Activation Function.

Hyperbolic Tangent Activation Function TanH

The activation that works almost always better than sigmoid function is Tanh function also known as Tangent Hyperbolic function that is shown in Figure 1.5. Usually used in hidden layers of a neural network as its values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier. The formula is as follows:

$$\text{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (1.4)$$

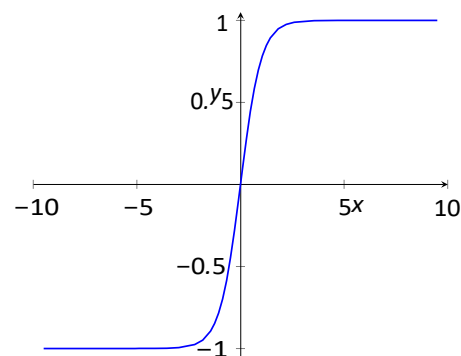


Figure 1.5: Hyperbolic Tangent Activation Function.

ReLU Activation Function

ReLU is non-linear activation function that is shown in Figure 1.6. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural network. ReLU is less computationally expensive than tanh and sigmoid [79] because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation. It can be defined mathematically as:

$$f(x) = \max(x, 0) \quad (1.5)$$

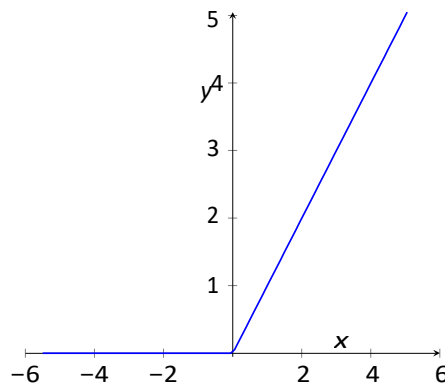


Figure 1.6: ReLU Activation Function.

Softmax Activation Function

The Softmax activation function is ideally used in the output layer of the classifier where are actually trying to attain the probabilities to define the class of each input. It can be expressed as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{i=1}^x e^i} \quad (1.6)$$

Chapter 1. Background

The number of neurons in the output layer is the same as the number of classes in the target. It is equal to K in the previous formula.

1.3.4 Deep Learning Approaches

1.3.5 Table of approach

Approach	Description	Example	Reference
Deep Supervised Learning	The intelligent agent makes a supposition that $y_i = f(x_i)$ given the input x_i , and then calculates a loss value $\rho(\hat{y}_i, y_i)$. Subsequently, the agent iteratively updates the network parameters to improve the approximation of the desired outputs.	Image classification	[77]
Deep Unsupervised Learning	Aims to create models that can extract meaningful representations from vast amounts of unlabeled sensory input. This approach draws inspiration from the visual cortex, which demonstrates the ability to derive insights from minimal labeled data.	Clustering, dimensionality reduction	[15]

Chapter 1. Background

Deep Reinforcement Learning	In reinforcement learning approaches, an artificial intelligence agent engages with an actual or simulated environment. This interaction prompts feedback between the learning system and the interaction session, aiding the machine in enhancing its performance on the task it is learning.	Game learning, robot control	[15]
------------------------------------	--	------------------------------	------

Table 1.1: Deep Learning Approaches

1.1.1 Loss Functions

To move from the Input layer to the Output layer in a neural network for prediction, we use activation functions in a process called **Forward Propagation**. Conversely, **Backward Propagation** is employed to modify the weights of the neural network, moving from the output layer to the input layer, aiming to minimize the loss function, which represents the algorithm's error. Different loss functions, such as mean squared error, cross-entropy loss, and hinge loss, exist for various tasks and models, including regression and classification. The selection of the appropriate loss function depends on the task at hand and the nature of the model being used. Especially in this section, we will talk about the most important loss functions used in neural networks.

Mean Absolute Error (MAE)

The mean absolute error (MAE) loss function is a function used for regression tasks. It computes the average of the absolute variances between predicted and true outputs. MAE loss is determined by averaging the absolute differences between the predicted and true values across all samples in the dataset. The MAE loss can be represented by the following equation:

$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (1.7)$$

Chapter 1. Background

Mean Squared Error (MSE)

The Mean Squared Error (MSE) loss function is defined as the average of the squared differences between the actual and predicted values. It stands as the most frequently employed regression loss function [76]. It calculates the average of squared differences between actual and predicted values. For a data point y_i with predicted value \hat{y}_i and a total of n data points in the dataset, the mean squared error can be represented by the following equation:

$$\text{MSE} = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad (1.8)$$

Cross Entropy Loss

The Cross-Entropy loss, also known as log loss, evaluate the output of a classification model, represented as a probability value ranging between 0 and 1. This loss increases as the predicted probability deviates from the actual label, It is calculated by the following formula :

$$J = \frac{1}{n} \sum_{i=1}^n y_i \log(\hat{y}_i) \quad (1.9)$$

Where n is the number of examples, y_i is the actual vector label, and \hat{y}_i is the classifier's output for the i -th example in the dataset.

1.1.2 Deep Learning Optimizers

Deep learning deals with complex tasks such as speech recognition and text classification. It consists of components like activation functions, layers, and loss functions, aiming to generalize data and make predictions on unseen data. Optimizers are algorithms used to adjust model parameters during training to minimize a loss function, thus improving accuracy and speeding up the training process, crucial for enhancing the performance of deep learning models.

Batch Gradient Descent (BGD)

Batch gradient descent calculates the error for each example within the training dataset, but updates the model only after evaluating all training examples. This cycle is termed a training epoch. The gradient of a function, synonymous with its slope, determines the rate of learning. A steeper gradient accelerates learning, while a zero gradient halts learning. Mathematically, the gradient represents partial derivatives with respect to inputs [27]. This process can be written as:

$$W^{(t)} = W^{(t-1)} - \alpha \frac{\partial J(W^{(t-1)}, T)}{\partial W} \quad (1.10)$$

where $t \geq 1$ denotes the updating iteration. $W^{(t)}$ denotes the updated model variable vector in iteration t , $\frac{\partial J}{\partial W}$ denotes the derivative of the loss function $J(W; T)$ with respect to the variable W based on the complete training set T and the variable vector value $W^{(t-1)}$ achieved in iteration $t-1$. α denotes the learning rate in the gradient descent algorithm, usually a hyperparameter with a small value. Fine-tuning the parameter is necessary for fast convergence in practical applications of the batch gradient descent algorithm [94].

Stochastic Gradient Descent (SGD)

Stochastic gradient descent calculates the gradient of the loss function using the entire training set, making it inefficient for large datasets. Stochastic gradient descent (SGD) addresses this inefficiency by updating model parameters using the gradient computed for individual instances. While SGD offers faster computation, it introduces fluctuations in the loss function during updates [94]. Formally, Given the training set and the initialized model variable vector $W^{(0)}$, for each instance $(x_i, y_i) \in T$, the SGD algorithm updates the model variable with the following equation iteratively:

$$W^{(t)} = W^{(t-1)} - \alpha \frac{\partial J(W^{(t-1)}, (x_i, y_i))}{\partial W} \quad (1.11)$$

Mini-batch Gradient Descent (MBGD)

To strike a balance between the Batch Gradient Descent (BGD) and Stochastic Gradient Descent (SGD) learning algorithms, Mini-Batch Gradient Descent suggests updating the model parameters using a mini-batch of training instances. Formally, let $\mathcal{B} \subset T$ denote the mini-batch of training instances sampled from the training set T . We can represent the variable updating equation of the DL model with the mini-batch as follows:

$$W^{(t)} = W^{(t-1)} - \alpha \frac{\partial J(W^{(t-1)}, \mathcal{B})}{\partial W} \quad (1.12)$$

Momentum

To mitigate the fluctuations encountered during the learning process in gradient descent algorithms, such as mini-batch gradient descent, the concept of Momentum [69] is introduced to expedite the convergence of variable updates. This technique aims to smooth out the variations in parameter updates, facilitating a faster convergence towards the optimal solution. Formally, Momentum updates the variables with the following equation:

$$W^{(t)} = W^{(t-1)} - \alpha V^{(t)} \quad (1.13)$$

where

$$V^{(t)} = \rho V^{(t-1)} + (1 - \rho) \frac{\partial J(W^{(t-1)})}{\partial W} \quad (1.14)$$

In the above equation, $V^{(t)}$ denotes the momentum term for recording historical gradients till iteration t , and $J(W^{(t-1)})$ denotes the loss function. Parameter $\rho \in [0, 1]$ denotes the weight of the momentum term.

Root Mean Squared Propagation (RMSProp)

When data passes through complex functions, such as neural networks, gradients tend to either vanish or explode. RMSprop is a stochastic mini-batch learning strategy that addresses these challenges by utilizing an adaptive learning rate, rather than treating the

learning rate as a hyperparameter. Additionally, it normalizes the gradient by using a moving average of squared gradients. RMSprop's update rule is as follows :

$$W^{(t)} = W^{(t-1)} - \frac{\alpha}{\sqrt{V^{(t)} + \epsilon}} \frac{\partial J(W^{(t-1)})}{\partial W} \quad (1.15)$$

where

$$V^{(t)} = \rho V^{(t-1)} + (1 - \rho) \frac{\partial J(W^{(t-1)})}{\partial W}^2 \quad (1.16)$$

Adaptive Moment Estimation (ADAM)

In recent years, a new learning algorithm called Adaptive Moment Estimation (Adam) has emerged[50]. Unlike traditional methods, Adam efficiently computes first-order gradients with minimal memory usage. Like RMSprop and Adadelata, Adam maintains a history of past squared first-order gradients, but it goes a step further by also retaining past first-order gradients. These stored gradients decay exponentially as the learning process progresses . Adam's update rule is given by:

$$m^{(t)} = \beta_1 m^{(t-1)} + (1 - \beta_1) \frac{\partial J(W^{(t-1)})}{\partial W} \quad (1.17)$$

$$V^{(t)} = \beta_2 V^{(t-1)} + (1 - \beta_2) \frac{\partial J(W^{(t-1)})}{\partial W}^2 \quad (1.18)$$

$$\hat{m} = \frac{m^{(t)}}{1 - \beta_1} \quad (1.19)$$

$$\hat{V} = \frac{V^{(t)}}{1 - \beta_2} \quad (1.20)$$

$$W^{(t)} = W^{(t-1)} - \frac{\alpha}{\sqrt{\hat{V} + \epsilon}} \hat{m} \quad (1.21)$$

According to Kingma and Ba, good default settings for ML problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

1.1.3 Deep Learning Types

Various types of deep learning (DL) models exist, each adapted for specific functions. This section will delve into the distinct types and explain their mechanisms.

Artificial Neural Network (ANN)

ANNs consist of artificial neurons derived from biological neurons. Each artificial neuron has an input and produces a unique output that can be sent to many other neurons [3]. The inputs can be characteristic values of an external data sample, such as an image or document, or they can be the output of other neurons. The output of the neurons The final output of the neural network completes the task, such as recognizing an object in

an image. To find the output of a neuron, we take the weighted sum of all the inputs, calculated as the weights of the connections from the input to the neuron. We add an bias term to this sum [25]. This weighted sum is sometimes called the activation. This weighted sum then goes through an activation function (usually non-linear) to produce the result. The initial input is external data, such as images and documents . The end result accomplishes the task, such as recognizing objects in images [4]. The ANN architecture is shown in Figure 1.7

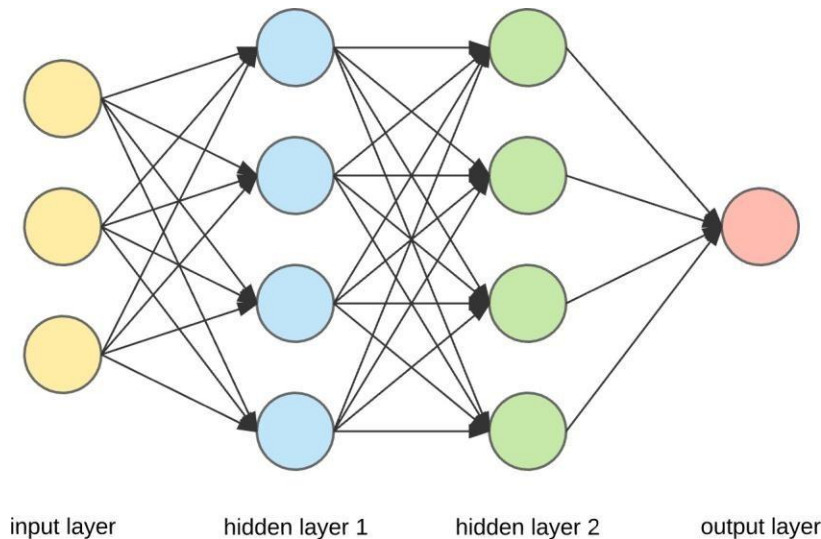


Figure 1.7: ANN Architecture.

Convolutional Neural Networks CNNs

A convolutional neural network (CNN), also known as ConvNet, belongs to the category of deep neural networks and has demonstrated success in numerous computer vision tasks, particularly in the analysis of visual images [40] . The CNN architecture, which is shown in Figure 1.8, comprises three main types of layers: (1) convolution, (2) pooling, and (3) fully connected. Convolutional layers utilize multiple filters to extract features (feature maps) from the dataset, enabling the preservation of their spatial information. Pooling, also known as subsampling, is used to reduce the dimensionality of feature maps obtained from the convolutional process. Common pooling operations in CNN include max pooling and average pooling [99]. The Fully-Connected Layer is accountable for classifying inputs based on the features extracted by the preceding layers. While ReLu functions are commonly employed by convolutional and pooling layers for input classification, FC layers usually employ a softmax activation function to generate probabilities ranging from 0 to 1.

Using the above figure, we know that there are:

- Convolution layer, where convolution happens.
- Subsampling layer, where the pooling process happens.
- Fully Connected Layers.

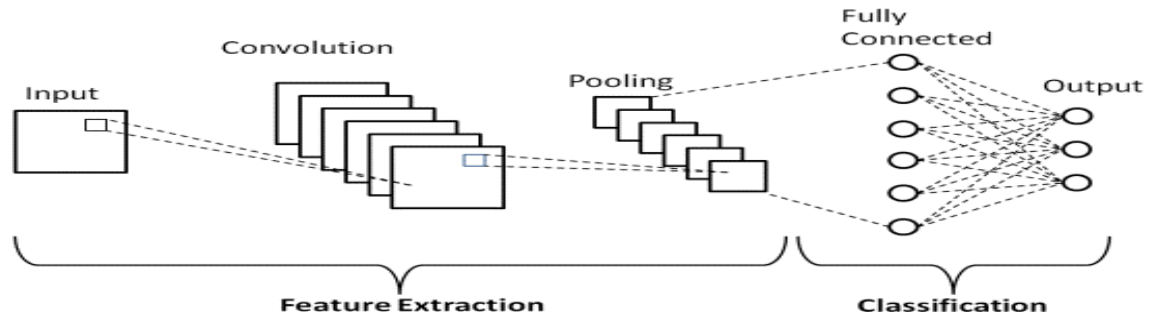


Figure 1.8: CNNs Architecture .

Zhang and Sabuncu [97] mentioned that the typical approach to training CNNs for classification involves using stochastic gradient descent along with Cross Entropy (CE) loss.

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are neural network structures mainly used for identifying patterns within sequential data. This data can include handwriting, genomes, text, or numerical time series often found in industrial environments such as stock markets or sensor readings [21] [65]. The main difference between Recurrent Neural Networks (RNNs) and Feedforward Neural Networks, also known as Multi-Layer Perceptrons (MLPs), lies in how information is passed through the network. While Feedforward Networks transmit information through the network without cycles, RNNs have cycles and transmit information back into themselves. This allows RNNs to expand the functionality of Feedforward Networks to consider previous inputs as well, not just the current input. The RNN architecture is shown in Figure 1.9 .

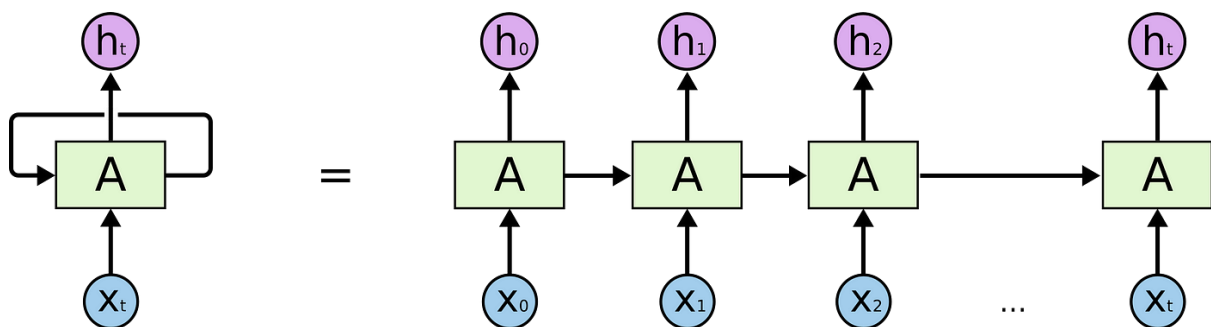


Figure 1.9: RNN Architecture.

If you know how to train Feedforward Neural Networks using backpropagation, you might wonder how to do it with Recurrent Neural Networks (RNNs). One way is to use a tech-

nique called Backpropagation Through Time (BPTT), which adapts the backpropagation algorithm for RNNs [92]. Essentially, it unfolds the RNN into a regular Feedforward Neural Network, allowing us to use backpropagation. However, as the number of time steps increases, BPTT can become very computationally expensive.

Long Short Terme Memory (LSTM)

Long Short-Term Memory Units (LSTMs) were created to effectively handle the vanishing gradient problem [37]. By using a more consistent error, LSTMs enable Recurrent Neural Networks (RNNs) to learn from significantly more time steps, often exceeding 1000 [65]. To achieve this, LSTMs store additional information outside the conventional flow of the neural network, within structures known as gated cells [16]. To ensure proper functioning of an LSTM, we employ three types of gates: an output gate (Ot) to regulate information flow out of the cell, an input gate (It) to control data input into the cell, and a forget gate (Ft) to manage the retention or removal of information within the cell.

They are mentioned in Figure 1.10.

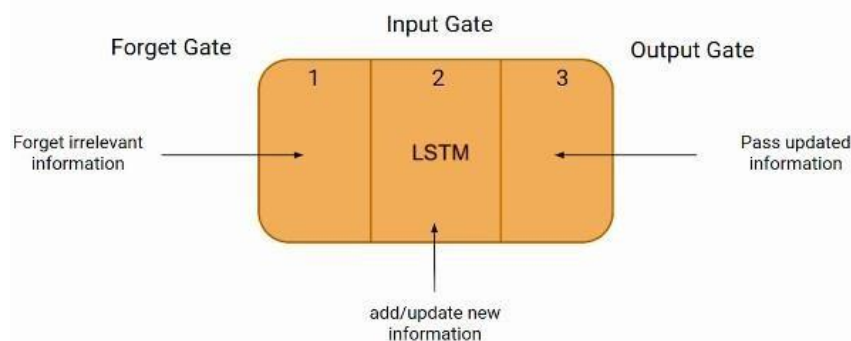


Figure 1.10: LSTM Architecture.

Gated recurrent unit (GRU)

The Recurrent Control Unit (GRU) is a control mechanism in recurrent neural networks, introduced in 2014 by Cho et al. [19]. GRU is like a long short-term memory (LSTM) with an activation mechanism to input or forget certain features [30], but it lacks context vectors or output gates, resulting in fewer parameters than LSTM [73]. The performance of GRU on several polyphonic music models, speech signal models, and natural language processing tasks is found to be similar to LSTM [71] [80]. GRU found that controls in general were indeed useful, and Bengio's team reached no specific conclusion about which of the two control units was better [20] [31].

A GRU cell keeps track of important information maintained throughout the network. The GRU network achieves this using the following two ports: Reset Port and Update Port. As shown in figure 1.11, a GRU cell has two inputs: The previous hidden state and the entry in the current timestamp. The cell combines them and passes them through the update and reset ports. To get the output at the current time step, we need to pass

this hidden state through a dense layer with softmax activation to predict the output. In doing so, a new hidden state is obtained and then propagated to the next time step.

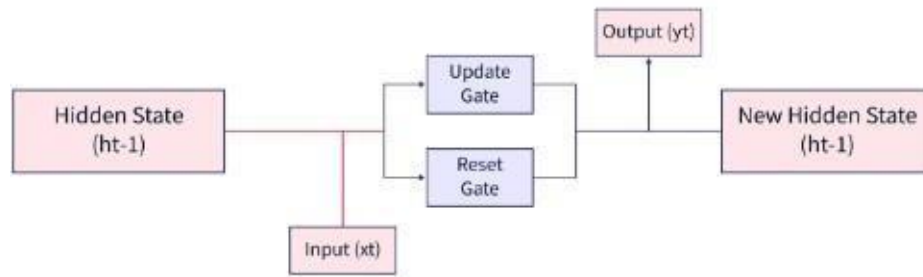


Figure 1.11: GRU Architecture.

DenseNet network

DenseNet networks are convolutional neural networks that follow a specific architecture commonly referred to as densely connected convolutional networks (DenseNet). A growing challenge with this type of network is performance optimization. A naive solution would be to simply stack more convolutional layers: an obvious problem that arises from this approach is the increase in network depth. When backpropagating gradients through the layers, one is essentially performing an operation on the partial derivatives of each hidden layer of the network, which can potentially complicate weight updates during the training phase.

The key specificity of this architecture lies in the input to the layers, which concatenates all inputs from the previous layers (see figure 4.10): thus creating a feature map while maintaining the same spatial resolution this is referred to as channel-wise concatenation.

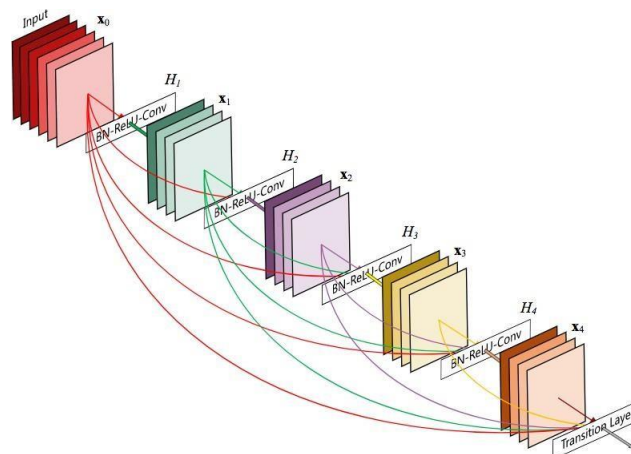


Figure 1.12: DenseNet Architecture.

1.2 Transfer Learning

Transfer learning is a machine learning research problem that involves saving knowledge acquired while solving one problem and applying it to a different but related problem [86]. Humans naturally have the ability to use what they know from one thing to understand another. When we meet a new challenge, we don't need to start over. Instead, we use what we've learned before to help us figure out the new task faster and better. Unlike regular machine learning, where we learn one thing at a time without looking at other areas, transfer learning takes what we know from other areas and uses it to help us learn something new, as shown in Figure 1.13.

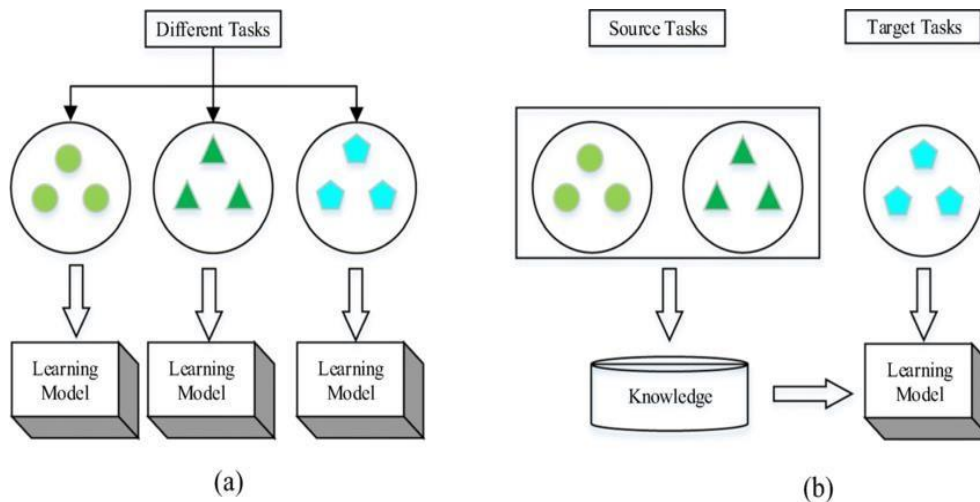


Figure 1.13: Difference between Traditional ML and TL.

Pan and Yang [66] introduce some notations and definitions such as domain, task, and marginal probabilities to understand TL. They are defined as follows: A domain D contains two elements, feature space, χ , and marginal probability, $P(X)$, where $X = \{x_1, x_2, \dots, x_n\} \in \chi$ is a sample data point. Thus, they represent the domain mathematically as $D = \{\chi, P(X)\}$. A task T can be defined as a two-element tuple of the label space, Y , and objective function $f: \chi \rightarrow Y$. The function f is used to predict the corresponding label $f(x)$ of a new instance x . This task, denoted by $T = \{Y, f(x)\}$, is learned from the training data consisting of pairs $\{x_i, y_i\}$, where $x_i \in \chi$ and $y_i \in Y$. It can also be denoted as $P(Y|X)$ from a probabilistic view point.

Given a source domain D_S a target domain D_T and learning task T_T , where $D_S \neq D_T$, or $T_S \neq T_T$, TL aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S [55].

According to Pan and Yang [66], The following three crucial questions must be answered throughout the TL process:

- **What to transfer:** This question involves determining which specific knowledge needs to be moved from the source to the target domain.
- **When to transfer:** This means understanding when it's appropriate to use transfer learning. Also, it's important to know when it's not a good idea to transfer

Knowledge, like when the source and target areas are not similar. Trying to force a transfer in such cases might not work well and could actually make things worse, which is called negative transfer.

- **How to transfer:** This refers to the method of moving knowledge and adjusting the target algorithm to enhance its performance

1.3 Federated Learning

Federated Learning (FL) [45] has emerged as a revolutionary paradigm for collaborative machine learning, enabling multiple entities to jointly train a global model without the need to share raw data. . It was first proposed by Google in 2016 [58], which improves scalability, reduces the costs associated with data transfer and storage, and allows real-time model updates by utilizing data directly at its source. Comprehensive surveys of the concepts and applications of FL are available in the literature [90] [51].

Compared to traditional machine learning, federated learning offers numerous benefits. It ensures data privacy and security by eliminating the requirement for data centralization, Thus facilitating adherence with data protection regulations such as the European Union’s General Data Protection Regulation (GDPR) [38] and the United States’ California Consumer Privacy Act (CCPA) [63]. Therefore, here is the Structure of federated learning as shown in Figure 1.14

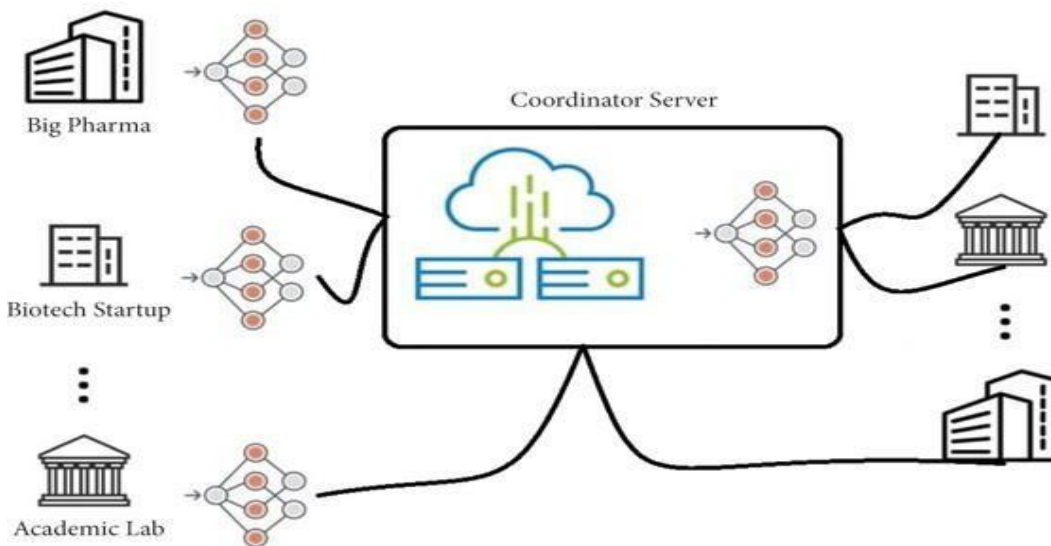


Figure 1.14: Structure of federated learning.

1.3.1 Federated Learning Approaches

As shown in Figure 1.15, FL approaches can be categorized into three types [28]: horizontal federated learning, vertical federated learning, and federated transfer learning.

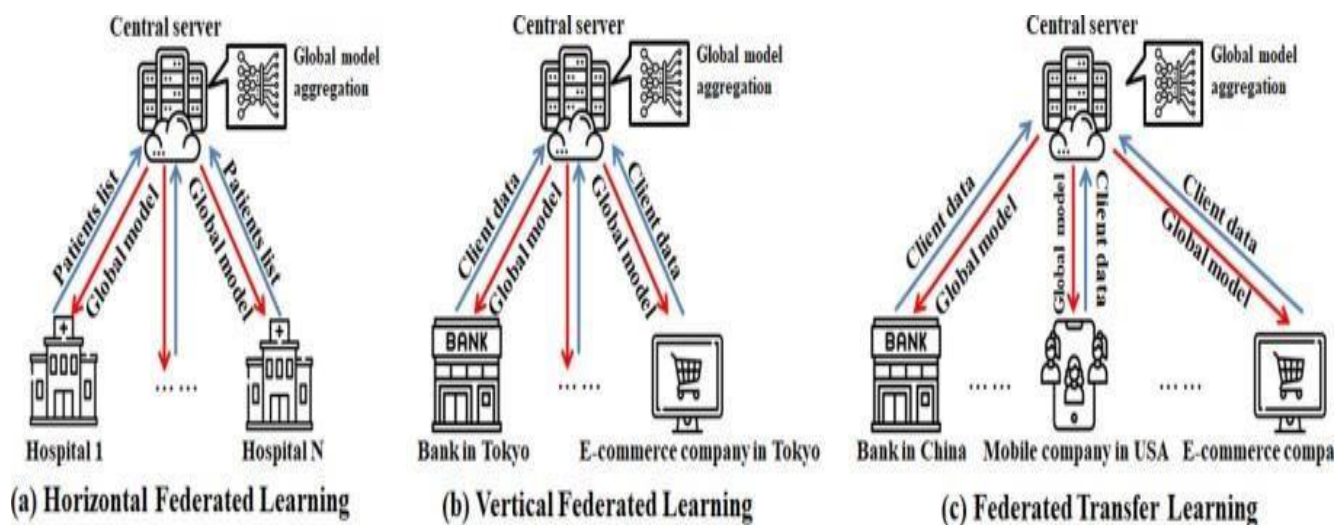


Figure 1.15: Classification of federated learning.

- **Horizontal FL**

Occurs when various datasets from different clients share the same features but differ in the samples they represent. For instance, datasets from different hospitals may share patient information (feature space) but differ in the specific patient data they contain (sample space).

- **Vertical FL**

Deals with clients that have data sharing the same sample space but differing in feature space. For example, this could involve the bank statements and shopping history of the same group of individuals

- **Federated transfer learning**

Applies to multiple datasets that differ both in the sample space and feature space.

Degree of federated Learning

In the participants' context, FL can be classified into "Cross-Silo" and "Cross device" that depend on the number and the power of the participants.

- **Cross-Silo Federated Learning**

Multiple organizations collaborate to train a machine learning model using their respective local datasets. It is suitable when a limited number of entities, such as companies or hospitals, join forces to collectively improve the model's performance without directly sharing their raw data.

- **Cross device Federated Learning**

In this architecture, numerous devices are involved, and each device has a relatively small amount of data and a relatively low computing power. Computing power compared to the cross-silo FL.

Network topology

- **Centralized FL**

In the centralized federated learning framework setting Figure 1.16, a central server plays a pivotal role in managing the various stages of the algorithms and coordinating the participating nodes throughout the learning procedure. The server’s responsibilities include selecting nodes at the onset of the training phase and aggregating the model updates received from these nodes. However, because all selected nodes must transmit updates to a solitary entity, the server can potentially become a bottleneck in the process

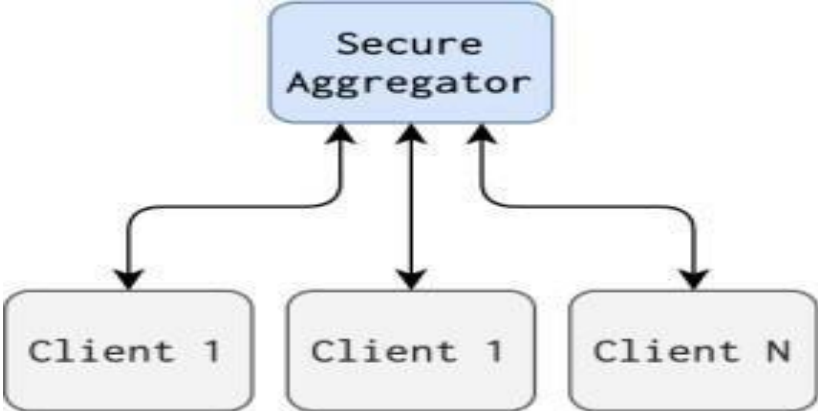


Figure 1.16: Centralized Federated Learning Architecture.

- **Peer to Peer Decentralized Federated Learning Architecture**

In the decentralized, federated learning setting 1.17, nodes have the ability to self-coordinate in acquiring the global model. This setup prevents single point failures as the model updates are exchanged only between interconnected nodes without the central server’s orchestration. However, it’s worth noting that the particular network topology could impact the efficiency of the learning process.

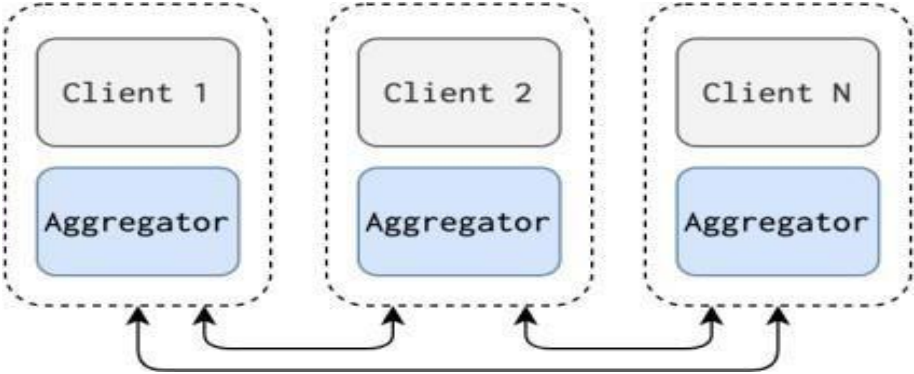


Figure 1.17: Peer to peer decentralized Federated Learning Architecture.

1.3.2 The Life cycle of a Model in Federated Learning

The objective of the FL process is to create an optimal model for a specific application. A standard procedure involves:

1. **Problem identification:** The first step involves identifying the problem to be addressed using Federated Learning (FL) and selecting the most suitable machine learning model.
2. **Device selection:** Depending on the available online devices, they are either randomly chosen or selected through another mechanism to participate in the learning process.
3. **Local training:** Initially, each device initializes the parameters of its model and trains it until it converges.
4. **Upload the local models:** Following local training, all connected devices upload their model parameters to the central server using secure communication methods, which will be discussed later.
5. **Model aggregations** Once the aggregator (central server) receives all local models, it aggregates the parameters to update the new, global, and optimal model.
6. **Broadcast the global model:** The server redistributes the parameters of the global model, and the devices update their parameters accordingly.

This process is repeated until the entire training process converges.

1.3.3 Federated Optimization Algorithms

The objective of FL is to optimize the weight parameters \mathbf{w} of the global model that minimizes the loss function values for all local models:

$$L(\mathbf{w}) = \frac{\sum_{i=1}^N |D_i| f_i(\mathbf{w})}{\sum_{i=1}^N |D_i|} \quad (1.22)$$

Where f_i denotes the loss function of the model trained by device i local dataset D_i .

The Federated Averaging Algorithm (FedAVG)

McMahan et al.[58] introduced the FedAVG approach in 2017, which is shown in Algorithm below. The FedAvg algorithm [49] is currently the most commonly used federated learning optimization algorithm. Unlike the conventional optimization algorithm [41], its essential idea is to use the local stochastic gradient descent method to optimize the local model on the data holder and perform aggregation operations on the central server-side [23].

Chapter 1. Background

Algorithm 1 Federated Averaging (FedAVG) algorithm [58]

C is the fraction of clients selected to participate in each communication round. The \mathbf{K} clients are indexed by \mathbf{k} ; \mathbf{B} is the local mini-batch size, P_k is the dataset available to client \mathbf{k} , \mathbf{E} is the number of local epochs, and η is the learning rate.

```

1: Procedure FedAVG d run on the server
2: Initialize  $w_0$ 
3: for each round  $\mathbf{t} = 1, 2, 3, \dots$  do
4:    $m \leftarrow \max(C \cdot K, 1)$ 
5:    $S_t \leftarrow$  (random set of  $m$  clients)
6:   for each client  $\mathbf{k} \in S_t$  do
7:      $w_t^k \leftarrow \text{ClientUpdate}(\mathbf{k}, w_t)$  d In Parallel
8:   end for
9:    $w = \sum_{k=1}^K \frac{n_k}{n} w_t^k$ 
10: end for
11: End procedure FedAVG
12: Procedure ClientUpdate( $k, w_t$ ) d run on client  $\mathbf{k}$ 
13:  $\beta \leftarrow$  ((Split  $P_k$  into mini-batches of size  $\mathbf{B}$ )
14: for each local epoch  $\mathbf{i}$  from 1 to  $\mathbf{E}$  do
15:   for batch  $\mathbf{b} \in \beta$  do
16:      $w \leftarrow w - \eta \Delta(w, b)$ 
17:   end for
18: end for
19: return  $\mathbf{t}$  to the Server
20: End procedure ClientUpdate( $k, w$ )

```

Federated Learning with Personalization Layers (FedPer)

The FedPer algorithm works similarly to FedAvg in updating weights in combined models. However, it differs significantly in which parts of the model are considered during combination. FedPer splits the model into basic and personalized layers [9]. While only the basic layers are sent to the server for aggregation, the personalized layers are kept local. In this approach, transfer learning techniques are used by the federated server. For instance, in a two-layered CNN, the last dense layer serves as the personalized layer, remaining on the client side and not sent to the server. The server only aggregates the basic layers, and the client trains the lower layers using the federated learning approach. The global approach is illustrated in figure 1.18.

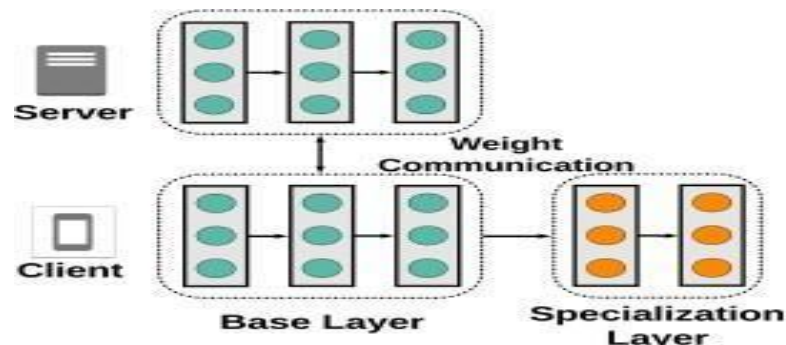


Figure 1.18: FedPer illustration.

Federated Genetic Algorithms

The **FedGA** algorithm, introduced by Guendouzi et al. [33], is an aggregation technique employing genetic algorithms to address the issue of data and model diversity. In FedGA, participants upload only the base layer weights to the central server. Subsequently, the server calculates new weights using genetic algorithms, treating the weight vector as a chromosome. Assuming a central repository of public data on the aggregator server for building the global model, alongside private datasets from various collaborators as target domains, FedGA-ICPS and other federated learning aggregation methods may encounter challenges due to domain shifts. Additionally, refining only segments of neural network models isn't optimal for decision-making. Furthermore, deploying FedGA-ICPS with numerous collaborators can escalate time complexity due to the growing number of collaborators, increasing the algorithm's complexity.

1.3.4 Federated Learning Challenges

Li et al.[53], describe four fundamental challenges associated to addressing the distributed optimization problem. These challenges set apart the federated setting from other classical problems, like distributed learning in data center environments or traditional private data analyses.

Expensive Communication

Federated networks can consist of a vast number of devices, such as millions of smart devices. Consequently, communication between these devices and the central server is crucial in distributed networks. To further minimize communication in such a context, two key aspects must be considered: (i) reducing the total number of communication rounds, or (ii) reducing the size of transmitted messages in each round.

Systems Heterogeneity

Due to variations in device specifications such as processor and memory, as well as differences in network connectivity and power, the storage, computational, and communication capabilities of each device in distributed networks may vary. Given that distributed networks are large in scale, each device may fail to participate due to connectivity issues or power constraints. Therefore, distributed learning methods developed and analyzed must : (i) anticipate limited participation, (ii) tolerate heterogeneity in device specifications, and (iii) be robust to devices dropping out of the network.

Statistical Heterogeneity

Distributed networks generate and collect data in a non-identically manner across devices, violating assumptions of uniform and identical distribution. There may be varying numbers of data points and an underlying structure connecting devices and their distributions, increasing modeling and analysis complexity. Although distributed learning aims to learn a single global model, alternatives such as learning local models simultaneously exist. Current approaches allow for personalized or device-specific modeling, aiding in handling data variance more naturally.

Privacy Concerns

Privacy is often a major concern in applications using federated learning. By sharing model updates, federated learning takes a step towards safeguarding data generated on each device. Modern methods aim to enhance privacy in federated learning using tools such as secure multiparty computation or differential privacy. Understanding these exchanges and achieving a balance between them, both theoretically and empirically poses a significant challenge in realizing private federated learning systems.

1.4 ICPS

According to Alguliyev et al. [7], Cyber-Physical Systems (CPS) are capable of effectively integrating physical and digital components using modern technologies of sensors, computing, and networks.

They consist in grouping several concepts such as, Automation, Connection, Cloud Computing, Internet of Things (IoT), Big data and System Integration.

CPSs are the basis for the development of the following areas: smart manufacturing, smart medicine, smart buildings and infrastructures, smart city, smart vehicles, wearable devices, mobile systems, defense systems, meteorology, etc.

To build an ICPS, it is necessary to define some network architectures as it's shown in Figure 1.19 that presents IoT, Cloud Computing, Fog Computing, and Edge Computing layers.

1.5 IoT

The Internet of Things (IoT), also known as the Internet of Objects, is a network of interconnected physical objects capable of collecting and exchanging data. Equipped with sensors, software, and other technologies, these objects connect to the Internet or other communication networks. The term "Internet of Things" describes scenarios where Internet connectivity and computing capability extend to various objects, devices, sensors, and everyday items. [74].

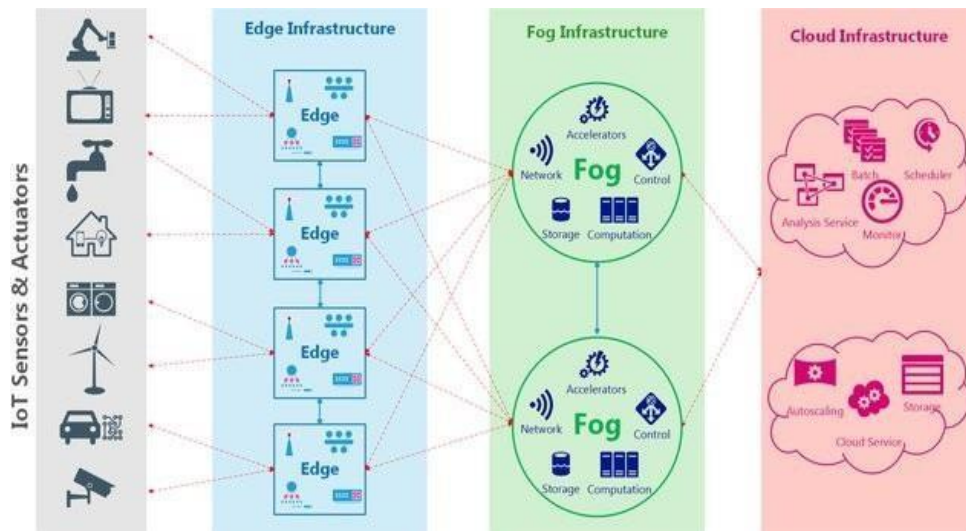


Figure 1.19: Example of a proposed IoT-Edge-Fog-Cloud architecture [14].

1.5.1 IoT Architecture

Concerning to IoT design architecture a study by Trappey et al. [83] established a logical framework based on layers to classify IoT technology and used it to describe and identify sensitive information systems. According to several researchers [52],[35]-[12] common layer designs in traditional IoT architecture networks consist of four main layers, as illustrated in Figure 1.20, which include the following layers:

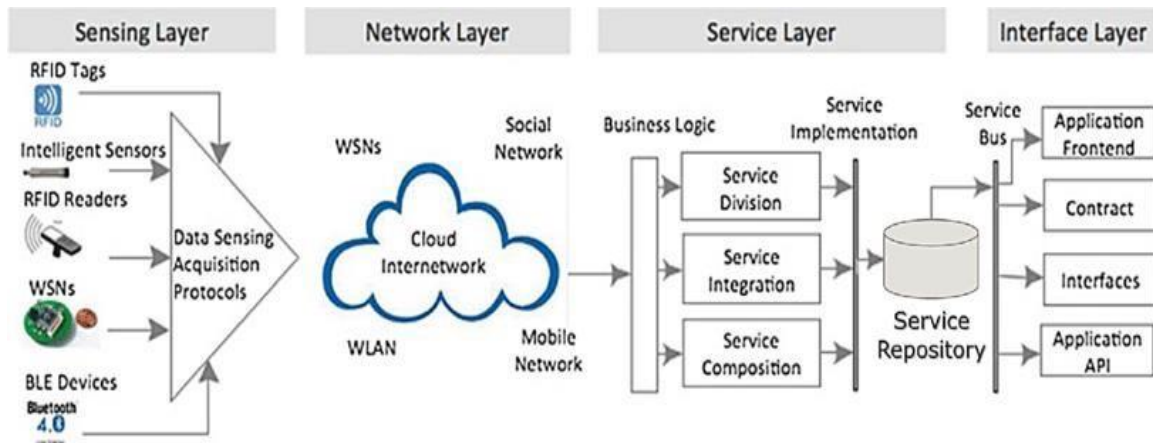


Figure 1.20: IoT SOA Architecture [6].

Sensing Layer

Sensing Layer plays a crucial role in capturing data from the physical environment and transferring it to the network layer through gateways. It is designed to sense the status of various "things," integrating actuators, sensors to serve as diverse types of objects. Sensors, which measure variables like temperature, pressure, movement, and more, are instrumental in this process. Actuators, on the other hand, initiate actions

autonomously. In deploying the perception layer, certain considerations must be adhered to, including cost-effectiveness, size constraints, energy efficiency, deployment flexibility, handling heterogeneity, efficient communication, and network integration.

Network Layer

In the realm of the Internet of Things (IoT), the network layer stands as the foundational support infrastructure, orchestrating the seamless flow of information from the sensor layer to the application layer. It serves as the backbone of communication, enabling the exchange of data across wired or wireless networks.&

Service Layer

In the **IoT** systems, once the data traverses the network layer, servers within this layer receive the data through Application Programming Interfaces (**APIs**) and various protocols and applications. Subsequently, they process, store, and execute actions based on this data. These actions can be physical, involving actuators, or logical, such as intelligent decisions, all with rapid response times. This layer is divided into 4 principal components [22]:

- **Service discovery:** It identifies objects that can provide the necessary services and information promptly.
- **Service composition:** It enhances service quality and reliability by establishing connections between multiple objects.
- **Trustworthiness management:** Its objective is to establish trust systems capable of analyzing and integrating data from various services to build a dependable system.
- **APIs Services:** They facilitate interactions among services within the IoT ecosystem.

Interface Layer

It is designed to simplify the interconnection and management of "things" while providing users with clear and comprehensible information to interact with the system seamlessly.

1.6 Cloud

Cloud computing aims to enable access to a shared computing resources with minimal management effort or interaction with the service provider [59].

Cloud computing involves a network of servers distributed across different geographical locations. These servers are rented as needed to provide computer resources like storage, processing power, and networking capabilities. These resources are accessible from anywhere via the Internet.

1.6.1 Characteristics

According to Mell et al. [59], Cloud Computing consists of the following five characteristics:

- **Self-service on demand:** This feature enables users to access cloud resources as required, without the need for human interaction between users and service providers.
- **Broad network access:** This feature enables users to remotely access the cloud using various platforms, such as mobile devices, laptops, tablets, and more, without encountering platform-specific restrictions.
- **Resource pooling:** Within cloud computing, multiple clients have the ability to concurrently share both physical and virtual resources in a dynamic manner. This implies that clients can use resources based on their individual consumption requirements, such as storage bandwidth, processing power, memory, and network access. Security measures are implemented to safeguard data handled within shared resources, using standard tools deployed on multi-user servers.
- **Rapid elasticity:** This characteristic enables the cloud to rapidly scale up or down, efficiently allocating and releasing resources in response to customer demands, regardless of quantity or timing.
- **Measured service:** This service enables the use of resources under rigorous monitoring and control, ensuring transparency between the provider and the customer. Payment is made based on the type of service, capacity, and usage quantity.

1.6.2 Service Models

National Institute of Standards and Technology (NIST) has categorized cloud services into three models (layers) according to the type of service they offer: software, platform, or infrastructure [59] as it is shown in Figure 1.21.

Software as a Service (SaaS)

Software as a Service (SaaS) is a software distribution and delivery model in which a fully functional and comprehensive software product is provided to users over the internet on a subscription basis. SaaS finds applications in various domains including email services, customer relationship management (CRM)¹, communication tools, virtual desktop solutions, and gaming platforms.

Platform as a Service (PaaS)

Platform as a Service (PaaS) provides users to deploy their own applications or acquired software into the cloud infrastructure. These applications use programming languages,

¹Customer Relationship Management.

Chapter 1. Background

libraries, services, and tools provided by the platform. Unlike traditional setups, users do not oversee or govern the underlying cloud infrastructure, encompassing networks, servers, operating systems, or storage. Nonetheless, they maintain authority over the deployed applications and can adjust settings for the application-hosting environment as needed.

Infrastructure as a Service (IaaS)

This model revolves around the hardware aspect of the cloud. Customers lease and manage cloud infrastructure resources, including servers, computing power, network components, storage capacity, and memory, instead of specific services.

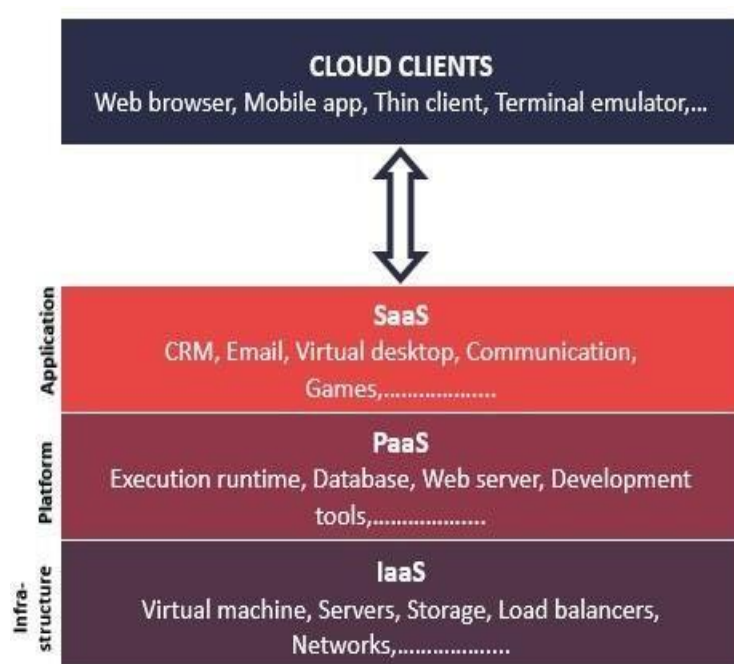


Figure 1.21: Cloud Computing Models [43].

1.6.3 Cloud Solutions

According to Mell et al. [59], Cloud computing can be deployed in four models: private, public, community, and hybrid clouds. These models correspond to various uses.

Public Cloud

The public cloud is typically available to the general public via the Internet and used by clients, universities, companies, or any organization to secure the data of each entity. Examples of public cloud providers include Google Cloud, Microsoft Azure, Amazon, and iCloud.

Private Cloud

The cloud infrastructure is designed for a single organization, serving multiple clients. Examples include Eucalyptus, OpenNebula, and OpenStack. This type of cloud solution can either be internal, with resources hosted and managed within the organization's own infrastructure, or external, where resources are hosted and deployed by a service provider.

Community cloud

The community cloud infrastructure is tailored for use exclusively by a specified community of clients from organizations that share common concerns. For instance, organizations like Amadeus ², GSA (General Services Administration)³, and CMed ⁴ have implemented community cloud solutions for their respective needs.

Hybrid Cloud

It is the combination of two or more different cloud infrastructures (private, community or public). For instance, an organization might possess its private cloud while also using certain services from a public or community cloud.

1.7 Fog

Fog computing, an intermediary layer within CPS networks, bridges the gap between IoT devices and Cloud datacenters. It addresses security and privacy concerns, which remain significant from a business perspective compared to cloud computing. By handling data processing tasks, it alleviates the communication load between IoT devices and the cloud, thereby reducing latency and network congestion. Fog computing represents a more advanced approach compared to cloud computing, offering enhanced performance for managing user requests and adhering to emerging standards [93].

A fog architecture comprises multiple edge nodes, often referred to as fog nodes, which have limited processing capabilities. These fog nodes possess relatively lower processing power and storage capacity. In a fog network, there are instances where both edge nodes and multiple servers are referred to as cloudlets [87][17]. These cloudlets are integrated into the shared computing environment, operating within the network edge. Using fog devices enables clients to achieve real-time responses for latency-sensitive applications.

1.7.1 Characteristics

According to Iorga et al. [42], The core attributes that delineate fog computing from other computing paradigms encompass six essential characteristics. Nonetheless, IoT devices are not obligated to employ all these features while using a fog computing service.

²Amadeus is the world's largest travel company with more than 150 airlines

³GSA includes US government agencies

⁴CMed is a startup that was launched in 2010 for pharmaceutical companies

- **Low latency:** As fog nodes frequently reside in closer proximity to IoT devices, the processing and response to data originating from these devices occur notably faster compared to using cloud services.
- **Geographical distribution:** The distribution and identification of fog computing nodes must be strategically planned to ensure effective control over mobile objects and maintain a high level of quality in service delivery.
- **Heterogeneity:** Fog computing facilitates various forms of network communication to ensure efficient data processing.
- **Interoperability and federation:** Components within fog computing systems need to seamlessly work together, while services should extend across different domains, enabling collaboration among various providers, particularly for real-time streaming services. services that require the cooperation of different providers.
- **Real-time interactions:** Fog computing enables the processing of data and the delivery of responses to end users in real time.
- **Scalability and agility of federated, fog-node clusters:** Fog computing displays adaptability at both the cluster and cluster-of-clusters levels, accommodating elastic computing, resource pooling, fluctuations in data load, and changes in network conditions, among other adaptive functionalities it supports.

The main distinction between fog and cloud computing lies in their network architectures: the cloud operates within a centralized network, while fog uses a decentralized distributed infrastructure. Furthermore, various distinctions exist between these architectures. For instance, fog computing offers more advanced advantages compared to cloud computing. For example, fog computing delivers quicker responses, whereas cloud responses tend to be slower. Additionally, parameters such as request types, data transmission, security measures, user and resource management, scheduling, interoperability, failure management, service pricing, and service types are superior in fog computing when compared to cloud computing [13],[1],[2],[34].

1.7.2 Fog Node Attributes

To enable the good deployment of a fog computing, fog nodes should possess one or more of the following attributes:

- **Autonomy:** Fog nodes have the capability to function autonomously, allowing them to make local decisions at either the individual node level or within clusters of nodes.
- **Heterogeneity:** Fog nodes are available in various physical configurations and can be installed in diverse environments
- **Hierarchical clustering:** Fog nodes facilitate hierarchical architectures, where distinct layers offer various subsets of service functions, collaborating seamlessly as a unified entity

- **Manageability:**Fog nodes are overseen and coordinated by complex systems capable of automating the majority of routine tasks.
- **Programmability:**Fog nodes possess inherent programmability across multiple levels, allowing various stakeholders such as network operators, domain specialists, equipment suppliers, or end users to engage in programming activities.

1.7.3 Fog Node Architectural Service and Deployment Models

Similar to the traditional cloud computing model, fog computing offers architectural implementations across multiple layers of the network's topology. Just as defined in NIST's cloud computing service models, fog computing can implement Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Additionally, fog computing, being an extension of the traditional cloud-based model, supports various deployment models such as Private fog nodes, Community fog nodes, Public fog nodes, or Hybrid fog nodes.

1.7.4 Architecture

Based on our examination, it was discovered that researchers have suggested three [57], four [10], five [24], and six [2] layers within the architecture of Fog computing. In this section, we explore the diverse elements comprising the architecture of Fog computing as outlined by Naha et al. [62] and depicted in Figure 1.22. These components are categorized into multiple groups according to their respective functionalities, which are defined as layers. These functionalities facilitate communication between IoT devices, various Fog devices, servers, gateways, and the cloud. A comprehensive description of each layer follows below:

Physical layer

The physical layer is tasked with gathering data from various sensor types: Physical sensors include smart devices, temperature sensors, humidity sensors, etc., while virtual sensors are mathematical functions that merges data from multiple sensors to estimate a new quantity not directly measurable by a physical sensor.

Fog device, server, and gateway layer

The Fog device, server, and gateway layer consist of clusters of fog devices that connect to a group of sensors, fog servers that aggregate these fog device clusters, and gateways for communication purposes. Within this layer, a designated cluster of fog devices, connected to the same server, can communicate with each other to collaborate on processing and decision-making tasks. Additionally, this layer oversees hardware and storage configurations, device and server connectivity, and computation requirements requested by various applications.

Monitoring layer

It comprises three vital components: the monitoring system, which is tasked with closely monitoring system and resource performance, services, and responses, and selecting suitable resources during operation. The resource demand component monitors current resource usage and may forecast future resource requirements based on current consumption and user activities. Performance prediction monitors based on system load and resource availability are employed to predict Fog computing performance, ensuring that service level agreements meet appropriate Quality of Service (QoS) requirements.

Pre and post-processing layer

This layer is accountable for data cleansing and validation through various components that analyze, filter, segment, and reconstruct the data.

Storage layer

It comprises two components: the storage module, which stores data using storage virtualization, and the data backup component, which ensures data availability.

Resource management layer

The components within this layer oversee resource allocation and scheduling, in addition to addressing energy-saving concerns. The reliability component ensures the dependability of application scheduling and resource allocation. In periods of peak demand, when resource utilization is high, scalability ensures the expandability of Fog resources.

Security layer

This layer ensures data confidentiality by using encryption tools and authentication techniques.

Application layer

Fog computing has become indispensable in many recent application domains to facilitate rapid responses and real-time control. The protocols used in this layer are consistent with those discussed in section 1.7.

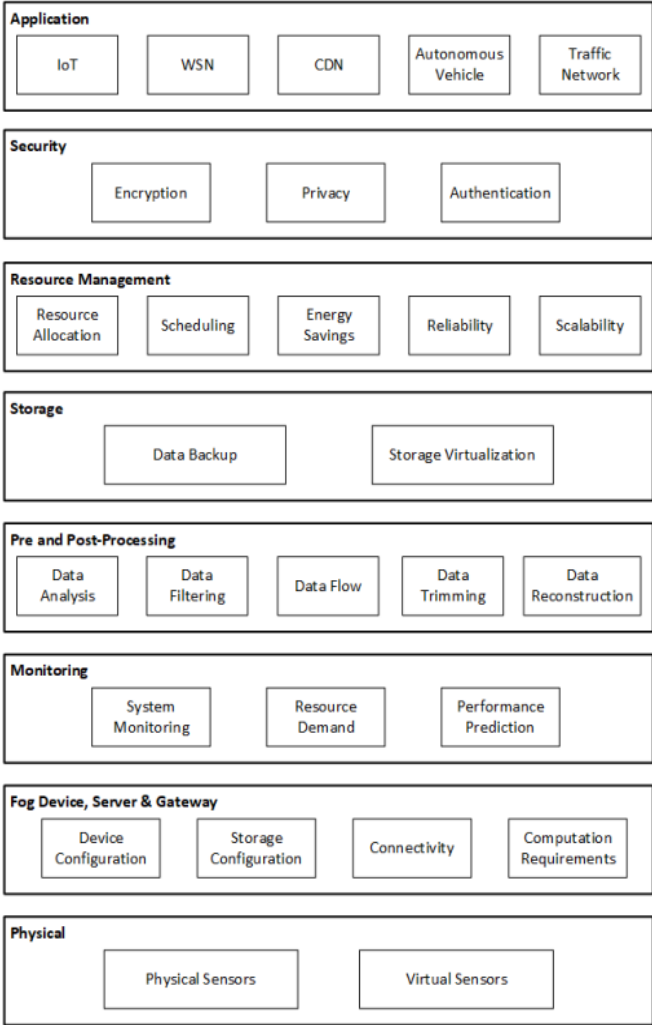


Figure 1.22: Fog Computing Architecture [62].

1.8 Edge

Edge computing is an evolved version of cloud computing, aiming to reducing latency by bringing services in close to end-users. Edge computing is defined by its high bandwidth, extremely low latency, and immediate access to network information, all of which are used by various applications [84] [72]. It is described as a small-scale data center that stores and processes data collected directly from the device generating it, minimizing the transmission of irrelevant or sensitive data over the network. It shares similar characteristics with Fog Computing but is positioned nearest to users.

Edge computing is characterized by its rapid processing and swift application response time, which are critical criteria in surveillance, virtual reality, and real-time traffic monitoring applications [48].

1.8.1 Architecture

According to IBM ⁵, Edge Computing comprises two essential elements, outlined as follows:

Data source

This represents the data collected by IoT devices and sensors, which is then transmitted to edge gateways via communication protocols. It is diverse and contingent on the device and its environment.

Edge nodes

These nodes are positioned between IoT sensors and the core network (cloud). There are three types of edge nodes, as depicted in Figure 1.23 ⁶ and elaborated below:

- **Edge Devices:** These encompass various devices such as smartphones, robots, and intelligent machines, used for local processing. They store their data locally to ensure confidentiality and conduct local computations based on predefined traditional or artificial rules set by the datacenter.
- **Edge server or gateway:** Serving as leaders of multiple edge devices, these devices control and deploy applications on them. They function as communication point for numerous devices.
- **Edge network or microdata center:** Also known as a micro-data center, the edge network can be likened to a local cloud to which devices connect. It offers data analytics and storage capabilities, reducing the distance for data transfer, thus addressing latency and bandwidth issues.

⁵<https://www.ibm.com/cloud/architecture/architectures/edge-computing/>

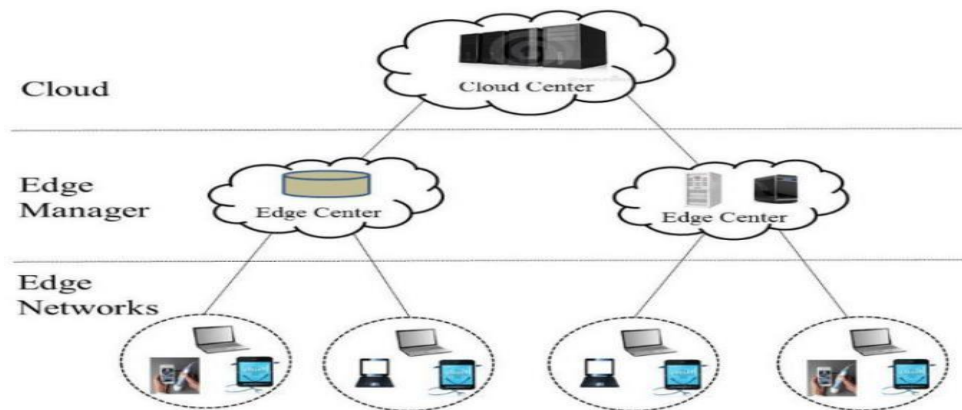


Figure 1.23: Edge Computing Architecture.

1.8.2 Features

The following features are the requirement that an edge solution have to consist:

Connectivity

To ensure connectivity among Edge devices, an Edge solution must support the most prevalent protocols. Examples of these protocols include Z-Wave, ZigBee, KNX, Bluetooth LE, LoRa, and others.

Local Data Processing

An edge solution must ensure that all edge devices can execute applications deployed by Edge servers within a predefined domain, adhering to clear constraints.

Remote monitoring

An edge solution is required to monitor, control, and issue commands to gateways and the devices connected to them. To facilitate this functionality, an open API should enable remote applications to communicate with the gateway using REST, Web Sockets, or JSON-RPC⁷ protocols.

Portability

The software deployed in edge computing must be adaptable, capable of running on any device regardless of physical and system limitations.

⁷<https://www.ibm.com/cloud/architecture/architectures/edge-computing/>

Security

An edge solution must ensure the implementation of security measures on all edge nodes, including permission-based access control, secure encrypted communication, certificate management, and other relevant techniques.

1.9 Conclusion

In this chapter, we have introduced the fundamental concepts of Machine Learning, providing a comprehensive overview of deep learning, including its various approaches and types such as CNNs, RNNs, and LSTMs. Subsequently, we delved into Transfer Learning, covering its definitions, categorizations, strategies, and real-world applications. Furthermore, we explored Federated Learning techniques, discussing their approaches, optimization algorithms, and challenges. Lastly, we outlined and compared various tools for Deep Learning and Federated Learning that can be utilized for our research. Additionally, we defined Industrial Cyber-Physical Systems, detailing concepts such as IoT, Cloud Computing, Fog Computing, and Edge Computing. We also mentioned different network and IoT tools applicable to our research. In the next chapter, we will present an overview of Industrial Cyber-Physical Systems and their deployment tools. Following that, we will delve into the current state of the art of deploying Federated Learning on various network architectures within CPS FL on different network architecture in CPS.

⁶<https://www.ibm.com/cloud/architecture/architectures/edge-computing/>

Chapter 2

State of the art

2.1 Introduction

The use of ML techniques described in Chapter 1 has simplified CPS operations, increased efficiency and productivity, and most importantly, ensured data security and integrity. However, to meet specific needs or address specific problems, contributions to the state of the art are made differently in each of the three architectures. This chapter evaluates several existing architectures and their vulnerabilities that use federation, broadcast, and DL methods to make decisions in the cloud, fog, and edge layers.

2.2 Cloud-based FL

Polap et al.[67] introduced a new agent-based approach designed to assess and safeguard medical data derived from the Internet of Medical Things (IoMT). The decentralized nature of the data is ensured through encryption on a blockchain platform. Within this framework, three key actors - Learning, Indirect, and Data Management (DM) agents - play integral roles. The Learning agent (LA) is tasked with initiating six threads aimed at training CNN models tailored to specific sections of the database. Meanwhile, the Indirect Agent (IA) is responsible for categorizing inputs from the DM agent and signaling LA when model retraining is necessary due to insufficient or ambiguous results. In cases where the training data prove inadequate, IA collaborates with the Data Management Agent (DMA) to acquire additional entries. However, it's worth noting that DMA's data collection process, particularly in gathering patient and doctor information, may introduce delays, thus compromising real-time functionality. Despite the implementation of Federated Learning (FL) techniques, the selection of an appropriate classification method remains challenging, especially given the noisy and heterogeneous nature of the data.

Tian et al. [82] used techniques from deep learning to propose an asynchronous Federated Learning (FL) based anomaly detection approach, termed Delay Compensated Adam, tailored for IoT devices with constrained resources. Their approach is structured into sequential stages, commencing with an initial phase. Initially, parameter pre-initialization is conducted, where a subset of clients is randomly selected to transmit a

small portion of their data to the server, facilitating the training of an initial global model. Subsequently, an asynchronous training methodology is implemented for the model. However, they did not assess the reliability of the participating nodes, despite the method being specifically designed for anomaly detection. Following the initiation of the three-task server, challenges arise due to high bandwidth demands, and system failure occurs if the server crashes, leading to termination of the entire system.

In the domain of wearable healthcare Chen et al.[18] introduced a concept of federated transfer learning fused with cloud computing. Their proposed framework manages the separation of Federated Learning (FL) data and customization of models. Initially, the cloud server constructs a global model utilizing public datasets, then disseminates it to clients employing homomorphic encryption. Subsequently, clients retrain their local models and upload the refined versions. The aggregator updates the global model using the fedAVG algorithm and incorporates transfer learning techniques.

Hao et al. [36] introduced Privacy-enhanced Federated Learning (PEFL) for Industrial AI, aiming to enhance the security of model gradients and the accuracy of local models. The approach involves three main components: (1) Key Generation Center (KGC), which distributes private keys to each participant, (2) Cloud-based aggregator (CS), and (3) participants. Each participant trains a local model, generates local gradients, and applies Differential Privacy (DP) to perturb them. The perturbed gradients are then encrypted into BGV ciphertext using Homomorphic encryption. The CS decrypts the encrypted gradients for aggregation. This approach ensures model privacy and security while maintaining accuracy, even when a small fraction of participants are affected.

Zhang et al.[96] proposed a method for preserving privacy in Federated Learning (FL) for disaster classification. They employed Paillier homomorphic encryption to safeguard the data of social computing nodes and Transfer Learning (TL) to mitigate computation and communication expenses. Initially, a dedicated server known as the Key Generation Center (KGC) distributed encryption keys to each node. Subsequently, each node fine-tuned its local model using TL, where it utilized a pre-trained model with extensive datasets, extracting only the feature extraction layers and training the classification layers. Following this, the node encrypted the weights of the higher layers and transmitted them to the global server (Cloud). Upon receiving the local weights of all nodes, the server aggregated them using the FedAVG algorithm. In their experiments, they utilized the Multimodal Damage Identification Dataset (MDI) [61], which was randomly divided among four social computing nodes. Results indicated that compared to training from scratch, the accuracy of various models improved by an average of 18.5% when employing TL with pretrained VGG-11, 13, 16, and 19 models from the ImageNet ¹ dataset. After completing 300 aggregation rounds, the accuracy of different FedTL models improved by an average of 20% compared to FL from scratch. Moreover, transferring only the classification layer parameters of the model can reduce resource costs and communication expenses for social computing nodes.

Kevin et al.[46] introduced a new federated transfer learning framework named FTL - CDP, aimed at cross-domain prediction. This framework combines the principles of Federated Learning (FL) and Transfer Learning (TL), leveraging the advantages of both to

¹<https://www.image-net.org/>

tackle the challenges of data scarcity and privacy encountered by many machine learning approaches, particularly in modern and heterogeneous smart manufacturing environments with cross-domain applications. Their system comprises heterogeneous application groups, each of which aggregates a collection of smart devices (SDs) sharing the same application. When a new component joins the Federated Learning network, the central server (CS) searches for a suitable base model using similarity measures. The SD then performs Transfer Learning, retraining its model and sharing its parameters with the CS. Instead of aggregating into one global model, aggregation occurs for each sub-global model related to an application group. In their experiments, they deployed a base model trained with the COCO ² dataset on the CS and conducted Transfer Learning on the SD. To demonstrate the efficacy of their system, they compared it with centralized learning and Federated Learning without Transfer Learning. Results revealed that FTL-CDP achieves a learning time 50% faster than other approaches, albeit with higher communication overheads due to the sharing of the base model. Furthermore, FTL-CDP exhibits faster convergence, particularly with small sample sizes.

2.3 Fog-based FL

Zhou et al. [98] proposed a fog computing approach for federated learning with data protection as a means to protect privacy. To complete the learning process, each fog node collects and analyzes data from IoT devices. Due to uneven distribution of data and gaps in computing resources, this strategy improves training efficiency, and uneven distribution of data improves model accuracy. Withstand data attacks on IoT devices by using blinding and homomorphic Perrier encryption to protect model parameters and using differential privacy to defend against data attacks on IoT devices. Their experiments are based on Fashion-MNIST ³ data and prove that their system is indeed very efficient.

Saha et al.[75] introduced the FogFL framework to implement federated learning (FL) at the fog computing layer, which aims to reduce communication latency and energy consumption of resource-constrained edge devices while improving system reliability. Their system architecture consists of a cluster of edge devices, with each cluster associated with a nearby fog node. First, each edge device trains a local model with private data and then sends local weights to the fog node for aggregation. The frequency of local updates at the edge device depends on the data size and the physical characteristics of the device. After local aggregation, each fog node sends its workload and communication latency parameters to the cloud server. Then, the cloud server calculates the logical variables for each Fog node and FL iteration, synthesizes the above parameters, and selects the Fog node with the minimum variables to act as a global aggregator. To evaluate the performance of the proposal, the authors deployed six of his Raspberry Pis and two computers with different physical characteristics as fog nodes, using Radio Access Network (RAN) technology. Each Fog node was equipped with his three Raspberry Pis as edge nodes, providing a single cloud system. The impact of his FogFL on the above challenges was evaluated by comparing with the results of other algorithms such as the FedAVG algorithm and the

²<https://cocodataset.org/home>

³<https://www.kaggle.com/datasets/zalando-research/fashionmnist>

hierarchical FL framework. This evaluation was performed using his MATLAB software, and the MNIST dataset was split into non-IID data for the client. The results show that FogFL reduces the delay by 85% and 68% compared to FedAVG and HFL, respectively, and reduces the energy consumption by 92% compared to FedAVG.

Wang et al. [85] proposed the use of fog computing for air quality monitoring by using heterogeneous datasets from multiple sources and applying a federated learning (FL) approach. This approach involves collecting data from multiple sources and using disparate multisource data collection methods. The system architecture includes IoT, edge nodes, and fog gateway devices (FGD) to form a local multi-source heterogeneous data fusion system “LMFS”. “CHTS” represents a centralized homogeneous training system consisting of IoT, edge nodes, and fog gateway devices. Within LMFS, five subclassifiers are deployed on fog nodes. Each subclassifier is associated with one or more heterogeneous datasets that undergo preprocessing before performing a common task. Numerical features extracted from all five layers of edge nodes are used to obtain local evaluation results.

Yao and Ansari [91] used fog computing to accelerate the Federated Learning (FL) process and reduce power consumption of IoT devices. They introduced a new FL improvement technique that leverages CPU frequency control and radio transmit power (WTP) control to streamline FL operation, shorten FL duration, and minimize energy consumption. The goal is to ensure that the FL completion time is less than the maximum allowed FL time. This includes both the computation time for local model training and wireless transmission time for uploading local model updates to fog nodes to meet quality of service (QoS) requirements. Similarly, energy consumption during FL operation must be minimized. Yao and Ansari implemented the Alternative Direction Algorithm (ALTD) at each local iteration to calculate optimal WTP and CPU frequency values for all IoT devices tested.

2.4 Edge-based FL

Qu et al. [70] proposed a novel system for Cognitive Computing in Industry 4.0 Networks, pointing to upgrade the execution of Industry 4.0 fabricating through decentralized Blockchain-enabled Unified Learning (FL). This system addresses key challenges such as information security, productive handling, incentivizing support in learning, and avoiding harming assaults. Inside this system, each device within the arrange holds its private information, guaranteeing information security and proficient processing through show sharing instead of crude information transmission. Rather than a central server, they presented a blockchain design with open records to completely decentralize FL. This was accomplished utilizing the proof-of-work (PoW) agreement calculation, where a brief aggregator is chosen in each circular. Parameters of upgraded models from end-devices are sent to a cluster of mineworkers, who confirm their authenticity through a cross-verification instrument. The PoW agreement calculation allots a target nonce for each circular, with mineworkers compensated for distinguishing the target. This incentivizes support and guarantees an effective learning handle. The worldwide show parameters are amassed utilizing conveyed inexact Newton DANE [8], put away in a piece, and down-

loaded by all machines. Evaluation of worldwide show precision uncovered a noteworthy enhancement with blockchain integration, accomplishing a extend of 0.74–0.82 compared to around 0.7 without blockchain.

Khan et al. [47] proposed a combined learning (FL) approach at the organize edge utilizing the Stackelberg diversion to incentivize device interest. Their framework, planned with a Co-design approach, incorporates a Base Station (BS) speaking to the edge server and client devices with non-IID and heterogeneous information sizes. They utilized the CoCoA system to handle frameworks and measurable heterogeneity. In their approach, IoT gadgets yield reactions to the BS based on advertised remunerate rates, and the BS overhauls the worldwide show in like manner. Execution assessment employing a classification errand appeared that expanding the remunerate rate spurs clients to repeat more, moving forward exactness. In any case, the consider did not address security concerns between edge gadgets and the edge server, clearing out it as future work.

Xu et al.[89] proposed ELFISH, a federated learning system outlined to address the heterogeneity in computation capacity among edge devices, which frequently leads to delays in sharing models due to physical resource limitations. ELFISH utilizes a soft-training strategy to accelerate model preparing by powerfully veiling a select number of neurons at each preparing cycle, considering variables such as computational workload, preparing memory utilization, preparing time utilization, and worldwide merging contribution of each neuron. Moreover, they presented a parameter accumulation conspire to recuperate veiled weights during accumulation, subsequently upgrading accuracy and merging speed for training-on-edge scenarios. To evaluate their system, they utilized Nvidia Jetson Nano advancement sheets to simulate computation-capable and straggler devices with shifting resource limitations. They utilized two CNN models, AlexNet and LeNet, for system testing. Comes about illustrated that ELFISH reliably beats ordinary synchronized/asynchronized FL approaches, accomplishing way better exactness and speedier merging speed. Additionally, indeed with non-IID information, ELFISH shown predominant execution compared to other plans.

Liu et al. [56] proposed a vehicle interruption location framework based on dispersed combined learning (FL) and blockchain innovation to ensure security conservation for vehicles using Differential Security procedures. The framework points to reduce communication overhead and computation costs by dispensing with the global server and enabling secure demonstrate sharing among edge nodes (Roadside units - RSUs) that collect data from vehicles traveling within the same range. Each RSU acts as an aggregator inside its range, conducting FL, storing last models within the blockchain, and competing to confirm accumulation demonstrate preparing exchanges. The winner composes these exchanges into the blockchain employing a conveyed agreement instrument, with show precision utilized for believe assessment. Two sorts of cars are considered: conventional edge vehicles utilized for conveyed edge preparing based on interruption discovery information, and agent vehicles chosen by RSUs to total neighborhood models and send them to the fitting RSU. Communication is established utilizing homomorphic encryption to relieve against pernicious foes. To assess their system's execution, they utilized PyTorch and Syft Python libraries for FL execution, the KDDCup99⁴ dataset, and the Go language to construct the open blockchain. Comes about demonstrate that both exactness

⁴<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

and time costs increment with bigger dataset sizes. Also, expanding the number of collaborated nodes leads to moved forward precision of the worldwide show. Comparing two agreement calculations, POW (proof of work) and POA (proof of accuracy), they found that POA successfully improves the efficiency of POW by diminishing square era time due to expanded believe, hence diminishing mining trouble.

In [68] [88], Federated Learning (FL) has been employed in the realm of COVID-19 diagnosis. Qayyum et al. [68] used a cluster-based FL approach (CFL) to automate the process of COVID-19 diagnosis. Their method, while prioritizing data security, led to a notable 16% enhancement in CFL method performance. Contrary to [68], Zhang et al. [95] introduced a fresh FL technique based on dynamic fusion. This innovative approach determines the involvement of clients based on their local model performance and organizes model fusion according to each client's training timeline, thereby amplifying detection flexibility.

Ni et al. [64] proposed a novel dual filtering mechanism to address Byzantine attacks in Federated Learning (FL) within edge-IoT environments. Their approach aims to enhance the security of the FL training process by identifying and eliminating malicious gradients. Additionally, they developed an adaptive weight adjustment scheme to ensure consistent gradient sizes, thus promoting effective model aggregation. Despite its innovative design, Ni's scheme overlooks a critical aspect: the non-identically and non-independently distributed nature of real datasets used in Edge Computing (EC) environments. This oversight poses challenges to the robustness of existing FL aggregation algorithms and increases the vulnerability of the FL global model to attacks in such scenarios. Consequently, addressing this issue is crucial for ensuring the efficacy and security of FL in real-world EC settings.

2.5 Discussion and Comparison

In this study, we tried to include FL in CPS as well as list as many articles as possible to get a solid picture of its implementation, results, and limitations in CPS. Table 2.1, originally compiled by Mademoiselle Guendouzi, classifies and compares the contributions surveyed regarding FL applications across several CPS domains. We have updated this table in recent years to provide the most current and comprehensive overview. The table considers various criteria, including FL location (Cloud-based, Fog-based, or Edge-based) and architecture (centralized or P2P), aggregation method, aggregation selection case (fixed or dynamic), DL model, dataset reference, and data type (iid or non-iid), the use of TL, and the security method.

Initially, it's important to note that considerable research has been dedicated to implementing FL on the cloud layer to engage a large number of collaborators and enhance long-term learning. Conversely, recent studies seek to boost user involvement by integrating FL into the Fog or Edge layers to meet security and latency demands.

Subsequently, we observed that the majority of contributions [67], [82],[18], [36],[96], [46], [36], [75], [85], [70],[47], and [89] employ a centralized FL architecture, with exceptions being [70],[64] and [56], which utilize a P2P approach in scenarios where FL is deployed on the edge layer without a global or centralized server. Among them, FedAVG emerges as the prevalent aggregation algorithm due to its simplicity in handling homogeneous data

and models. However, [96], [46], and [36] implement the FedPer algorithm to enhance learning, while [70] and [89] devise their own aggregation methods tailored to their specific architecture. Regarding aggregator selection, only [75] and [70] have developed their own algorithms for selecting the aggregator for each FL round. Furthermore, current research endeavors to broaden architectures by incorporating heterogeneous users in terms of data or learning models. Consequently, TL approaches are favored to facilitate knowledge sharing, expedite convergence, and enhance system stability. Lastly, for security purposes, Homomorphic encryption and Differential Privacy are commonly employed for model sharing, whereas blockchain is utilized in P2P architectures.

Contribution	FL Position	Dataset	Model	Architecture	Aggregation	Aggregator	iid Data	TL	Security
Polap et al.[67]	Cloud-based	Skin cancer MNIST	CNN	Centralized	FedAVG	Fixed	Yes	No	Blockchain
Tian et al. [82]	Cloud-based	MNIST CIS-IDS 2017 IoT-23	Denosing Autoencoder	Centralized	FedAVG	Fixed	No	No	No
Chen et al.[18]	Cloud-based	UCI SmartPhone	CNN	Centralized	FedAVG	Fixed	No	Yes	Homomorphic Encryption Differential Privacy BGV Encrption
Hao et al. [36]	Cloud-based	MNIST	CNN	Centralized	FedAVG	Fixed	Yes	No	Homomorphic Encryption
Zhang et al.[96]	Cloud-based	MDI [61] ImageNet ⁵	CNN VGG-11,13,16,19	Centralized	FedPer	Fixed	No	Yes	Homomorphic Encryption
Kevin et al.[46]	Cloud-based	COCO ⁶	CNN	Centralized	FedPer	Fixed	No	Yes	No
Hao et al. [36]	Fog-based	Fasion-MNIST	CNN	Centralized	FedPer	Fixed	No	Yes	Blockchain Differential Privacy
Saha et al.[75]	Fog-based	MNIST	MLP	Centralized	FedAVG	Dynamic	No	No	No
Wang et al. [85]	Fog-based	MNIST	CNN	Centralized	FedAVG	Fixed	No	Yes	No
Yao and Ansari [91]	Fog-based	MNIST	CNN	Centralized	FedAVG	Fixed	No	No	No
Qu et al. [70]	Edge-based	CIFAR-10	CNN	P2P	DANE	Dynamic	No	No	Blockchain
Khan et al. [47]	Edge-based	MNIST	Multinomial logistic Regression	Centralized	FedAVG	Fixed	Yes	No	No
Xu et al.[89]	Edge-based	MNIST CiFAR-10	LENET AlexNet	Centralized	ELFISH	Fixed	Yes	Yes	No
Liu et al. [56]	Edge-based	KDDCup99	MLP	P2P	FedAVG	Fixed	Yes	No	Blockchain Differential Privacy
Qayyum et al. [68]	Edge-based	Covid_19 dataset	CNN	P2P	FedAVG	Dynamic	No	Yes	Blockchain Homomorphic Encryption
Ni et al. [64]	Edge-based	Fasion-MNIST MNIST CIFAR-10	CNN	P2P	FedAVG	Dynamic	No	Yes	Homomorphic Encryption Blockchain

Mlle. guendouzi Badra Souhila [100]	Cloud-based Edge-based Fog-based	Fasion-MNIST	CNN	P2P	FedGA FedPer FedAVG	Dynamic	No	Yes	Homomorphic Encryption Blockchain
--	--	--------------	-----	-----	---------------------------	---------	----	-----	---

Table 2.1: Comparison of the State of The Art.

2.6 Conclusion

In this chapter, we have looked at different FL architectures suggested for various areas of CPS. It has been shown that each proposal uses machine learning techniques in unique ways to meet specific needs, which affects their architectures. However, despite these differences, none of the architectures completely overcome all the challenges related to FL, mainly because of the complex nature of FL research.

Chapter 3

FedGA-ICPS: Federating Learning through Genetic Algorithms for ICPS

Industrial cyber-physical systems (ICPS) have attracted significant interest over the past decade and are expected to continue growing in the coming years, driven by technological advances and the emergence of new heterogeneous environments. As a result, there is growing interest in using machine learning (ML) techniques to unify decision-making processes in these ICPSs, which remains an area under research and development. Federated learning (FL) is a machine learning technique that facilitates collaboration between ICPS to improve decision making and accelerate the training of deep learning models through transfer learning.

We observe that the implementation of ML techniques, including FL, Deep Learning (DL) and Transfer Learning (TL), in ICPS varies across different research works, depending on the factors such as system requirements, network architecture, environmental conditions, security considerations, and capabilities of the entities involved, such as cloud, fog, and edge computing resources. However, despite the progress that has been made, no single solution can address all the challenges associated with FL due to the vast and evolving nature of this research field. Integrating a correct solution for a specific application in every situation remains a complex task. In this chapter, we will start by defining and building the architecture of our proposed FedGA-ICPS framework. We will then improve its performance through performing various tests.

3.1 Design Architecture

Considering the fact that the goal of our project is to deploy machine learning strategies (namely FL, DL and TL) at each of the three levels of the system (Edge, Fog and Cloud). Decentralized and centralized, with distributed learning across four system levels, is the design we presented. The IoT layer, represents a set of different industrial sensors and actuators. The Edge layer, is made up of different industrial grades. The Fog layer, consisting of a cluster of Fog nodes and Cloud layer, replaces ICPS servers globally. Figure

3.1 presents our developed architecture for the FedGA-ICPS framework. Details of the layers are explained below.

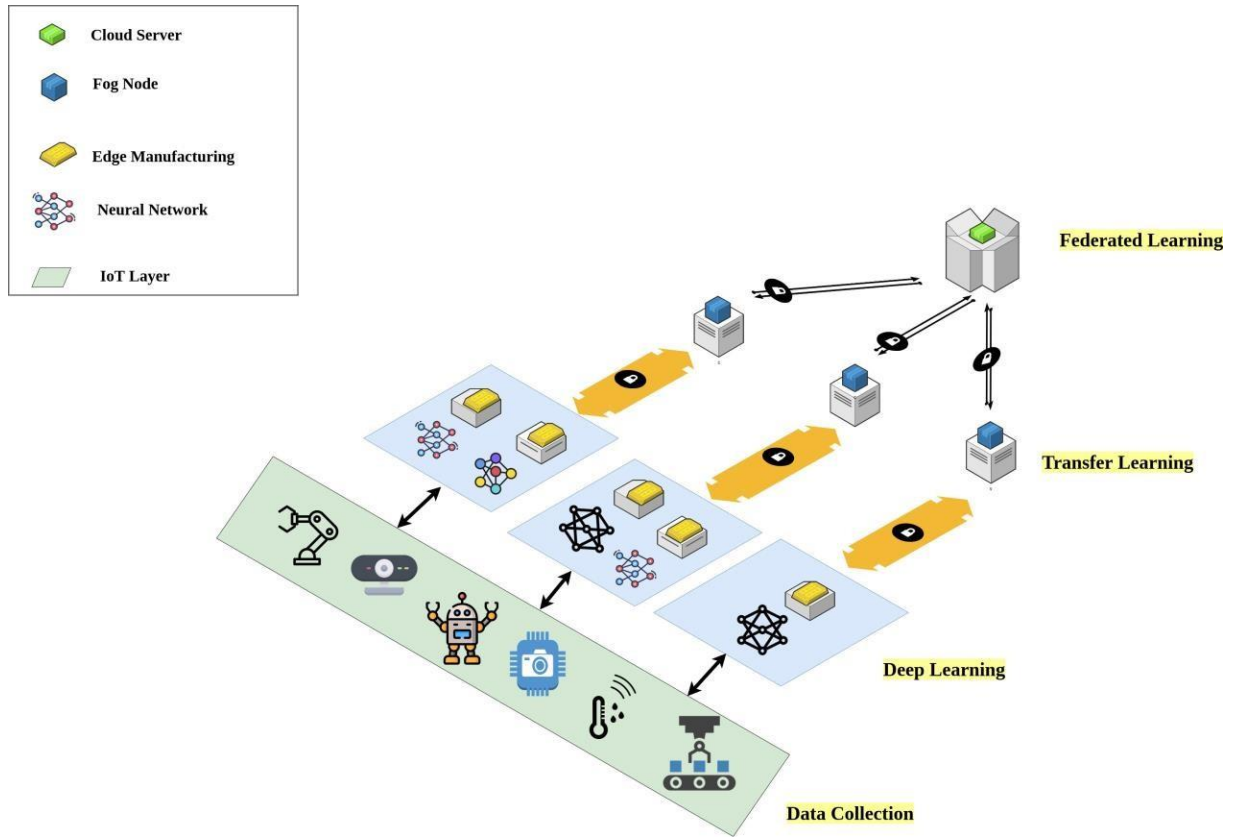


Figure 3.1: The Proposed Architecture Design for **FedGA-ICPS** framework.

3.1.1 IoT Layer

In the industrial setting, this layer encompasses all IoT components found within factories. Here, sensors and actuators are deployed for various purposes across different layers. Sensors, including robots, surveillance cameras, and a range of heterogeneous sensors (such as those for humidity, temperature, movement, and gas detection), gather data tailored to the factory's domain. For example, in an automobile assembly plant, robots collect data on defective spare parts. This data may or may not be relevant depending on the context. Subsequently, once data collection is complete, it is transmitted to the Edge layer, specifically to the edge server within the manufacturing facility, for further processing before being utilized in executing **FedGA-ICPS**.

During the execution of our framework, directives are issued to actuators either after or during the decision-making process to carry out domain-specific actions.

3.1.2 Edge Layer

The Edge layer or manufacturing layer serves as a consolidation of various industrial zones located geographically worldwide, spanning across countries and cities. Within each zone, there exists a cluster of factories exhibiting diversity in their application domains, computational resources, and crucially, learning capabilities, which directly impact their decision-making processes.

Every industrial factory maintains a dataset housing its proprietary information, which expands as IoT sensors continuously gather data. This data undergoes processing through data mining techniques to refine and extract pertinent information. The evolving dataset is utilized for local training of neural models using specific deep learning algorithms tailored to the factory's domain application. Consequently, the factory is equipped to make predictions autonomously without the need to transmit data to a central server.

3.1.3 Fog Layer

After the consolidation of various manufacturing facilities into separate industrial zones worldwide, the establishment of a node to represent the latter becomes imperative. Hence, the Fog layer embodies a cluster of nodes functioning as fog servers. Each node serves as the overseer for a specific zone and is strategically positioned in close proximity to its designated area.

The primary duties of each node encompass the collection and storage of local personalized models, ensuring transfer learning by facilitating the movement of these models between edge nodes, uploading the models to the aggregator, and finally, receiving the aggregated model and disseminating it to the corresponding zone.

3.1.4 Cloud Layer

The cloud layer, often referred to as the management layer, serves as the overseer of the entire system. It encompasses the global server, responsible for initially providing the default aggregator. This aggregator gathers all the local neural models and facilitates federated learning to generate a global model. This process involves executing a carefully selected aggregation algorithm, as elaborated in Section 3.2.4, tailored to different environmental contexts. Subsequently, the global model is broadcasted across the network participants.

3.2 FedGA-ICPS Framework

As depicted in Figure 3.2 and according to Guendouzi et al. [33], FedGA-ICPS framework comprises five stages: CPS (red rectangle), Learning (blue rectangle), Election (green rectangle), Aggregation (yellow rectangle), and Broadcasting (violet rectangle). In the following, we will elaborate on each component and step of the FedGA-ICPS FedGA-ICPSframework.

1. Initially, FedGA-ICPS establishes and implements an ICPS, which is composed of entities and components with diverse forms and characteristics. Each entity possesses its own structure and behavior, enabling communication and interaction within various environments.
2. Then, through simulation and run-time execution, FedGA-ICPS gathers, formats, cleans, and normalizes streaming data that is generated and communicated between different ICPS components. This data is utilized for the learning step, which relies on a neural network model assigned to each respective device.
3. Consequently, FedGA-ICPS proposes a set of components to elect the best candidate for federating the learning between the embedded local models. The election process considers various parameters, including processing and memory capacities, latency, availability, security, etc.
4. After a local convergence learning, FedGA-ICPS conducts aggregation using genetic algorithms. These algorithms consider the weights of the local models. Subsequently, in the selected component, the optimal weight vector for broadcasting is generated.
5. Finally, FedGA-ICPS broadcast to the different clients, edges, and components the resulting optimal weights through TL.

The following section details each step of FedGA-ICPS.

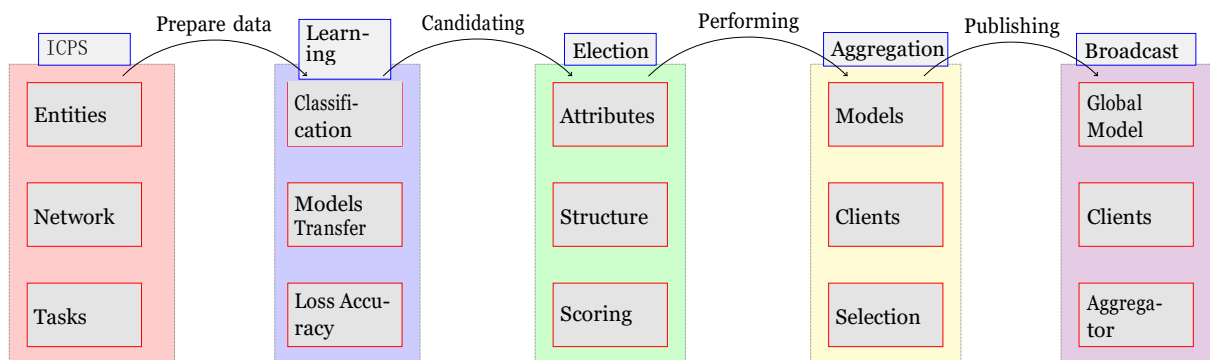


Figure 3.2: The Interoperability and Integrity Validation and Evaluation [33]

3.2.1 Learning

The goal of this phase is to create a local model for each edge entity. It includes two subprocesses. Here are the details:

Deep Learning

During the learning phase, individual edge nodes undergo training processes for their neural network models using locally sourced datasets. These datasets and training methodologies vary across industries, reflecting the unique needs of the factories in which they operate. Upon achieving high model accuracy post-training, the edge nodes transmit the updated parameters, including the model itself and its precision, to the nearest fog node. Subsequently, the fog node facilitates the dissemination of these parameters to the Cloud server, facilitating the update of the repository housing efficient models. Conversely, if the model accuracy falls short of expectations post-training, the edge node initiates a request for a learning transfer, seeking assistance to improve its model performance.

Transfer Learning

This methodology has been integrated into our framework, serving two distinct purposes. Firstly, in cases where the accuracy of the local model diminishes, the fog node initiates a request to the Cloud server for a high-performance model relevant to the same application domain and task as the edge node. The Cloud server consults the structured repository, as depicted in Figure 3.4, to identify the suitable model, which is then transferred to the designated edge node via the fog node. Secondly, the framework facilitates model sharing for Federated Learning (FL), a concept elaborated upon in Section 3.2.5. FL involves the collaborative training of models across distributed edge nodes.

Registry				
IpAddress_Edge	IpAddress_Fog	Domain	Task	Accuracy
192.168.10.25	192.168.10.1	Clothing	Image Classification	99.87%
192.168.1.8	192.168.1.1	Car Assembly	Object detection	99.61%

Figure 3.4: TL Registry Structure.

Both phases of deep and transfer learning are represented by the sequence diagram shown in Figure 3.5.

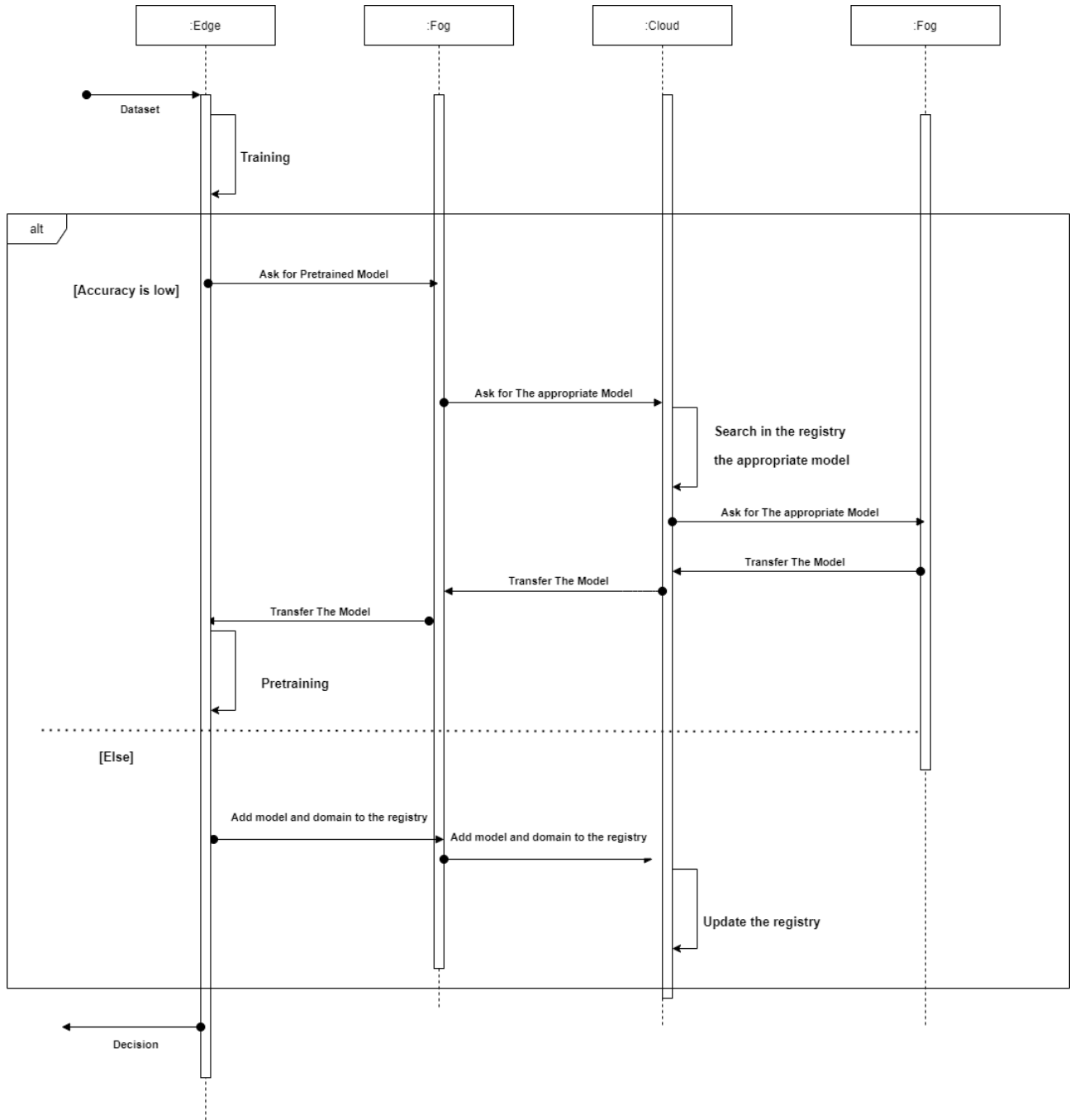


Figure 3.5: Learning Sequence Diagram.

3.2.2 Election

In the process of selecting the suitable candidate for federated learning, FedGA-ICPS assesses the capability of various components within the system, such as Fog or Cloud nodes. By default, the computing cloud server is designated as the primary aggregator. However, depending on factors like available memory and processing capacity, the cloud server may designate another component as a secondary aggregator. FedGA-ICPS organizes all S components, denoted as S, into a prioritized list of aggregators based on their capacities.

Chapter 3. FedGA-ICPS Framework

Guendouzi Badra Souhila. [100], have formulated the election process by an election tree ET, which is characterized by the tuple $\langle E, C, P \rangle$, where:

- E are the entities that are detailed in Section 3.2.1.
- C returns a value that takes into consideration an entity attributes as its actual available capacity of memory, processing capacity, etc.
- P assigns a priority value to a given entity ϵ to resolve the nondeterministic problem when entities capacities are equals.

In the context of FedGA-ICPS, periodic capacity assessments of each node are integral, as they can influence the election order. Illustrated in Figure 3.6, the cloud server ϵ_1 assumes the position of the root of the tree, signifying its status as the most potent entity at time $t = 0$. Conversely, children nodes represent the less efficient entities, while leaves typically often IoT devices, regarded as the weakest within the system. The priority functions of each entity, denoted as P_i , are depicted by the arcs within the tree structure.

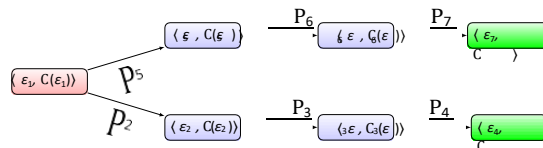


Figure 3.6: A Configuration of the Election Process at a slot of time.

The summary of the election phase is represented by the sequence diagram shown in Figure 3.7.

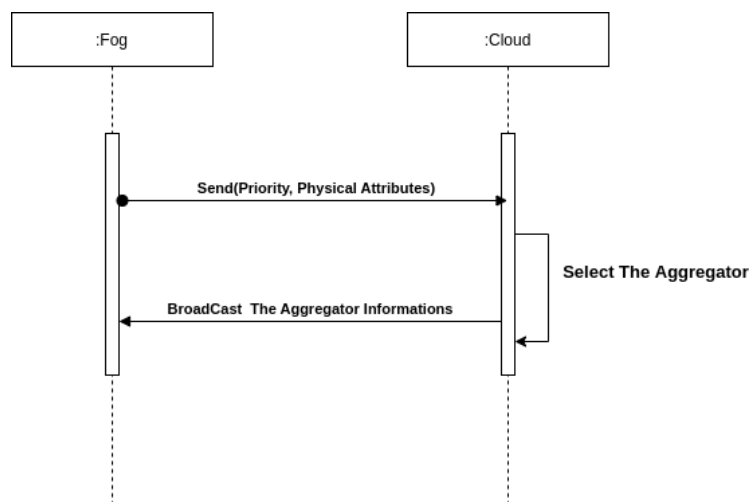


Figure 3.7: Election Sequence Diagram.

3.2.3 Federation

In contrast to FedAvg [81] and FedPer [9], the FedGA technique developed within FedGA-ICPS operates differently. In FedGA, all edge nodes solely transmit the base layer weights to the selected aggregator. Subsequently, the aggregator computes the updated weights by invoking the genetic algorithm (as illustrated in Figure 3.8). The genetic algorithm operates as follows: a weight vector serves as a representation of the system across the chromosomes.

1. Define an adequate *chromosome* which the weight vectors.
2. Select a large set of chromosomes, *population* takes into account all weight vectors collected from the different components.
3. Apply the reproduction operators (*selection, crossover, mutation*). *selection* is applied on vectors with high ranking (fitness evaluation). In our case, the fitness is the loss function. The *crossover* operation is based on the single point paradigm. It means that a vector is divided into two parts to be exchanged with another vector to form a new population. Finally, the *mutation* operation collects randomly only 10% of weights to reproduce new vectors.
4. FedGA-ICPS repeats this process until all edge nodes models converge with a maximal accuracy.

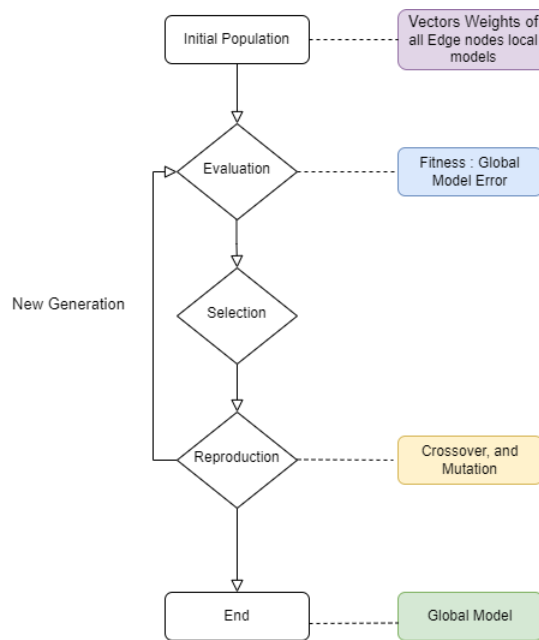


Figure 3.8: FedGA Schema.

FL process is summarized in Figure 3.9.

Algorithm 2 Federated Genetic Algorithm (FedGA) [32]

The \mathbf{K} clients are indexed by \mathbf{k} ; \mathbf{B} is the local mini-batch size, D_k is the dataset available to client \mathbf{k} , D_t is the dataset used for the test which is available on the aggregator, $W_{\mathbf{B}}$ is the vector of base layers, $W_{\mathbf{P}}$ is the vector of personalized layers, \mathbf{E} is the number of local epochs, and η is the learning rate.

```

1: Procedure FedGA d Run on the server.
2: Initialize  $W_{\mathbf{B}}^0$ ;
3: for each round  $\mathbf{t} = 1, 2, 3, \dots$  do
4:   for each client  $\mathbf{k} \in \mathbf{K}$  do
5:      $W_{\mathbf{B}, \mathbf{k}}^{t+1} \leftarrow \text{ClientUpdate}(\mathbf{k}, W_{\mathbf{B}, \mathbf{k}}^t, D_k, \eta);$  d In Parallel.
6:   end for

```

Chapter 3. FedGA-ICPS Framework

```

7:    $W_{\mathbf{B}}^{t+1} = GA(D_t, W_{\mathbf{B}}^t);$            d Only base layers are aggregated
8: end for
9: End procedure FedGA
10: Procedure ClientUpdate ( $k, w_{\mathbf{B}}^t$ )           d Run on client k.
11:  $\beta \leftarrow$  (Split  $D_k$  into mini-batches of size  $B$ ;)
12: for each local epoch i from 1 to E do
13:   for batch  $\mathbf{b} \in \beta$  do
14:      $w_{\mathbf{B}}, w_{\mathbf{P}} \leftarrow w - \eta \Delta L(w_{\mathbf{B}}, w_{\mathbf{P}}, b);$    d base layers are updated and trained and
     personalized layers are trained
15:   end for
16: end for
17: return t to the Server
18: End procedure ClientUpdate ( $k, w_{\mathbf{B}}$ )

```

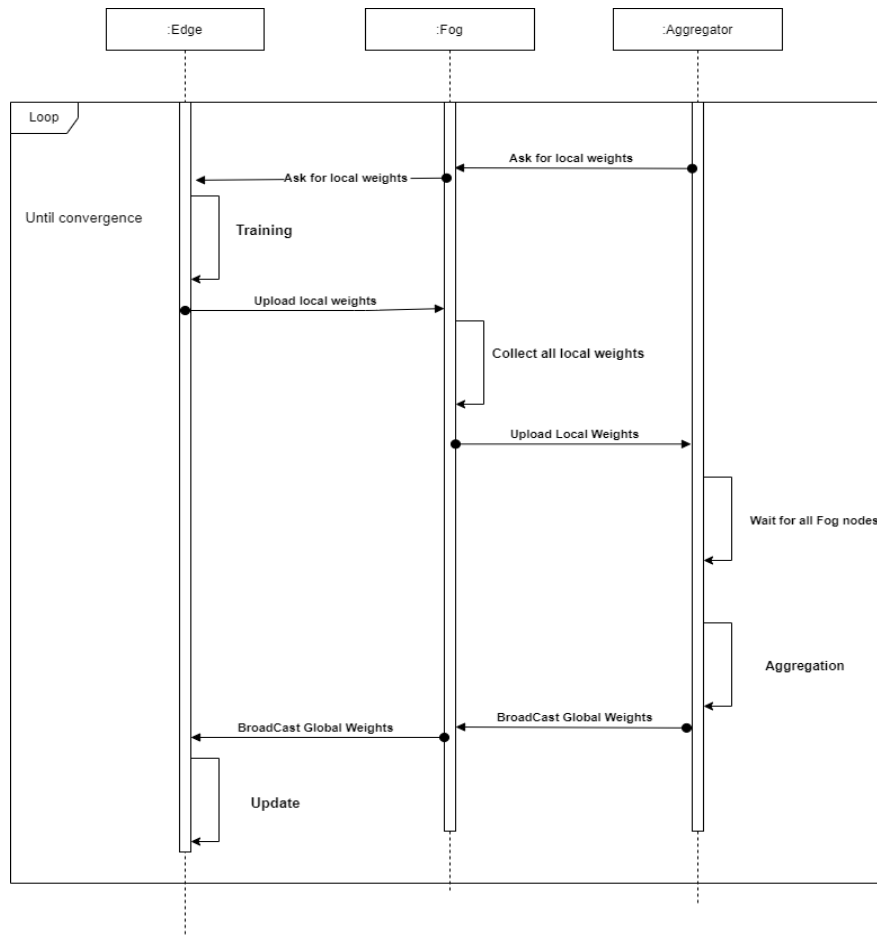


Figure 3.9: FL Sequence Diagram

3.2.4 Broadcasting

Following the aggregation phase, FedGA-ICPS disseminates the weights of the model's base layers to the involved components during the learning phase. Consequently, a new model characterized by high accuracy and minimal losses is crafted during the training phase. The broadcast phase from the aggregator to fog nodes and subsequently to edge

nodes employs the TL technique, which selectively transfers personalized layers.

3.3 Conclusion

In this chapter, the motivation for the proposed FedGA-ICPS architecture and framework is defined by Guendouzi Badra Souhila. [100], the architecture is then explained in detail, with a description of its four layers. Furthermore, a comprehensive description of the framework is provided, detailing the steps that define the FedGA-ICPS implementation and illustrating them using various UML diagrams. In the next chapter, the effectiveness of the FedGA-ICPS framework will be demonstrated through the presentation of its application .

Chapter 4

FedGA Application

In the work already done by Mlle. Guendouzi Badra Souhila. [100], (<https://github.com/SouhilaGuendouzi/SoftwareImpacts-FedGA-ICPS>), which utilized Python along with PyTorch, NumPy, Matplotlib, PyGAD, Socket, Threading, Pandas, among others that we have mastered well, we tested this pre-existing work initially performed with GPU by transitioning it to CPU. Additionally, we integrated datasets such as Fashion MNIST, EMNIST, MNIST, and CIFAR-10 into this work and made topology changes including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs), Artificial Neural Networks (ANNs), and DenseNet. Furthermore, we conducted tests with two optimization algorithms, Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), using the EMNIST and MNIST datasets.

4.1.1 Dataset Fashion MNIST

The Fashion MNIST dataset (short for Fashion Modified National Institute of Standards and Technology) consists of 70,000 small square images measuring 28×28 pixels. These images depict Zalando’s clothing items³ and are categorized into 10 classes numbered from 0 to 9. Since its introduction in 2017, this dataset has been widely used for evaluating classification algorithms. Figure 4.2 illustrates a sample of images from this dataset.

For our experiments, we employed this dataset in two scenarios: iid dataset; where it is partitioned into homogeneous subdatasets at the edge nodes, ensuring they have the same size and distribution of samples, and non-iid dataset; where it is partitioned into heterogeneous subdatasets, resulting in unequal sizes and distributions of samples across nodes.



Figure 4.2: Taken from the Fashion MNIST dataset.

4.1.2 Dataset EMNIST

³<https://github.com/zalandoresearch/fashion-mnist>

EMNIST (Extended Modified National Institute of Standards and Technology) is an extension of the MNIST dataset commonly used in machine learning and computer vision errands. It comprises of written by hand character pictures comparable to MNIST but incorporates capitalized and lowercase letters as well as digits. The EMNIST dataset contains 814,255 pictures part into preparing and testing sets. Each picture is 28x28 pixels, making it appropriate for preparing and assessing models for assignments like character acknowledgment and classification. Figure 4.3 illustrates a sample of images from this dataset.



Figure 4.3: Taken from EMNIST dataset.

4.1.3 Dataset MNIST

The MNIST dataset could be a collection of 70,000 grayscale pictures of manually written digits from zero to nine. Each picture is 28x28 pixels in estimate and is labeled with the comparing digit it speaks to. The dataset is commonly used in machine learning and computer vision investigate as a benchmark for assessing calculations in assignments such as digit classification and picture acknowledgment. The MNIST dataset is partitioned into 60,000 preparing pictures and 10,000 testing pictures, making it a standard dataset for preparing and assessing machine learning models. Figure 4.4 illustrates a sample of image from this dataset.



Figure 4.4: Taken from MNIST dataset.

4.1.4 Dataset Cifar-10

The CIFAR-10 dataset consists of 60,000 colour images in 10 categories, with each category containing 6,000 images. These categories include objects such as aircraft, cars, feathered creatures, cats, deer, frogs, horses, ships, and trucks.

Each image measures 32 x 32 pixels and is widely used to create and test machine learning and computer vision models, especially for tasks such as image classification and protest identification. Figure 4.5 illustrates a sample of image from this dataset.



Figure 4.5: Taken from Cifar-10 dataset.

4.1.5 Neural Network Models

In order to test our classification with five distinct neural organisation models described in gives the first chapter in

section 1.3.7 , we present the code corresponding to each demonstration in figures 4.6,4.7,4.8,4.9,and 4.10. These areas detail the specific implementation of each neural organisation, including parameters, layers, execution capabilities and optimisers. This approach will allow us to take an in-depth look at how each demonstration works and compare its implementation on the modified datasets we have chosen. By examining the code of each demonstration, we will also be able to understand the differences and similarities between the CNN, ANN, RNN, GRU and DenseNet approaches in terms of design and operation.

```
def __init__(self, args):
    super(ClientModel, self).__init__()
    self.conv1 = nn.Conv2d( in_channels: 3, out_channels: 6, kernel_size: 5)
    self.conv2 = nn.Conv2d( in_channels: 6, out_channels: 16, kernel_size: 5)
    self.conv3 = nn.Conv2d( in_channels: 16, out_channels: 32, kernel_size: 5)

    self.pool = nn.MaxPool2d( kernel_size: 2, stride: 2)
    self.fc1 = nn.Linear(32 * 5 * 5, out_features: 120)
    self.fc2 = nn.Linear( in_features: 120, out_features: 84)
    self.fc3 = nn.Linear( in_features: 84, args.num_classes)

def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = self.pool(F.relu(self.conv3(x)))
    x = x.view(-1, 32 * 5 * 5)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

Figure 4.6: CNN model.

```
def __init__(self, args):
    super(ClientModel, self).__init__()
    self.fc1 = nn.Linear(3 * 32 * 32, out_features: 120)
    self.fc2 = nn.Linear( in_features: 120, out_features: 84)
    self.fc3 = nn.Linear( in_features: 84, args.num_classes)

def forward(self, x):
    x = x.view(-1, 3 * 32 * 32)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

Figure 4.7: ANN model.

4.1.6 Transfer Learning

To ensure Transfer Learning (TL) between Edge entities, it is essential to define the application domain and tasks of each entity. In our experiments, we assume that all

```

def __init__(self, args):
    super(ClientModel, self).__init__()
    self.embedding = nn.Linear(3 * 32 * 32, out_features=120)
    self.rnn = nn.RNN(input_size=120, hidden_size=84, num_layers=1, batch_first=True)
    self.fc = nn.Linear(in_features=84, args.num_classes)

def forward(self, x):
    x = x.view(-1, 3 * 32 * 32)
    x = torch.relu(self.embedding(x))
    x, _ = self.rnn(x.unsqueeze(1))
    x = self.fc(x.squeeze(1))
    return x

```

Figure 4.8: RNN model.

```

def __init__(self, args):
    super(ClientModel, self).__init__()
    self.embedding = nn.Linear(3 * 32 * 32, out_features=120)
    self.gru = nn.GRU(input_size=120, hidden_size=84, num_layers=1, batch_first=True)
    self.fc = nn.Linear(in_features=84, args.num_classes)

def forward(self, x):
    x = x.view(-1, 3 * 32 * 32)
    x = torch.relu(self.embedding(x))
    x, _ = self.gru(x.unsqueeze(1))
    x = self.fc(x.squeeze(1))
    return x

```

Figure 4.9: GRU model.

entities share the same task, which is image classification, but have distinct domains. We have partitioned our benchmark dataset into non-iid subdatasets among them, with the exception of Edge ε_5 , which shares the same domain and task as Edge ε_3 for performance evaluation purposes. Table 4.1 outlines our TL configuration for all five Edge nodes.

	Domain	Task
Edge ε_1	A	Image Classification
Edge ε_2	B	Image Classification
Edge ε_3	C	Image Classification
Edge ε_4	D	Image Classification
Edge ε_5	C	Image Classification

Table 4.1: Application Domains and Tasks.

4.1.7 Election

In the first round of FL, the Cloud node selects an aggregator, which can be itself or another fog node. This selection is based on their capacity, which is initially a dynamic

```
def __init__(self, args):
    super(ClientModel, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5)
    self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
    self.conv3 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5)
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    self.fc1 = nn.Linear(32 * 5 * 5, out_features=120)
    self.fc2 = nn.Linear(in_features=120, out_features=84)
    self.fc3 = nn.Linear(in_features=84, args.num_classes)

    # Utilisation de DenseBlock de DNN dancenet
    self.denseblock1 = DenseBlock(in_channels=6, out_channels=32, growth_rate=8, num_layers=4)
    self.denseblock2 = DenseBlock(in_channels=16, out_channels=64, growth_rate=8, num_layers=4)
    self.denseblock3 = DenseBlock(in_channels=32, out_channels=128, growth_rate=8, num_layers=4)

def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.pool(self.denseblock1(x))
    x = F.relu(self.conv2(x))
    x = self.pool(self.denseblock2(x))
    x = F.relu(self.conv3(x))
    x = self.pool(self.denseblock3(x))
    x = x.view(-1, 32 * 5 * 5)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

Figure 4.10: densenet model.

random integer, and their priority (see section 3.2.3 and Figure 3.7). The other fog nodes then upload their local models to the chosen aggregator instead of the default unique aggregator, which is the Cloud node. The capacity and priority attributes are initialized using the **random** Python module, as shown in Figure 4.11.

```
self.capacity=99
self.priority = 99
self.capacity=int(random.uniform(a: 0, b: 100))
self.priority=args.priority
```

Figure 4.11: Election attributes..

4.1.8 Federated With Genetic Algorithm (FedGA)

To start, we characterised a global show that will be tested using a fraction of the dataset from each Edge node. After collecting all local personalized models, we created our initial population, which consists of a set of individuals representing weight vectors. Each vector includes all weights of a personalized model. As illustrated in Figure 4.12, we selected 5 GA generations and 3 solutions to be chosen as parents in the mating pool. The parents

were selected based on their rank using the Fitness evaluation function.

```
def FedGA(initial_population, model, dataset):
    global data_loader
    data_loader = DataLoader(dataset, batch_size=32, shuffle=True)
    num_generations = 5
    num_parents_mating = 3
    parent_selection_type = "rank"
    crossover_type = "single_point"
    mutation_type = "random"
    mutation_percent_genes = 10
    keep_parents = 1
    ga_instance = pygad.GA(num_generations=num_generations,
                           num_parents_mating=num_parents_mating,
                           initial_population=initial_population,
                           fitness_func=fitness,
                           parent_selection_type=parent_selection_type,
                           crossover_type=crossover_type,
                           mutation_type=mutation_type,
                           mutation_percent_genes=mutation_percent_genes,
                           keep_parents=keep_parents,
                           on_generation=callback_generation)

    ga_instance.run()
```

Figure 4.12: FedGA Configuration.

We selected the fitness function, illustrated in Figure 4.13, to be the global model error, determined by testing on a fraction of data from each Edge node.

```
def fitness(solution, sol_idx):
    model_weights_dict = convert_solution_to_model_weights(solution, model)
    model.load_state_dict(model_weights_dict)
    total_loss = 0.0
    with torch.no_grad():
        for idx, (data, target) in enumerate(dataset):
            prediction = model(data)
            loss = loss_function(prediction, target)
            total_loss += loss.item()
    average_loss = total_loss / len(dataset)
    fitness_score = 1.0 / (average_loss + 1e-9)
    return fitness_score
```

Figure 4.13: FedGA Fitness Function.

4 .2.9 Optimization Algorithms

4.2.9.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a search method that uses a set of agents moving through the search space to find the global minimum of an objective function. Each particle's trajectory is determined by a simple rule that incorporates its current velocity and the exploration histories of itself and its neighbors.

In this table, we examine the EMNIST dataset using two optimization techniques: the Standard Genetic Algorithm and Particle Swarm Optimization. We analyze how each method generates and interprets graph representations of the dataset. Finally, we discuss

Chapter 4. FedGA-ICPS Implementation and Experiments

and compare their performance, strengths, and potential applications, providing insights into the effectiveness of each approach in data analysis and machine learning.

Graphs of the EMNIST dataset with the standard genetic algorithm	Graphs of the EMNIST dataset with the Particle Swarm Optimization	Discussion and comparison
		<p>We noticed that the FedAVG graph with Particle Swarm Optimization achieves values 2% better in training and 10% better in testing than the first one with the standard genetic algorithm</p>
		<p>We noticed that the FedGA graph with Particle Swarm Optimization achieves values 2% better in training and 15% better in testing than the first one with the standard genetic algorithm</p>
		<p>We noticed that the FedPer graph with Particle Swarm Optimization achieves values 1.75% better in training and 10% simulated testing with the standard genetic algorithm.</p>

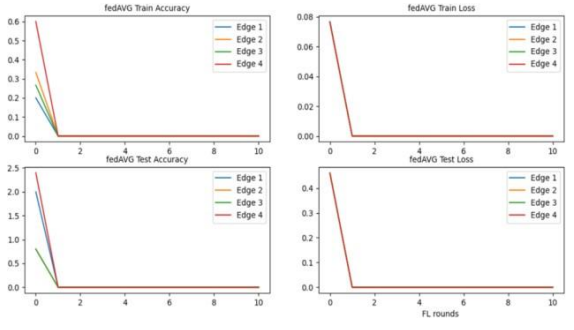
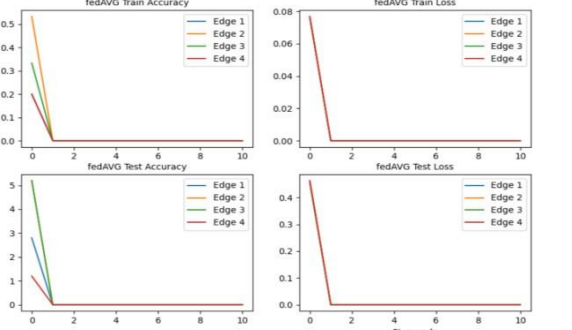
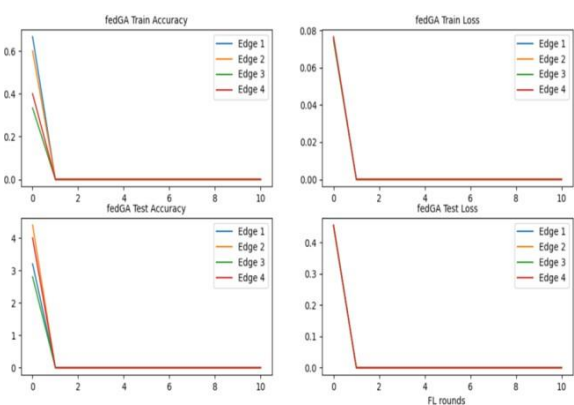
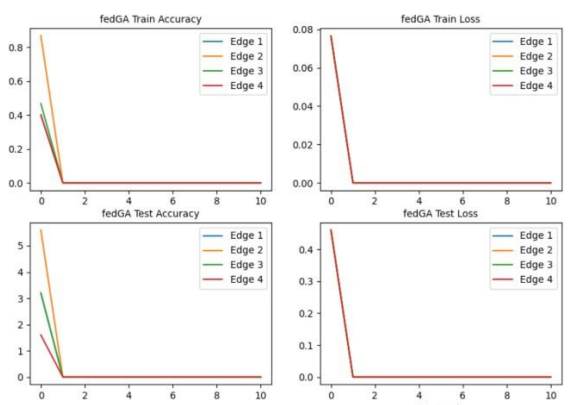
Table 4.2: Comparing graph representations between the Standard Genetic Algorithm and Particle Swarm Optimization

Chapter 4. FedGA-ICPS Implementation and Experiments

4.2.9.2 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a population-based metaheuristic designed to find approximate solutions to difficult optimization problems. In ACO, a group of software agents known as artificial ants search for good solutions to a given problem. This method transforms the optimization problem into finding the best path on a weighted graph. The ants incrementally build solutions by traversing the graph. The construction of these solutions is stochastic and guided by a pheromone model,[101] which is a set of parameters associated with the graph's components (nodes or edges) that are updated in real-time by the ants.

In this table, we examine the MNIST dataset using two optimization techniques: the Standard Genetic Algorithm and Ant Colony Optimization. We analyze how each method generates and interprets graph representations of the dataset. Finally, we discuss and compare their performance, strengths, and potential applications, providing insights into the effectiveness of each approach in data analysis and machine learning.

Graphs of the MNIST dataset with the standard genetic algorithm	Graphs of the MNIST dataset with the Ant Colony Optimization	Discussion and comparison
		<p>We noticed that the FedAVG graph with Ant Colony Optimization achieves values 0.5% simulated training with the standard genetic algorithm and 5% better in testing than the first one with the standard genetic algorithm</p>
		<p>We noticed that the FedGA graph with Ant Colony Optimization achieves values 0.8% better in training and 5% better in testing than the first one with the standard genetic algorithm</p>

Chapter 4. FedGA-ICPS Implementation and Experiments

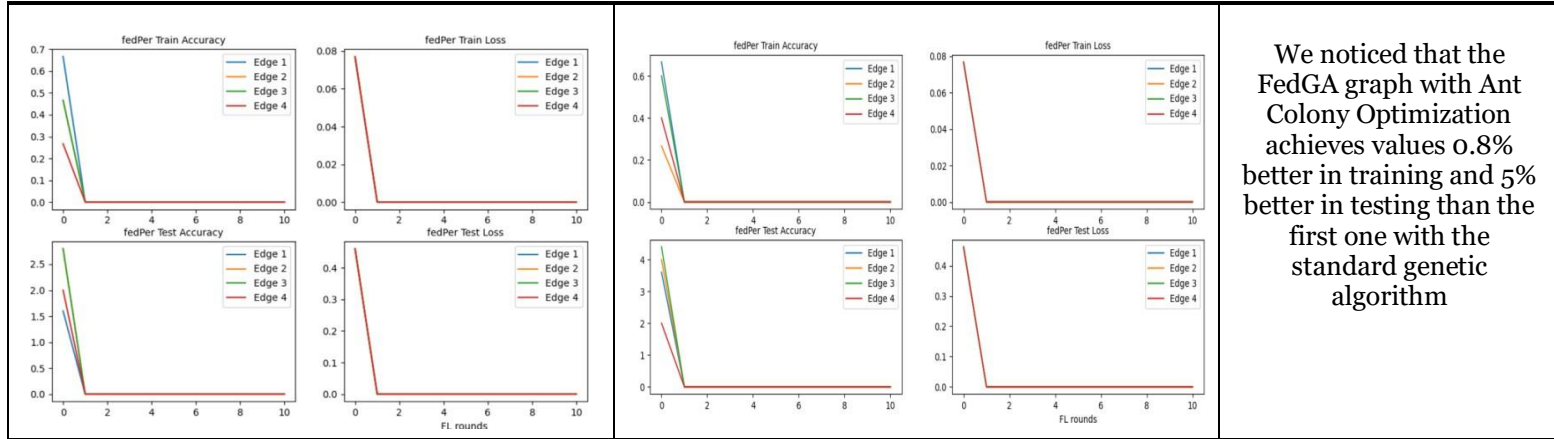


Table 4.3: Comparing graph representations between the Standard Genetic Algorithm and Ant Colony Optimization.

We noticed that the FedGA graph with Ant Colony Optimization achieves values 0.8% better in training and 5% better in testing than the first one with the standard genetic algorithm

4.2 Experiments and Results

The experiments were conducted on a framework named DESKTOP-FO7DGKA, equipped with an Intel(R) Core(TM) i5-10310U CPU @ 1.70GHz with a frequency of 2.21 GHz, and 8.00 G of RAM. The system runs a 64-bit operating system with an x64 processors. The nodes are locally based, and their communication is established using sockets and threads. The selection of models and datasets is detailed in section 4.2.5 and section 4.2.4, respectively.

4.2.1 FL Evaluation

We evaluated the FL method using our **FedGA** aggregation method and monitored training and test accuracies for all four datasets with all five models. The results of these scenarios are presented as follows:

Fashion MNIST dataset with Recurrent Neural Network (RNN) model

In this first scenario, we study the evolution of the four edges of the Fashion MNIST dataset using a recurrent neural network (RNN) model with aggregation algorithms.

- **FedAVG**

By applying FedAVG as an aggregation algorithm in this case, the outcomes displayed in Figure 4.14 show low training and testing accuracies for all the local models, with a low value of 4%. This is mostly due to the use of a single, ineffective model across all edge nodes, as well as the insignificance of the data as long as they all have the same amount of data and the same distribution.

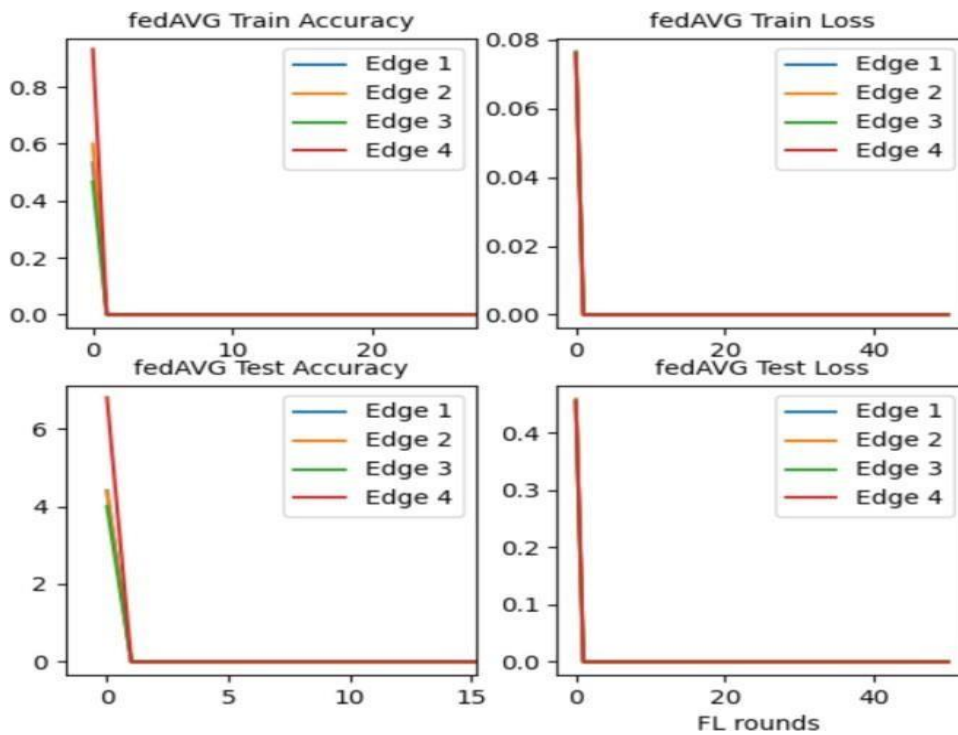


Figure 4.14: FedAVG(1).

- **FedPer**

With FedPer, local models are divided into base layers and custom layers. The

layers analyzed by the individual are not communicated to the server, and only the base layers are aggregated using TL. The results presented in Figure 4.15 show convergence up to the third round for all local edge node models of up to 5.9% in the training phase and 4% in the test phase, confirming an improvement in training accuracy over FedAVG.

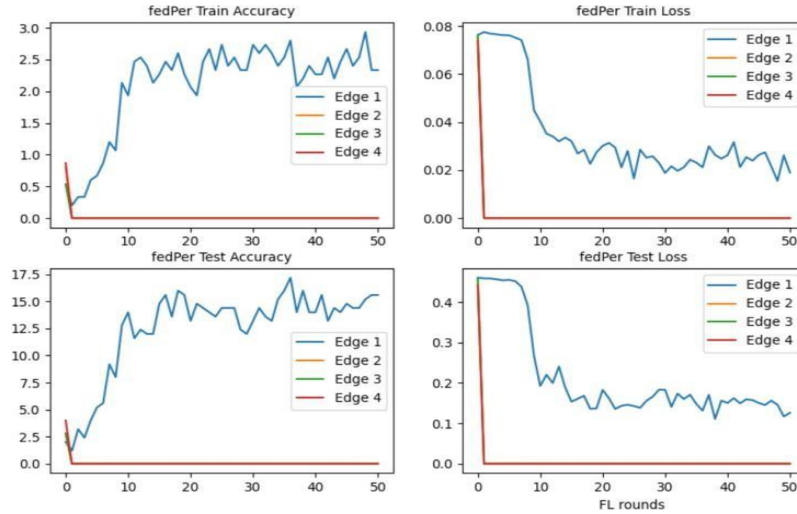


Figure 4.15: FedPer(1).

- **FedGA**

The results of using the **FedGA** algorithm are shown in Figure 4.16, where all local models diverge with a training accuracy of 0.44% and a testing accuracy of 3.6% for all edge nodes.

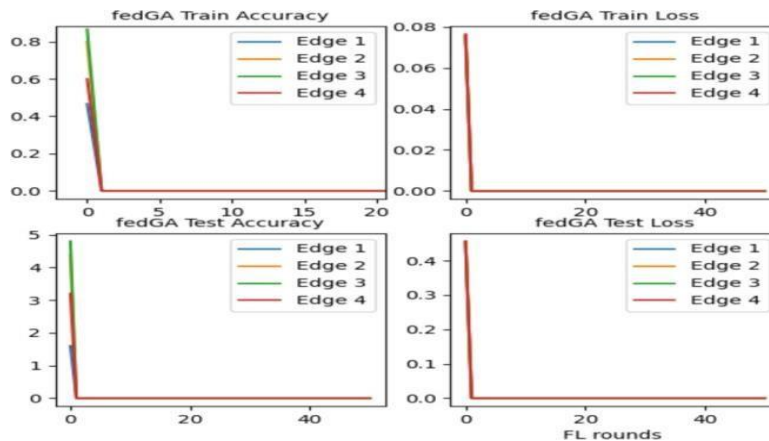


Figure 4.16: FedGA(1).

EMNIST dataset with Artificial Neural Network (ANN) model

For this second scenario, we consider that we have tested the development of the four edges with the EMNIST dataset and the Artificial Neural Network (ANN) model.

• **FedAVG**

By applying FedAVG as an aggregation algorithm with the EMNIST dataset with (ANN) model , the results displayed in Figure 4.17 show a value of 2.5% in the training phase and 10% in the testing phase.

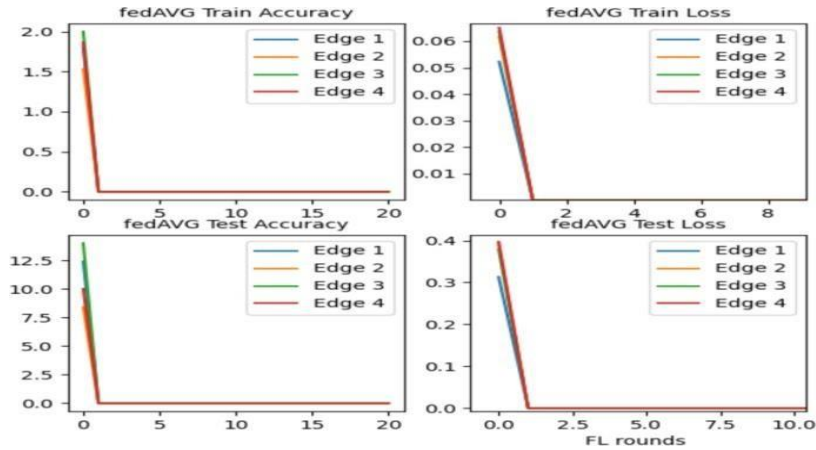


Figure 4.17: FedAVG(2).

• **FedGA**

The results of using the FedGA algorithm are shown in Figure 4.18 where all local models diverge with a training accuracy of 1.5% and a testing accuracy of 9% for all edge nodes.

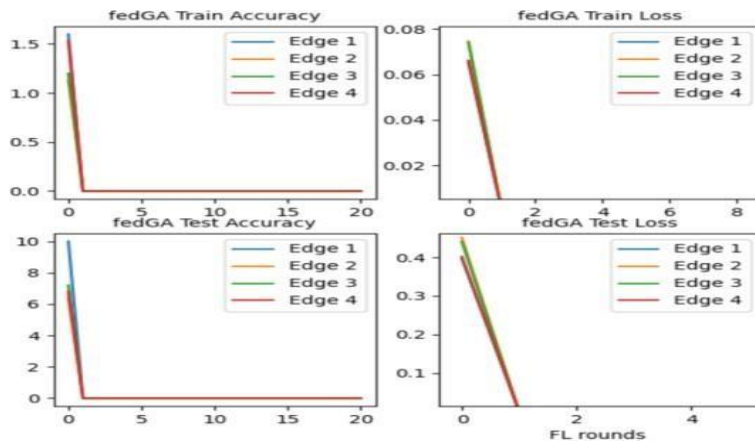


Figure 4.18: FedGA(2).

• **FedPer**

With FedPer, local models are divided into base layers and custom layers. The layers analyzed by the individual are not communicated to the server, and only the base layers are aggregated using TL. The results presented in figure 4.19 show a convergence up to the third round for all local edge node models of 1.7% in the learning phase and 9% in the test phase, confirming a Regression in learning accuracy compared to FedAVG.

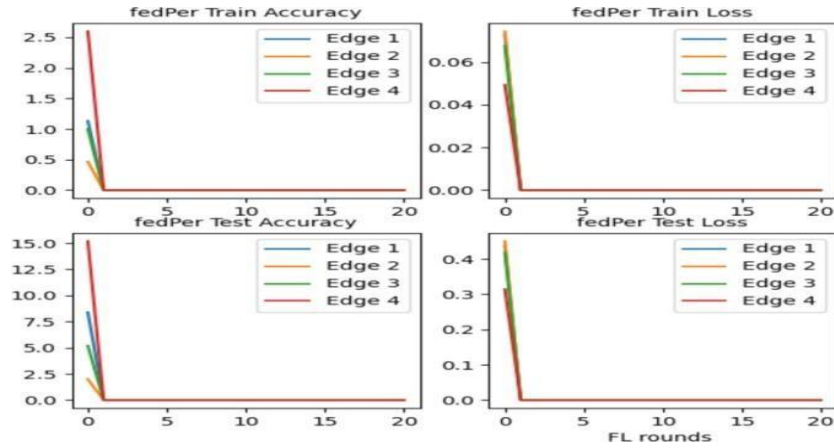


Figure 4.19: FedPer(2).

MNIST dataset with DenseNet model

For this scenario, we consider that we have tested the development of the four edges with the MNIST dataset and the DenseNet model.

- **FedAVG**

By applying FedAVG as an aggregation algorithm with the EMNIST dataset with (ANN) model , the results displayed in Figure 4.20 show a value of 0.5% in the training phase and 3% in the testing phase.

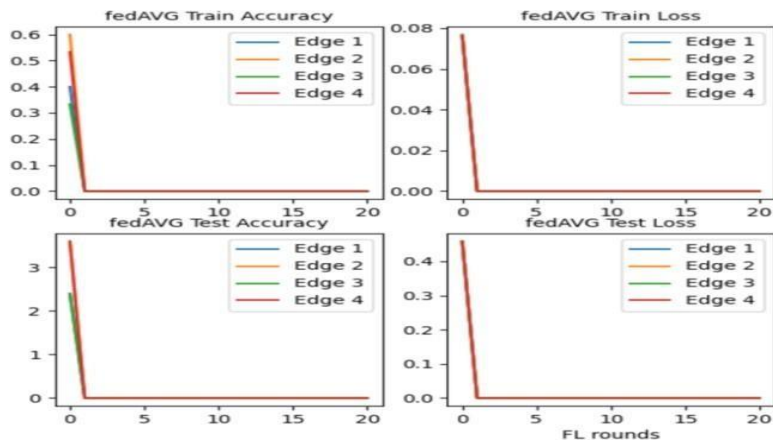


Figure 4.20: FedAVG(3).

- **FedGA**

The results of using the FedGA algorithm are shown in Figure 4.21 where all local models diverge with a training accuracy of 0.8% and a testing accuracy of 4.5% for all edge nodes.

- **FedPer**

With FedPer, local models are divided into base layers and custom layers. The layers analyzed by the individual are not communicated to the server, and only the

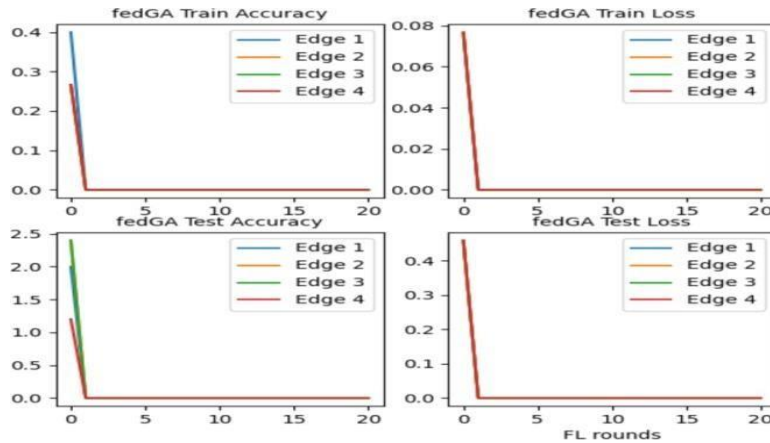


Figure 4.21: FedGA(3).

Base layers are aggregated using TL. The results presented in figure 4.22 show a convergence up to the third round for all local edge node models of 0.6% in the learning phase and 4% in the test phase.

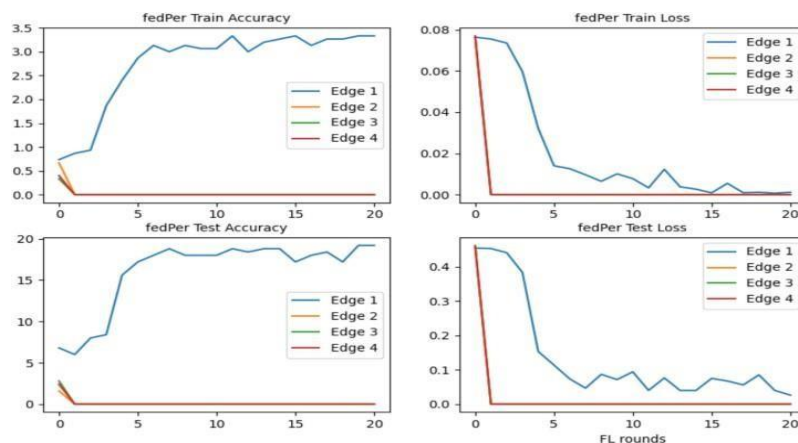


Figure 4.22: FedPer(3).

Cifar-10 dataset with Convolutional Neural Network model

In this scenario, we consider that we have tested the development of the four edges with the Cifar-10 dataset with Convolutional Neural Network model.

- **FedAVG**

By applying FedAVG as an aggregation algorithm with the EMNIST dataset with (ANN) model, the results displayed in Figure 4.23 show a value of 1.9% in the training phase and 8.8% in the testing phase.

- **FedGA**

The results of using the FedGA algorithm are shown in Figure 4.24 where all local models diverge with a training accuracy of 1.6% and a testing accuracy of 8% for all edge nodes.

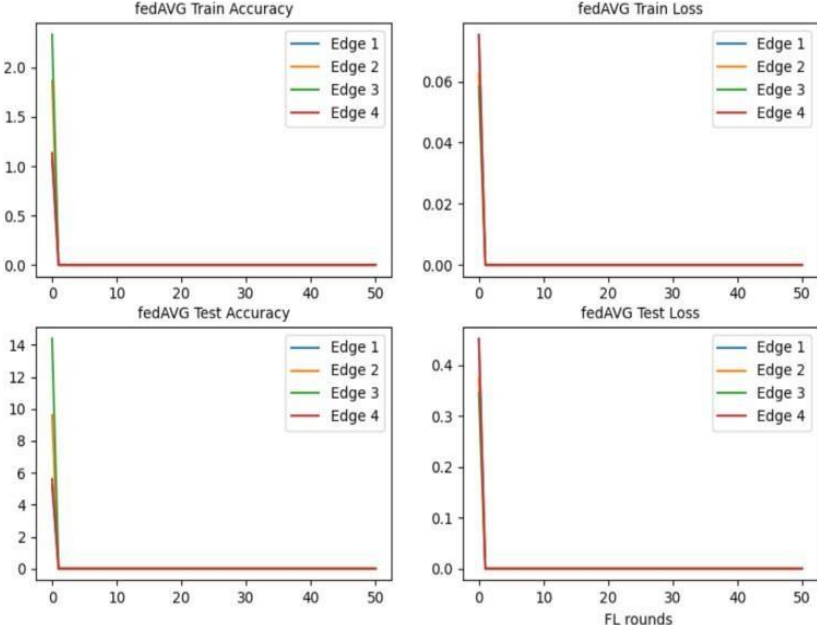


Figure 4.23: FedAVG(4).

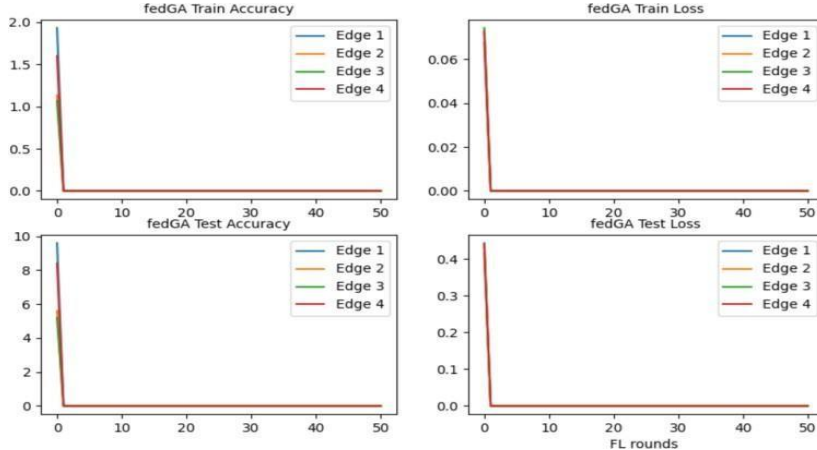


Figure 4.24: FedGA(4).

- **FedPer**

With FedPer, local models are divided into base layers and custom layers. The layers analyzed by the individual are not communicated to the server, and only the base layers are aggregated using TL. The results presented in figure ?? show a convergence up to the third round for all local edge node models of 1 .5% in the learning phase and 9% in the test phase.

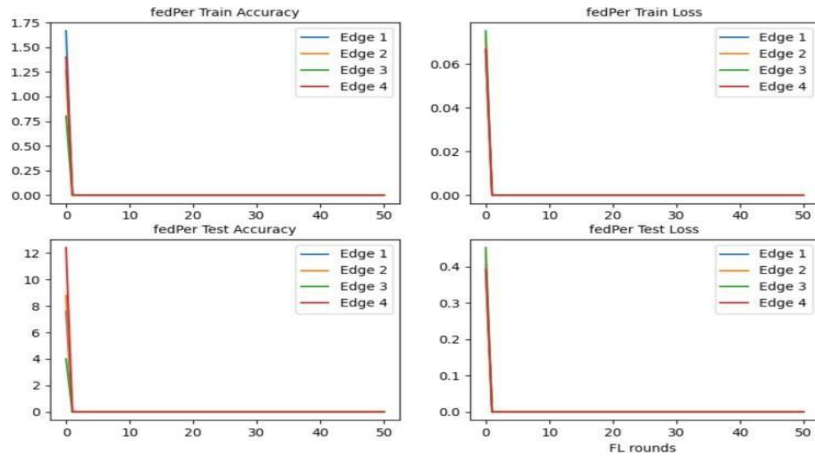


Figure 4.25: FedPer(4).

4.3 Discussions and comparison

Based on the experiments conducted and the results obtained in the previous section through both steps of the FedGA aggregation algorithm and the developed TL procedure, we have drawn several observations.

In the first scenario, which uses the Fashion MNIST dataset with a Recurrent Neural Network (RNN) model, we observed that FedPer performs better compared to both FedGA and FedAVG. Although FedGA and FedAVG are very close in performance, FedAVG slightly outperforms FedGA. This indicates that FedPer is particularly effective with this type of dataset and model combination.

For the second scenario, which relies on the EMNIST dataset with an Artificial Neural Network (ANN) model, FedPer again shows outstanding performance compared to FedAVG and FedGA. FedPer stands out above both algorithms, though FedPer and FedAVG are relatively close in terms of performance. This suggests that FedPer maintains its superiority across different types of neural network models and datasets.

In the third scenario, which uses the MNIST dataset with a DenseNet model, the results are very similar to those of the second scenario. This similarity arises due to the partial sharing of data between these scenarios. Additionally, we noted that the type of processor used has a significant impact on improving the accuracy of the graphs. This highlights the importance of hardware in optimizing model performance.

Furthermore, it is important to emphasize the strength of FedGA-ICPS. Its main

Chapter 4. FedGA-ICPS Implementation and Experiments

advantage lies in its ability to automatically correct a node when its model is weak by requesting a pre-trained model, if one exists. This capability ensures robustness and adaptability, making it a powerful tool in federated learning environments.

Overall, datasets with fewer objects tend to yield better results compared to those with more objects, as evidenced by the comparative performance of the algorithms. We also tested both the EMNIST and MNIST datasets with two optimization algorithms: Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). The results underscore the versatility and efficacy of these optimization techniques in enhancing model performance in various scenarios.

4.4 Conclusion

In this chapter, we have defined the work of Mlle. Guendouzi Badra Souhila [100]. We have integrated datasets such as Fashion MNIST, EMNIST, MNIST, and CIFAR-10 into this work and made topology changes including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs), Artificial Neural Networks (ANNs), and DenseNet. Furthermore, we conducted tests with two optimization algorithms, Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), using the EMNIST and MNIST datasets. We also compared the results between FedPer, FedAVG, and FedGA for each dataset and model.

General Conclusion

In today's world, technological advancements have led to an era of interconnected devices, providing us with unprecedented amounts of data. This data has fueled the success and widespread application of artificial intelligence across various sectors. However, concerns regarding data confidentiality and user privacy present significant obstacles to utilizing this sensitive information. Federated Learning (FL) has recently emerged as a promising solution to this issue. FL enables distributed client devices to collaboratively train a shared prediction model while keeping all training data locally on the individual devices. In recent years, this approach has garnered significant attention in both industry and academia. Leading tech companies have already implemented FL in production across multiple domains to tackle privacy and data collection challenges.

In this work, we evaluated the innovative solution, FedGA-ICPS, which was developed by Mademoiselle Guendouzi Badra Souhila. [100]. Extensive tests were conducted on multiple benchmarks and topologies, including datasets such as Fashion MNIST, EMNIST, MNIST, and CIFAR-10. We also experimented with models like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs), Artificial Neural Networks (ANNs), and DenseNet.

Additionally, we tested two optimization algorithms, Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), using the EMNIST and MNIST datasets. These tests aimed to evaluate the effectiveness of these optimization techniques in enhancing model performance. The results highlight the versatility and efficacy of PSO and ACO in various scenarios, demonstrating their potential to improve the performance of Federated Learning models in diverse environments.

As a future perspective, we would like to have the opportunity to test this work on a machine with powerful computational capabilities, including a GPU. Currently, our PC lacks a GPU, which limits our ability to perform multithreading. Therefore, to validate these tests, it is necessary to conduct them on a larger scale. Additionally, we can explore testing other genetic algorithms to further enhance our study.

Bibliography

- [1] Mohammad Aazam and Eui-Nam Huh. Fog computing and smart gateway based communication for cloud of things. In *2014 International conference on future internet of things and cloud*, pages 464–470. IEEE, 2014.
- [2] Mohammad Aazam and Eui-Nam Huh. Fog computing micro datacenter based dynamic resource estimation and pricing model for iot. In *2015 IEEE 29th international conference on advanced information networking and applications*, pages 687–694. IEEE, 2015.
- [3] Maysam F Abbod, James WF Catto, Derek A Linkens, and Freddie C Hamdy. Application of artificial intelligence to the management of urological cancer. *The Journal of urology*, 178(4):1150–1156, 2007.
- [4] Naeem Akhtar, Anurag Rana, Ruhul Amin, Hitumoni Nath, Kalpana Nath, Parthvi Kirankumar Jingar, and Selvan Ravindran. Analyze the impact of digital transformation on learning using soft computing. *International Journal of Intelligent Systems and Applications in Engineering*, 12(14s):565–572, 2024.
- [5] Ali Al Bataineh, Devinder Kaur, and Seyed Mohammad J Jalali. Multi-layer perceptron training optimization using nature inspired computing. *IEEE Access*, 10: 36963–36977, 2022.
- [6] Vítor Alcácer and Virgilio Cruz-Machado. Scanning the industry 4.0: A literature review on technologies for manufacturing systems. *Engineering science and technology, an international journal*, 22(3):899–919, 2019.
- [7] Rasim Alguliyev, Yadigar Imamverdiyev, and Lyudmila Sukhostat. Cyber-physical systems and their security issues. *Computers in Industry*, 100:212–223, 2018.
- [8] Tor Anderson, Chin-Yao Chang, and Sonia Martínez. Distributed approximate newton algorithms and weight design for constrained optimization. *Automatica*, 109: 108538, 2019.
- [9] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818*, 2019.
- [10] Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili. Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications. *Journal of Network and Computer Applications*, 82:152–165, 2017.

Bibliography

- [11] Eddy Bajic. Localisation et identification de ressources industrielles par l'internet des objets. *GeSI-Revue des Départements de Génie Electrique et Informatique Industrielle*, (92):19–27, 2018.
- [12] Mohamed Ben-Daya, Elkafi Hassini, and Zied Bahroun. Internet of things and supply chain management: a literature review. *International journal of production research*, 57(15-16):4719–4742, 2019.
- [13] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. *Big data and internet of things: A roadmap for smart environments*, pages 169–186, 2014.
- [14] Hung Cao, Monica Wachowicz, Chiara Renso, and Emanuele Carlini. Analytics everywhere: generating insights from the internet of things. *Ieee Access*, 7:71749–71769, 2019.
- [15] Adrian Carrio, Carlos Sampedro, Alejandro Rodriguez-Ramos, Pascual Campoy, et al. A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors*, 2017, 2017.
- [16] Gang Chen. A gentle tutorial of recurrent neural network with error backpropagation. *arXiv preprint arXiv:1610.02583*, 2016.
- [17] Yang Chen, Yu Chen, Qiang Cao, and Xiaowei Yang. Packetcloud: A cloudlet-based open platform for in-network services. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1146–1159, 2015.
- [18] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 35(4):83–93, 2020.
- [19] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [20] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- [21] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International conference on machine learning*, pages 2067–2075. PMLR, 2015.
- [22] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.
- [23] Saneev Kumar Das and Sujit Beborntta. Heralding the future of federated learning framework: architecture, tools and future directions. In *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 698–703. IEEE, 2021.

Bibliography

- [24] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of things*, pages 61–75. Elsevier, 2016.
- [25] Christian W Dawson and Robert Wilby. An artificial neural network approach to rainfall-runoff modelling. *Hydrological Sciences Journal*, 43(1):47–66, 1998.
- [26] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and trends® in signal processing*, 7(3–4):197–387, 2014.
- [27] Niklas Donges. Gradient descent: an introduction to 1 of machine learning’s most popular algorithms. *Built In*, 23, 2021.
- [28] Zhaoyang Du, Celimuge Wu, Tsutomu Yoshinaga, Kok-Lim Alvin Yau, Yusheng Ji, and Jie Li. Federated learning for vehicular internet of things: Recent advances and open issues. *IEEE Open Journal of the Computer Society*, 1:45–61, 2020.
- [29] Rohollah Fallah Madvari. Artificial intelligence (ai), machine learning (ml) and deep learning (dl) on health, safety and environment (hse). *Archives of Occupational Health*, 6(4):1321–1322, 2022.
- [30] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.
- [31] Nicole Gruber and Alfred Jockisch. Are gru cells more specific and lstm cells more sensitive in motive classification of text? *Frontiers in artificial intelligence*, 3:40, 2020.
- [32] Souhila Badra Guendouzi, Samir Ouchani, and Mimoun Malki. Genetic algorithm based aggregation for federated learning in industrial cyber physical systems. In *Computational Intelligence in Security for Information Systems Conference*, pages 12–21. Springer, 2022.
- [33] Souhila Badra Guendouzi, Samir Ouchani, and Mimoune Malki. Enhancing the aggregation of the federated learning for the industrial cyber physical systems. In *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 197–202. IEEE, 2022.
- [34] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.
- [35] Sarra Hammoudi, Zibouda Aliouat, and Saad Harous. Challenges and research directions for internet of things. *Telecommunication Systems*, 67:367–385, 2018.
- [36] Meng Hao, Hongwei Li, Xizhao Luo, Guowen Xu, Haomiao Yang, and Sen Liu. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics*, 16(10):6532–6542, 2019

Bibliography

- [37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [38] Chris Jay Hoofnagle, Bart Van Der Sloot, and Frederik Zuiderveen Borgesius. The european union general data protection regulation: what it is and what it means. *Information & Communications Technology Law*, 28(1):65–98, 2019.
- [39] Junyan Hu, Hanlin Niu, Joaquin Carrasco, Barry Lennox, and Farshad Arvin. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(12):14413–14423, 2020.
- [40] Shih-Chia Huang and Trung-Hieu Le. *Principles and labs for deep learning*. Academic Press, 2021.
- [41] Ahmed Imteaj and M Hadi Amini. Fedar: Activity and resource-aware federated learning model for distributed mobile robots. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1153–1160. IEEE, 2020.
- [42] Michaela Iorga, Larry Feldman, Robert Barton, Michael J Martin, Nedim S Goren, and Charif Mahmoudi. Fog computing conceptual model. 2018.
- [43] Md Aminur Islam, F Bin Abul Kasem, S Khan, MT Habib, and F Ahmed. Cloud computing in education: potentials and challenges for bangladesh. *International Journal of Computer Science, Engineering and Applications*, 7(5):11–21, 2017.
- [44] Marija Jegorova, Chaitanya Kaul, Charlie Mayor, Alison Q O’Neil, Alexander Weir, Roderick Murray-Smith, and Sotirios A Tsaftaris. Survey: Leakage and privacy at inference time. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [45] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.
- [46] I Kevin, Kai Wang, Xiaokang Zhou, Wei Liang, Zheng Yan, and Jinhua She. Federated transfer learning based cross-domain prediction for smart manufacturing. *IEEE Transactions on Industrial Informatics*, 18(6):4088–4096, 2021.
- [47] Latif U Khan, Shashi Raj Pandey, Nguyen H Tran, Walid Saad, Zhu Han, Minh NH Nguyen, and Choong Seon Hong. Federated learning for edge networks: Resource optimization and incentive mechanism. *IEEE Communications Magazine*, 58(10):88–93, 2020.

Bibliography

- [48] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

Bibliography

- [49] Hyungbin Kim, Yongho Kim, and Hyunhee Park. Reducing model cost based on the weights of each layer for federated learning clustering. In *2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 405–408. IEEE, 2021.
- [50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [51] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3347–3366, 2021.
- [52] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information systems frontiers*, 17:243–259, 2015.
- [53] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020.
- [54] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE internet of things journal*, 4(5):1125–1142, 2017.
- [55] Yuan-Pin Lin and Tzyy-Ping Jung. Improving eeg-based emotion classification using conditional transfer learning. *Frontiers in human neuroscience*, 11:334, 2017.
- [56] Hong Liu, Shuaipeng Zhang, Pengfei Zhang, Xinqiang Zhou, Xuebin Shao, Geguang Pu, and Yan Zhang. Blockchain and federated learning for collaborative intrusion detection in vehicular edge computing. *IEEE Transactions on Vehicular Technology*, 70(6):6073–6084, 2021.
- [57] Tom H Luan, Longxiang Gao, Zhi Li, Yang Xiang, Guiyi Wei, and Limin Sun. Fog computing: Focusing on mobile users at the edge. *arXiv preprint arXiv:1502.01815*, 2015.
- [58] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agueray Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.
- [59] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.
- [60] Luis Miralles-Pechuán, Dafne Rosso, Fernando Jiménez, and Jose M Garcia. A methodology based on deep learning for advert value calculation in cpm, cpc and cpa networks. *Soft Computing*, 21(3):651–665, 2017.
- [61] Hussein Mouzannar, Yara Rizk, and Mariette Awad. Damage identification in social media posts using multimodal deep learning. In *ISCRAM*. Rochester, NY, USA, 2018.

Bibliography

- [62] Ranesh Kumar Naha, Saurabh Garg, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Longxiang Gao, Yong Xiang, and Rajiv Ranjan. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE access*, 6: 47980–48009, 2018.
- [63] Trong Nguyen. An empirical evaluation of the implementation of the california consumer privacy act (ccpa). *arXiv preprint arXiv:2205.09897*, 2022.
- [64] Lina Ni, Xu Gong, Jufeng Li, Yuncan Tang, Zhuang Luan, and Jinqian Zhang. rfdfw: Secure and trustable aggregation scheme for byzantine-robust federated learning in internet of things. *Information Sciences*, 653:119784, 2024.
- [65] Chris Nicholson. A beginner’s guide to lstms and recurrent neural networks. *SkyMind. Saatavissa: <https://skymind.ai/wiki/lstm>. Hakupäivä*, 6:2019, 2019.
- [66] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.
- [67] Dawid Polap, Gautam Srivastava, and Keping Yu. Agent architecture of an intelligent medical system based on federated learning and blockchain technology. *Journal of Information Security and Applications*, 58:102748, 2021.
- [68] Adnan Qayyum, Kashif Ahmad, Muhammad Ahtazaz Ahsan, Ala Al-Fuqaha, and Junaid Qadir. Collaborative federated learning for healthcare: Multi-modal covid-19 diagnosis at the edge. *IEEE Open Journal of the Computer Society*, 3:172–184, 2022.
- [69] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.
- [70] Youyang Qu, Shiva Raj Pokhrel, Sahil Garg, Longxiang Gao, and Yong Xiang. A blockchained federated learning framework for cognitive computing in industry 4.0 networks. *IEEE Transactions on Industrial Informatics*, 17(4):2964–2973, 2020.
- [71] Mirco Ravanelli, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. Light gated recurrent units for speech recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2):92–102, 2018.
- [72] Jinke Ren, Yinghui He, Guan Huang, Guanding Yu, Yunlong Cai, and Zhaoyang Zhang. An edge-computing based architecture for mobile augmented reality. *IEEE Network*, 33(4):162–169, 2019.
- [73] Independently RNN. Recurrent neural network. 2021.
- [74] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things: An overview. *The internet society (ISOC)*, 80(15):1–53, 2015.
- [75] Rituparna Saha, Sudip Misra, and Pallav Kumar Deb. Fogfl: Fog-assisted federated learning for resource-constrained iot devices. *IEEE Internet of Things Journal*, 8(10):8456–8463, 2020.

Bibliography

- [76] Prashanth Saravanan. Understanding loss functions in machine learning. *Engineering Education (EngEd) Program/ Section*, 2021.
- [77] Iqbal H Sarker. Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6):420, 2021.
- [78] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [79] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.
- [80] Yuanhang Su and C-C Jay Kuo. On extended long short-term memory and dependent bidirectional recurrent neural network. *Neurocomputing*, 356:151–161, 2019.
- [81] Tao Sun, Dongsheng Li, and Bao Wang. Decentralized federated averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4289–4301, 2022.
- [82] Pu Tian, Zheyi Chen, Wei Yu, and Weixian Liao. Towards asynchronous federated learning based threat detection: A dc-adam approach. *Computers & Security*, 108:102344, 2021.
- [83] Amy JC Trappey, Charles V Trappey, Usharani Hareesh Govindarajan, Allen C Chuang, and John J Sun. A review of essential standards and patent landscapes for the internet of things: A key enabler for industry 4.0. *Advanced Engineering Informatics*, 33:208–229, 2017.
- [84] Pengfei Wang, Chao Yao, Zijie Zheng, Guangyu Sun, and Lingyang Song. Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems. *IEEE Internet of Things Journal*, 6(2):2872–2884, 2018.
- [85] Wendong Wang, Cheng Feng, Bo Zhang, and Hui Gao. Environmental monitoring based on fog computing paradigm and internet of things. *IEEE Access*, 7:127154–127165, 2019.
- [86] Jeremy West, Dan Ventura, and Sean Warnick. Spring research presentation: A theoretical foundation for inductive transfer. *Brigham Young University, College of Physical and Mathematical Sciences*, 1(08), 2007.
- [87] Md Whaiduzzaman, Anjum Naveed, and Abdullah Gani. Mobicore: Mobile device based cloudlet resource enhancement for optimal task response. *IEEE transactions on services computing*, 11(1):144–154, 2016.
- [88] Xiaolong Xu, Hao Tian, Xuyun Zhang, Lianyong Qi, Qiang He, and Wanchun Dou. Discov: Distributed covid-19 detection on x-ray images with edge-cloud collaboration. *IEEE Transactions on Services Computing*, 15(3):1206–1219, 2022.
- [89] Zirui Xu, Zhao Yang, Jinjun Xiong, Janlei Yang, and Xiang Chen. Elfish: Resource-aware federated learning on heterogeneous edge devices. *Ratio*, 2(r1):r2, 2019.

Bibliography

- [90] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.
- [91] Jingjing Yao and Nirwan Ansari. Enhancing federated learning in fog-aided iot by cpu frequency and wireless power control. *IEEE Internet of Things Journal*, 8(5):3438–3445, 2020.
- [92] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. *Dive into deep learning*. Cambridge University Press, 2023.
- [93] Huaqing Zhang, Yanru Zhang, Yunan Gu, Dusit Niyato, and Zhu Han. A hierarchical game framework for resource management in fog computing. *IEEE Communications Magazine*, 55(8):52–57, 2017.
- [94] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training. *arXiv preprint arXiv:1903.03614*, 2019.
- [95] Weishan Zhang, Tao Zhou, Qinghua Lu, Xiao Wang, Chunsheng Zhu, Haoyun Sun, Zhipeng Wang, Sin Kit Lo, and Fei-Yue Wang. Dynamic-fusion-based federated learning for covid-19 detection. *IEEE Internet of Things Journal*, 8(21):15884–15891, 2021.
- [96] Zehui Zhang, Ningxin He, Dongyu Li, Hang Gao, Tiegang Gao, and Chuan Zhou. Federated transfer learning for disaster classification in social computing networks. *Journal of safety science and resilience*, 3(1):15–23, 2022.
- [97] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.
- [98] Chunyi Zhou, Anmin Fu, Shui Yu, Wei Yang, Huaqun Wang, and Yuqing Zhang. Privacy-preserving federated learning in fog computing. *IEEE Internet of Things Journal*, 7(11):10782–10793, 2020.
- [99] Wenbo Zhu, Yan Ma, Yizhong Zhou, Michael Benton, and Jose Romagnoli. Deep learning based soft sensor and its application on a pyrolysis reactor for compositions predictions of gas phase components. In *Computer Aided Chemical Engineering*, volume 44, pages 2245–2250. Elsevier, 2018.
- [100] GUENDOUZI, BAdra Souhila. Federated Learning in Distributed Cyber Physical Systems: A State-of-the-Art. Diss. 2022.

Abstract

This thesis is part of our end-of-cycle project to obtain a master's degree in network and data engineering. The project explores Federated Learning in Distributed Networks, specifically leveraging collaborative machine learning without compromising data privacy. We evaluated the FedGA-ICPS approach developed by Ms. Badra Guendouzi on various datasets to assess its performance in heterogeneous environments. Extensive testing was conducted on benchmarks such as Fashion MNIST, EMNIST, MNIST, and CIFAR-10, using models like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs), Artificial Neural Networks (ANNs), and DenseNet. In addition, we tested two optimization algorithms, Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), with the EMNIST and MNIST datasets. These tests were conducted to evaluate the effectiveness of these optimization techniques in improving model performance. The results underscore the versatility and efficacy of PSO and ACO in various scenarios, demonstrating their potential to enhance the accuracy and efficiency of our models within the FedGA-ICPS framework. The project implementation was done in Python, as it is the most suitable language for this type of project, offering rich libraries in the domain.

Keywords :

Federated learning, machine learning, FEDGA-icps, benchmarks, models, Python.

Résumé

Cette thèse fait partie de notre projet de fin de cycle pour l'obtention d'un master en ingénierie des réseaux et des données. Le projet explore l'apprentissage fédéré dans les réseaux distribués, en mettant l'accent sur l'utilisation de l'apprentissage collaboratif sans compromettre la confidentialité des données. Nous avons évalué l'approche FedGA-ICPS développée par Mme Badra Guendouzi sur divers ensembles de données pour mesurer ses performances dans des environnements hétérogènes. Des tests approfondis ont été réalisés sur des jeux de données de référence tels que Fashion MNIST, EMNIST, MNIST et CIFAR-10, en utilisant des modèles tels que les réseaux de neurones convolutifs (CNN), les réseaux de neurones récurrents (RNN), les unités récurrentes avec portes (GRU), les réseaux de neurones artificiels (ANN) et DenseNet. De plus, nous avons testé deux algorithmes d'optimisation, l'optimisation par essaim particulaire (PSO) et l'optimisation par colonies de fourmis (ACO), avec les ensembles de données EMNIST et MNIST. Ces tests ont été menés pour évaluer l'efficacité de ces techniques d'optimisation dans l'amélioration des performances des modèles. Les résultats mettent en évidence la polyvalence et l'efficacité de la PSO et de l'ACO dans divers scénarios, démontrant leur potentiel à améliorer la précision et l'efficacité de nos modèles dans le cadre de FedGA-ICPS. La mise en œuvre du projet a été réalisée en Python, car il s'agit du langage le plus approprié pour ce type de projet, offrant des bibliothèques riches dans le domaine.

Mots-clés :

Apprentissage fédéré, apprentissage automatique, FEDGA-icps, jeux de données, modèles, Python.

ملخص

نُعدّ هذه الرسالة جزءًا من مشروع زهابة الدورة للحصول على درجة الماجستير في هندسة الشبكات والبيانات. يهدف المشروع إلى تعلم النبرالي في الشبكات الموزعة، مع التركيز على السنادة من التعلم التعاوني دون المساس بخصوصية البيانات. لُذ فمنا بتؤيم نهج FedGA-ICPS الذي طورته السيدة بدرة وندوزي على مجموعات بيانات مختلطة لتؤيم أداؤه في

البيات غير المتجانسة. تم إجراء اختبارات مكثفة على مجموعات بيانات مرجعية مثل Fashion MNIST، EMNIST، MNIST و CIFAR-10، باستخدام نماذج مثل الشبكات العصبية الالتفانية (CNNs)، الشبكات العصبية

التفرارية (RNNs)، الوحدات العصبية التفرارية المتغلقة (GRUs)، الشبكات العصبية الصطراعية (ANNs) و DenseNet. بالاضافة إلى ذلك، فمنا باختيار خوارزميتين للتحسين، وهما تحسين السرب الجزيئي (PSO) و تحسين مرعات النمل (ACO)، مع مجموعات بيانات EMNIST و MNIST. أُجريت هذه الاختبارات لتؤيم نوعية هذه

التؤينات في تحسين أداء النماذج. تؤكد النتائج على تنوع ونوعية PSO و ACO في سيناريوهات مختلطة، مما يظن إمكانيتهم في تيز دقة وكفاءة نماذجنا في إطار FedGA-ICPS. تم تنفيذ المشروع باستخدام لغة Python، لكونها اللغة الأتسب لهذا النوع من المشاريع، حيث توفر مكتبات غنية في هذا المجال.

الكلمات المفتاحية:

تعلم النبرالي، التعلم الآلي، FEDGA-ICPS، المعابر المرجعية، النماذج، بايؤن.