



University of
Zurich^{UZH}

Design and Implementation of Environmental Sustainability Module for Decentralized Federated Learning

Xi Cheng
Zurich, Switzerland
Student ID: 21-742-945

Supervisor: Chao Feng, Dr. Alberto Huertas Celdrán
Date of Submission: August 6, 2024

Abstrakt

Die schnelle Einführung von Distributed Federated Learning (DFL)-Methoden im Training von KI-Modellen verdeutlicht ein dringendes Problem: Der Einsatz von DFL führt zu einem erheblichen Energieverbrauch und CO₂-Emissionen, insbesondere in Regionen, die von fossilen Brennstoffen abhängig sind, was den globalen Klimawandel verschärft. Diese Masterarbeit stellt einen innovativen Rechenrahmen vor, der speziell für DFL-Systeme entwickelt wurde, um diese Umweltauswirkungen zu bewerten.

Durch die Bereitstellung einer praktischen Methode zur Berechnung des Energieverbrauchs und des CO₂-Fußabdrucks des DFL-Trainingsprozesses fördert diese Arbeit umweltfreundlichere KI-Praktiken. Der Kernbeitrag umfasst die Schaffung eines Rechenrahmens, der darauf ausgelegt ist, den inhärenten Energieverbrauch und die CO₂-Emissionen des DFL-Prozesses genau zu quantifizieren und zu analysieren. Dies umfasst das lokale Training an den Knoten, die Aggregation und die Kommunikation. Dieser Rahmen wird speziell auf der Fedstellar-Plattform implementiert und bietet detaillierte Einblicke in die Energieeffizienz und CO₂-Emissionen der Plattform. Darüber hinaus wurde eine umfassende Bewertung der ökologischen Nachhaltigkeit im Kontext von DFL durchgeführt, wobei der Einfluss von Faktoren wie der Komplexität der Föderation, der Anzahl der Knoten, den Standorten der Knoten (CO₂-Intensität) und den Beschleunigern (CPU und GPU) auf den gesamten Energieverbrauch und CO₂-Fußabdruck untersucht wurde. Diese Arbeit trägt nicht nur zu einem tieferen Verständnis der Umweltauswirkungen von DFL bei, sondern legt auch den Grundstein für zukünftige Innovationen in der nachhaltigen KI-Technologie.

Abstract

The rapid adoption of Distributed Federated Learning (DFL) methods in training AI models highlights an urgent issue: the use of DFL generates significant energy consumption and carbon emissions, especially in regions dependent on fossil fuels, exacerbating global climate change. This master's thesis introduces an innovative computational framework tailored to DFL systems to evaluate these environmental impacts.

By providing a practical method for calculating the energy consumption and carbon footprint of the DFL training process, this thesis advances more environmentally sustainable AI practices. The core contribution includes the creation of a computational framework designed to accurately quantify and analyze the inherent energy usage and carbon emissions of the DFL process, covering local training at nodes, aggregation, and communication. This framework is specifically implemented on the Fedstellar platform, offering detailed insights into the platform's energy efficiency and carbon emissions. Furthermore, a comprehensive environmental sustainability assessment was conducted within the DFL context, focusing on the impact of factors such as federation complexity, number of nodes, node locations (carbon intensity), and accelerators (CPU and GPU) on the overall energy consumption and carbon footprint. This work not only deepens the understanding of DFL's environmental impact but also lays the groundwork for future innovations in sustainable AI technology.

Acknowledgments

I would like to thank Prof. Dr. Burkhard Stiller for giving me the opportunity to undertake my master thesis in an area of my interest. It is a pleasant and rewarding experience to conduct research at the Communication Systems Group.

Further, I would like to express my sincere gratitude to my supervisors, Chao Feng and Dr. Alberto Huertas Celdrán for their continuous support and guidance throughout my master thesis. Their expertise in data science, coupled with the invaluable suggestions they provided, have been a great help for this thesis. I am thankful for the knowledge and experience gained under their mentorship.

Lastly, since this thesis concludes my studies at university, I would like to thank my family and friends. Their belief in me kept my spirits and motivation high and I am sincerely grateful for their love and support during this process and throughout my life.

Contents

Abstrakt	i
Abstract	iii
Acknowledgments	v
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	4
1.3 Thesis Outline	4
2 Background	7
2.1 Federated Learning	7
2.1.1 Centralized Federated Learning	8
2.1.2 Decentralized Federated Learning	9
2.2 Fedstellar	10
2.2.1 Introduction of Fedstellar	10
2.2.2 Architecture of Fedstellar	10
2.3 Environmental Sustainability	12
2.3.1 Energy Consumption	13
2.3.2 Carbon Emissions and Carbon Intensity	13
2.3.3 Renewable Energy Ratio	14
2.3.4 Power Usage Effectiveness	15
2.3.5 Thermal Design Power	15

3	Related Work	17
3.1	Energy Consumption and Carbon Footprint in Traditional ML, CFL, and DFL	17
3.2	Detailed Review of Previous Works	18
3.2.1	Related Work [14]	19
3.2.2	Related Work [22]	20
3.3	Summary and Insights	21
3.3.1	Findings and Limitations	21
3.3.2	Insights	22
4	Architecture	23
4.1	Calculation of Energy Consumption	25
4.1.1	Training Energy Consumption	25
4.1.2	Aggregation Energy Consumption	26
4.1.3	Communication Energy Consumption	27
4.2	Calculation of Carbon Intensity	29
4.3	Calculation of Carbon Emissions	30
5	Implementation	33
5.1	Fedstellar Platform	33
5.2	Frontend Implementation	35
5.2.1	Variables R_k and PUE_k	35
5.2.2	Variable $E_{\text{byte}}^{(C)}$	38
5.3	Backend Implementation and Visualization	40
5.3.1	Implementation of Calculations at Three Phases	40
5.3.2	Integration of Three Phases	47
5.3.3	Integration of All Nodes	48

<i>CONTENTS</i>	ix
6 Evaluation	51
6.1 Experimental Configurations	51
6.2 Scenario 1: Comparison Across Three Phases	53
6.2.1 Analysis of Experiment 2 Across Three phases	53
6.2.2 Analysis of Experiment 10 Across Three phases	55
6.3 Scenario 2: Federation Complexity	57
6.3.1 Scenario a: Aggregation Algorithm	57
6.3.2 Scenario b: Dataset and Model	59
6.4 Scenario 3: Number of Nodes	61
6.5 Scenario 4: Location	63
6.6 Scenario 5: Accelerator	64
7 Summary and Conclusions	67
7.1 Conclusion	67
7.2 Future Work	68
Bibliography	69
Abbreviations	73
List of Figures	73
List of Tables	77
A Installation Guidelines	81
A.1 Prerequisites	81
A.2 Deploy a virtual environment	81
A.2.1 Create virtual environment	82
A.2.2 Activate the environment	82
A.3 Building from source	83
A.3.1 Obtaining the platform	83

A.3.2	Dependencies	83
A.3.3	Checking the installation	83
A.4	Building the fedstellar participant	83
A.4.1	Docker image (CPU version)	83
A.4.2	Docker image (GPU version)	84
A.5	Checking the docker images	84
A.6	Running Fedstellar	84
A.7	Fedstellar Frontend	85
A.8	Stop Fedstellar	85
B	Contents of the CD	87

Chapter 1

Introduction

1.1 Motivation

Over the last decade, artificial intelligence (AI) has increasingly permeated various aspects of our lives, demonstrating its adaptability across different domains. For instance, in the realm of board games, AI technologies such as DeepMind's AlphaZero [1] have not only mastered Go but have also excelled in chess and Japanese shogi. Intelligent assistants such as Google Assistant [2] and Microsoft's Cortana [3] have become indispensable in both homes and offices, helping users manage schedules and answer questions. In the entertainment sector, AI's impact is also significant. YouTube's recommendation algorithm [4] continuously learns from user viewing habits to suggest personalized video content. Additionally, AI has excelled in creative writing and artistic creation, as exemplified by the AI painting program DeepArt [5], which can generate unique artworks based on specified styles. In the medical field, AI applications like IBM's Watson [6] are crucial, as they can assist in diagnosing diseases and recommending treatment plans, significantly improving the accuracy and efficiency of medical care. In corporate management, AI tools such as HireVue [7] analyze video interviews to help HR departments more effectively assess candidates. In the legal field, predictive policing software like Geolitica [8] has been used to analyze crime patterns and optimize patrol strategies, enhancing urban security. These applications not only demonstrate AI's broad applicability across various fields but also highlight its immense potential in enhancing work efficiency, optimizing decision-making processes, and improving quality of life.

However, the rapid development of AI has also introduced new challenges, particularly in terms of data privacy. Machine learning (ML), a crucial branch of AI, enables computer systems to learn from data and make predictions or decisions. Traditionally, ML processes require centralizing all data for training, which poses a threat to data privacy. To address this issue, Federated Learning (FL) [9] offers a method for distributed learning within isolated data environments, thereby protecting data privacy. Federated learning trains models on local devices and only sends model parameters to a central server for aggregation, thus avoiding the risks associated with centralized data storage. Despite significant progress in protecting data privacy, centralized federated learning (CFL) configurations

still face challenges. For instance, if a central server fails while aggregating the global model, the entire system is impacted, introducing a risk of a single point of failure [10]. To counter this issue, Decentralized Federated Learning (DFL) [11] has emerged. DFL employs a distributed method to train ML models, relying not on a central server but on collaboration between nodes to complete the training. This shift from centralized to decentralized not only enhances the robustness of the system but also further strengthens the protection of data privacy.

While addressing data privacy remains a paramount challenge in the development of AI, it is equally crucial to consider the environmental implications of these systems. As we pursue sustainable technological advancements, the importance of environmental sustainability has sharply increased. This concern is particularly acute during the life cycle of AI systems, where both the training and operational phases are known to consume significant amounts of energy. This energy consumption not only leads to substantial carbon footprints but also raises critical questions about the long-term viability and environmental impact of deploying large-scale AI technologies.

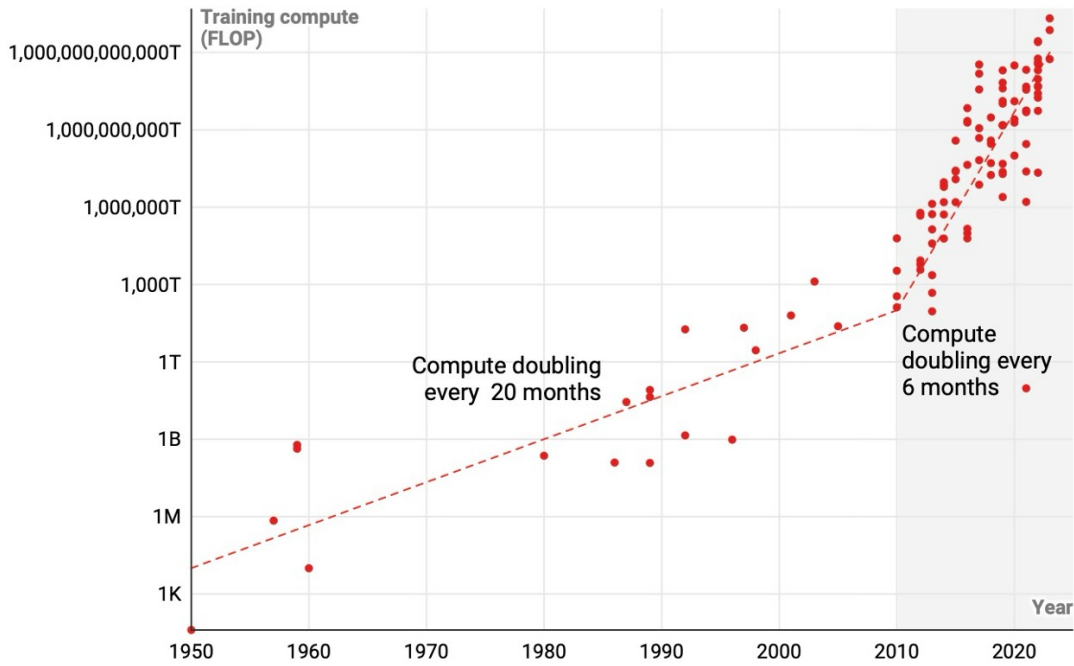


Figure 1.1: Amount of Computing Used to Train Notable AI Models [12]

According to Figure 1.1, the computational power utilized for training AI models has seen a significant increase since the mid-20th century, with a notable acceleration post-2010. In recent years, the computing power necessary for AI training has been doubling every six months—a stark increase from the biennial doubling observed fifty years ago. This exponential growth reflects the increasingly complex nature of AI systems and their expanding applications across various sectors.

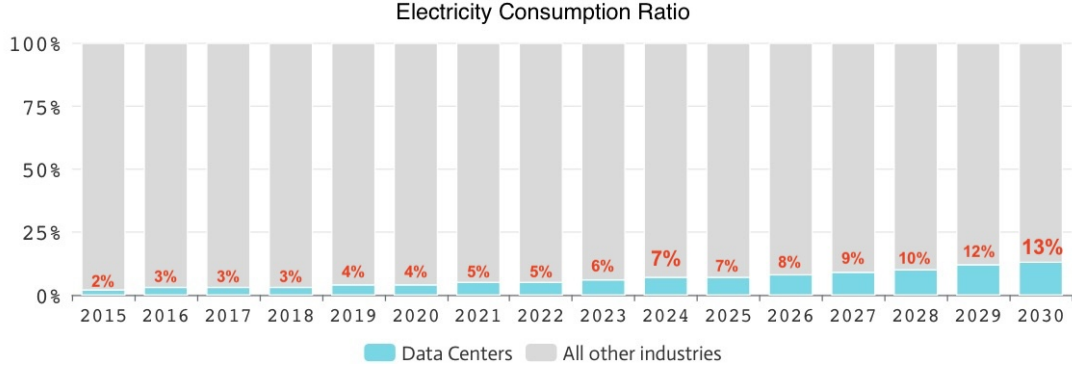


Figure 1.2: Global Electricity Consumption Ratio of Data Centers and All Other Industries [13]

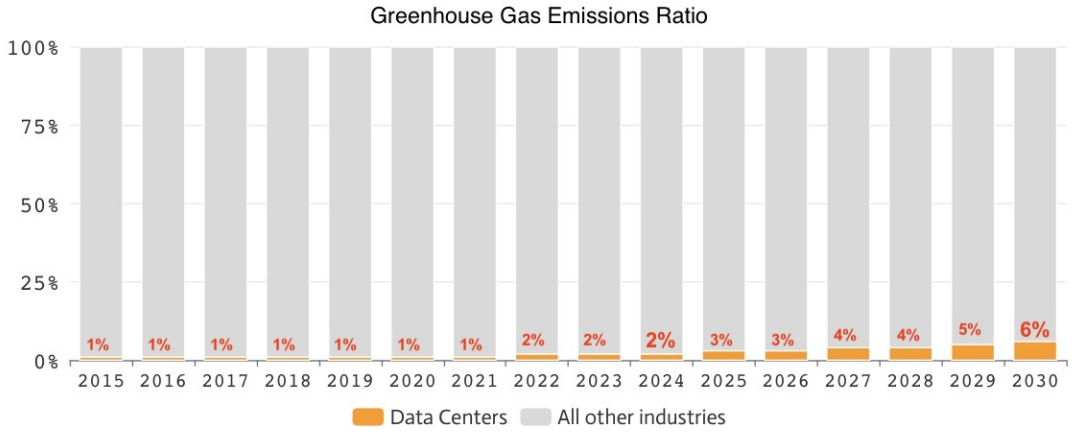


Figure 1.3: Global Greenhouse Gas Emissions Ratio of Data Centers and All Other Industries [13]

The surge in computational demands has correspondingly led to a substantial increase in the energy consumption of data centers, as depicted in Figure 1.2 and Figure 1.3. The energy demands of these data centers have escalated not only in absolute terms but also in their proportion of global energy consumption and greenhouse gas emissions. Projections suggest that by 2030, data centers might account for as much as 13% of annual global electricity usage and 6% of the global carbon footprint. This trend underscores a critical environmental concern linked directly to the advancement of AI technology. This data vividly illustrates that the training processes of large-scale AI models have a profound and growing impact on our environment. Therefore, there is an urgent need to integrate sustainability into the core of AI model training processes.

Considering the notable environmental impact associated with conventional AI infrastructures, it is crucial to investigate more eco-friendly options within AI frameworks, especially in federated learning systems. This prompts us to review the initial endeavors in sustainability within CFL and the need to broaden these concerns to DFL.

The early research in this field established metrics and concepts to quantify the environmental impact of CFL systems and developed a prototype sustainability algorithm [14].

However, the environmental impact of DFL has received less attention. Building on previous work focused on CFL and DFL, this study aims to create and deploy an environmental sustainability computational module for DFL, as well as explore additional metrics by comparing the similarities and differences between DFL and CFL.

1.2 Description of Work

The training of AI models via the DFL method entails local training at the node level, followed by the aggregation and communication of model updates. This process requires significant computational resources. The substantial energy consumption and resultant greenhouse gas emissions from these operations are especially pronounced in regions that depend heavily on fossil fuels, thus exacerbating the global climate change crisis. Given these challenges, this master’s thesis focuses on the following parts to evaluate the environmental impact of DFL systems effectively:

1. **Design of a Computational Framework:** A robust framework was developed to enable precise quantification and analysis of energy consumption and carbon emissions in DFL systems.
2. **Implementation of the Framework to a Specific DFL Platform:** The proposed framework was applied to the DFL platform, Fedstellar [15], allowing for a detailed assessment of the platform’s energy efficiency and carbon output.
3. **Environmental Sustainability Evaluation in the DFL system:** This work conducted a detailed analysis of DFL to assess the environmental sustainability impacts of various operational factors. Key factors included federation complexity, the number of nodes, carbon intensity, and the choice of accelerator.

These efforts contribute by providing a practical method to calculate the energy consumption and carbon emissions associated with the training processes of DFL models, thereby supporting the development of more sustainable artificial intelligence practices.

1.3 Thesis Outline

The remainder of this thesis encompasses the following content: Chapter 2 of this master thesis introduces background knowledge on FL, and Fedstellar platform, as well as conceptions related to environmental sustainability. Chapter 3 explores related work on energy consumption and carbon footprint in traditional ML, CFL, and DFL and conducts a detailed analysis of the papers most relevant to this thesis and identifies limitations. Chapter 4 introduces detailed methodologies for calculating energy consumption and carbon emissions within DFL environments. Chapter 5 focuses on implementing the calculation architecture on Fedstellar platform. Chapter 6 presents evaluation scenarios and their results. This chapter contains a comparative discussion of all evaluation

scenarios, their results, and the limitations of the current design and implementation. Chapter 7 summarizes this master thesis, draws conclusions, and provides insights about future work.

Chapter 2

Background

This chapter provides an overview of the fundamental technologies of FL and its categorization into centralized and decentralized models. It further delves into the introduction of the Fedstellar platform, then followed by an exploration of concepts related to environmental sustainability.

2.1 Federated Learning

In the early stages of AI development, model training often relied on centralized methods that required all data to be stored centrally, and users had to submit their data to a central server (see Figure 2.1). However, as concerns about data privacy have grown, a new need has emerged for training ML models without disclosing personal data to centralized management entities. FL was developed to address this privacy protection issue.

FL employs a distributed method to train machine learning models, where models are trained locally on devices ranging from dozens to millions (see Figure 2.1). The core advantage of this method is that it retains data locally, only sharing parameters used to improve the global model rather than the actual data itself, thus protecting users' privacy. This not only allows individual users to contribute to model training without disclosing their data but also enhances the general applicability and effectiveness of the model.

Specifically, FL is divided into two main types: CFL and DFL. In CFL, all devices send their model updates to a central server, which is responsible for merging these updates and sending the updated model back to each device. This mode facilitates rapid iteration and unified model updates but may introduce the risk of a single point of failure and potential data security risks at the central server. In DFL, model updates are exchanged and synchronized directly between devices without the involvement of a central server. This method enhances the system's resistance to interference and the security of data processing, making the entire system more robust and reliable. The following subsections provide a detailed introduction to these two different types of FL.

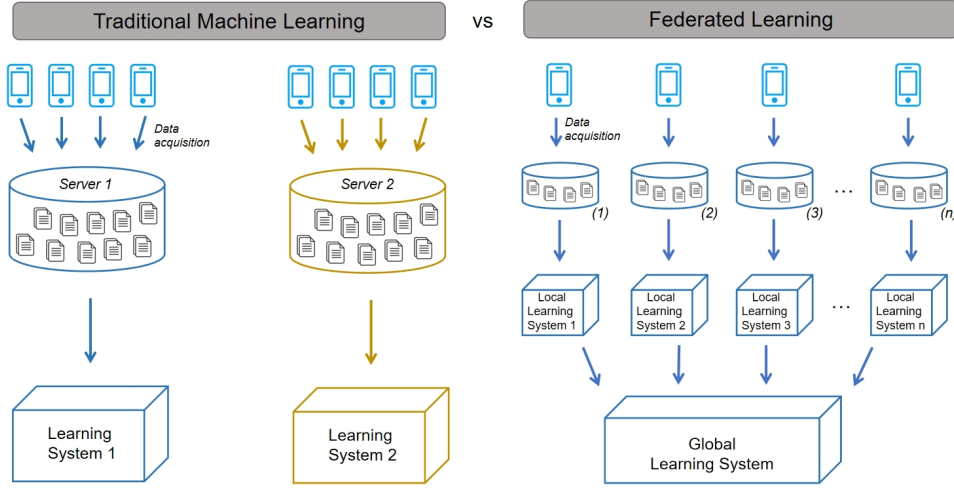


Figure 2.1: Traditional ML (left) and FL (right)

2.1.1 Centralized Federated Learning

CFL is a specific type of FL where a central server takes on the crucial roles of coordinating algorithmic steps and managing all participating nodes throughout the learning process. Compared to traditional machine learning approaches, CFL uniquely supports collaborative model training across distributed devices while ensuring data localization, thus enhancing model training efficiency while safeguarding data privacy and security.

As shown in Figure 2.2, the operation process of CFL is as follows: Initially, each participating node collects data locally and independently trains its machine learning model in its environment. At various stages of the model training, these nodes send their model parameters to the central server. The central server's primary task is to aggregate the parameters received from all the nodes and to update the global model based on this aggregation. Once the global model is updated, the new model parameters are transmitted back to all the nodes. Each node adjusts its local model based on the updated parameters received, a process that often requires multiple iterations, with each iteration involving training and aggregation steps. Through such iterations, the central server eventually integrates a well-optimized, highly convergent global model. This global model is better generalized to new datasets and reflects the collective learning outcomes of all participating nodes.

In summary, CFL provides an efficient and relatively simple way to integrate the learning capabilities of multiple nodes through a central server. Although this centralized structure may increase the risk of a single point of failure, its advantages in handling large-scale distributed learning tasks are undeniable. This method not only enhances the overall performance of the model but also effectively maintains data privacy and security by keeping data localized.

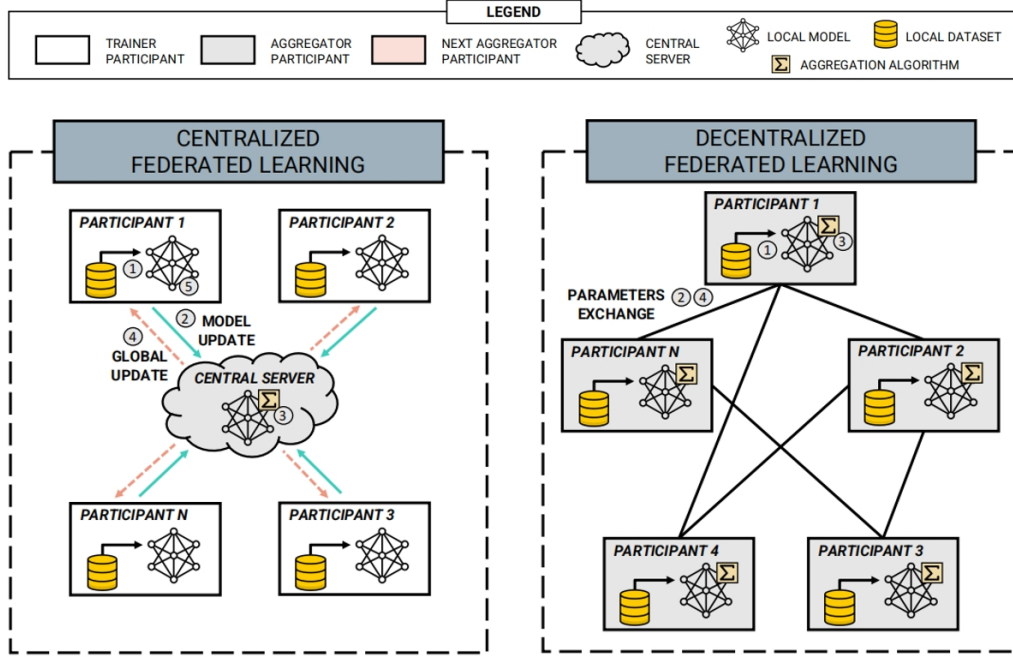


Figure 2.2: CFL (left) and DFL (right) [16]

2.1.2 Decentralized Federated Learning

Due to its centralized architecture, CFL faces significant challenges in terms of data privacy. In this configuration, all local updates from nodes must be transmitted to a central server, which not only increases the risk of data leakage but also exposes the system to single points of failure and communication bottlenecks. To overcome these issues associated with centralization, researchers have developed DFL.

DFL is a form of FL without a central server, where participating nodes coordinate directly with each other to build a global model. As illustrated in Figure 2.2, in a DFL environment, each node independently collects and processes its data. Each node then uses its data to train the model and sends the updates or gradients directly to other interconnected nodes, bypassing any central server. This method makes DFL extremely flexible, and capable of adapting to various network topologies, including fully connected, ring, star, and random configurations. Different topologies can affect the efficiency and effectiveness of the learning process in different ways.

Compared to traditional methods such as ML and CFL, DFL offers many significant advantages. One of the most notable benefits is that original data are not transferred during the model training and updating processes, greatly reducing the potential for data leakage. Additionally, DFL addresses the inherent problem of single points of failure in centralized models, as model updates are exchanged only among nodes without the need for central server coordination. Due to its enhanced security, privacy protection, and flexibility, DFL is an ideal choice for implementing model training.

In summary, DFL not only improves the security and privacy of data processing but also

enhances the system’s robustness and scalability through its decentralized characteristics. These features make DFL particularly advantageous when handling sensitive data or conducting complex model training on distributed devices.

2.2 Fedstellar

2.2.1 Introduction of Fedstellar

Fedstellar [15] is a cutting-edge platform designed to support decentralized federated learning across a wide range of both physical and virtual devices. The platform is engineered to standardize the development, deployment, and management of federated applications, making it easier for users to handle these complex tasks.

The versatility of Fedstellar is evident in its ability to support federations that include a variety of devices, network topologies, and algorithms. It is equipped with advanced management tools and performance metrics that enhance the monitoring and efficiency of the learning process. This capability is supported by modular components that provide robust data storage solutions, asynchronous processing capabilities, and effective mechanisms for model training and communication, ensuring comprehensive oversight of federation activities.

2.2.2 Architecture of Fedstellar

The architectural design of Fedstellar is characterized by its modular structure, which comprises three essential components:

1. **Frontend:** This user-friendly interface allows users to easily set up experiments and monitor their progress, simplifying the interaction with the platform. Here’s a detailed breakdown of what the Frontend offers:
 - *User Interface:* The user interface is designed to be intuitive and user-friendly (illustrated in Figure 2.3), enabling users to efficiently set up their federated learning experiments. Through this interface, users can define various aspects of their federations, including the selection of devices involved, the network topology, the machine learning algorithms to be used, and the datasets on which these algorithms will train.
 - *Advanced Mode:* To cater to more experienced users or specific testing needs, Fedstellar Frontend includes an advanced mode. This mode allows users to evaluate the robustness of their models against potential threats and vulnerabilities. For instance, users can simulate attacks like label flipping, sample poisoning, and model poisoning, as depicted in Figure 2.4.

- *Real-time Monitoring Metrics:* One of the standout features of the Frontend is its capability to provide real-time monitoring (see Figure 2.5). This feature allows users to observe both the performance of the model and the underlying network infrastructure. By providing a comprehensive suite of metrics, Fedstellar enables users to gain deep insights into the effectiveness and efficiency of the ongoing training processes.

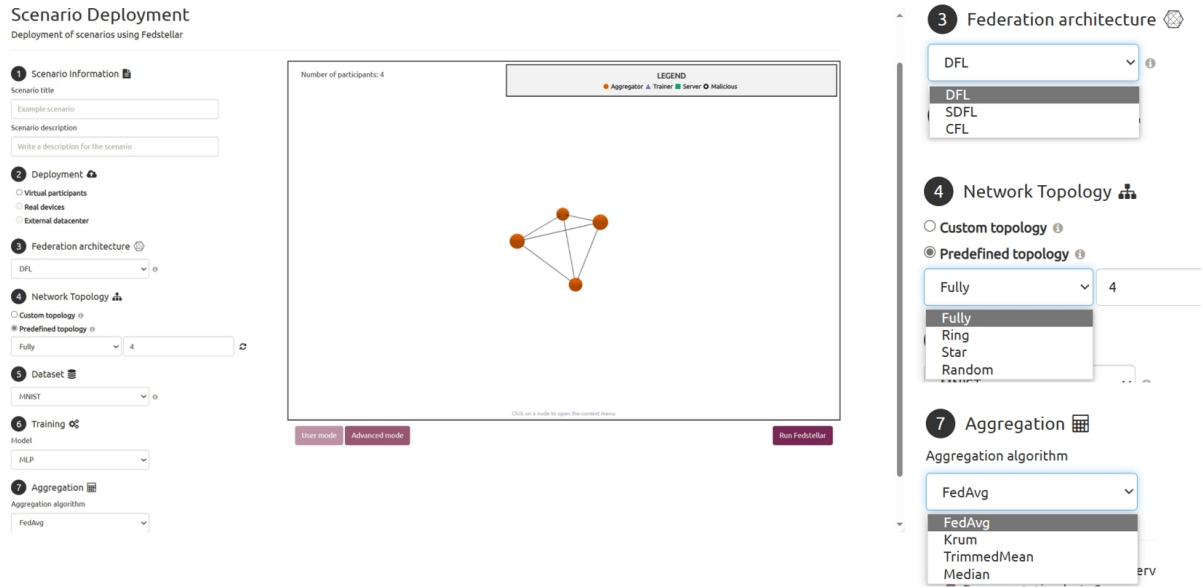


Figure 2.3: Fedstellar User Interface

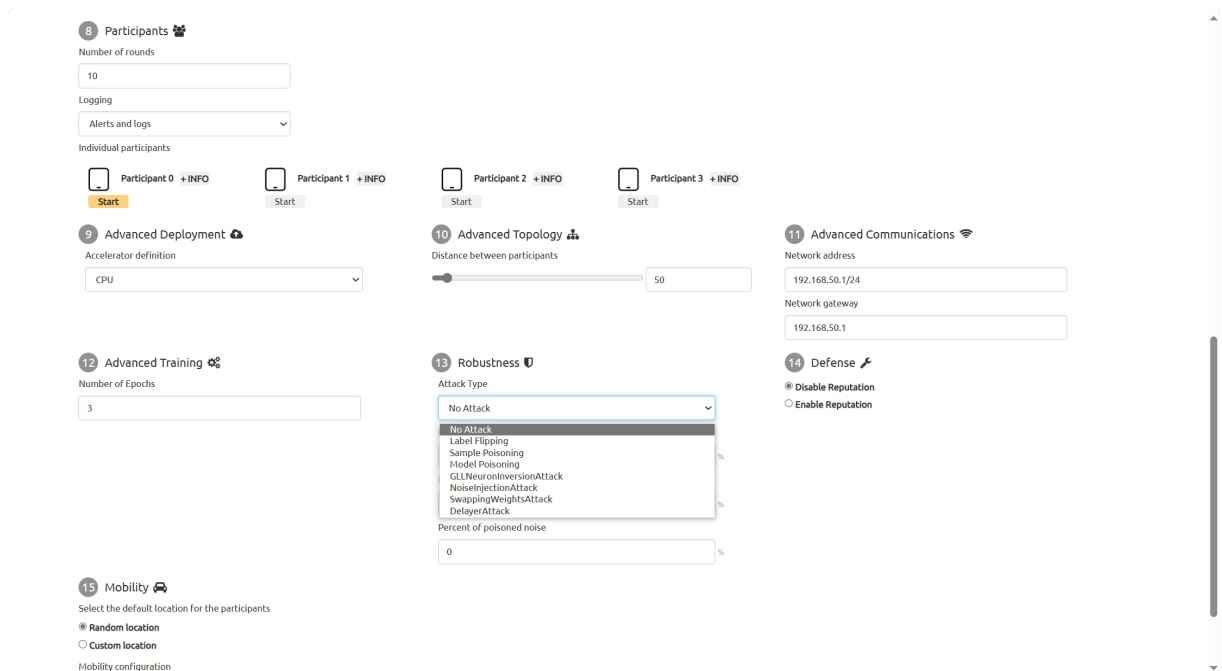


Figure 2.4: Fedstellar Advanced Mode

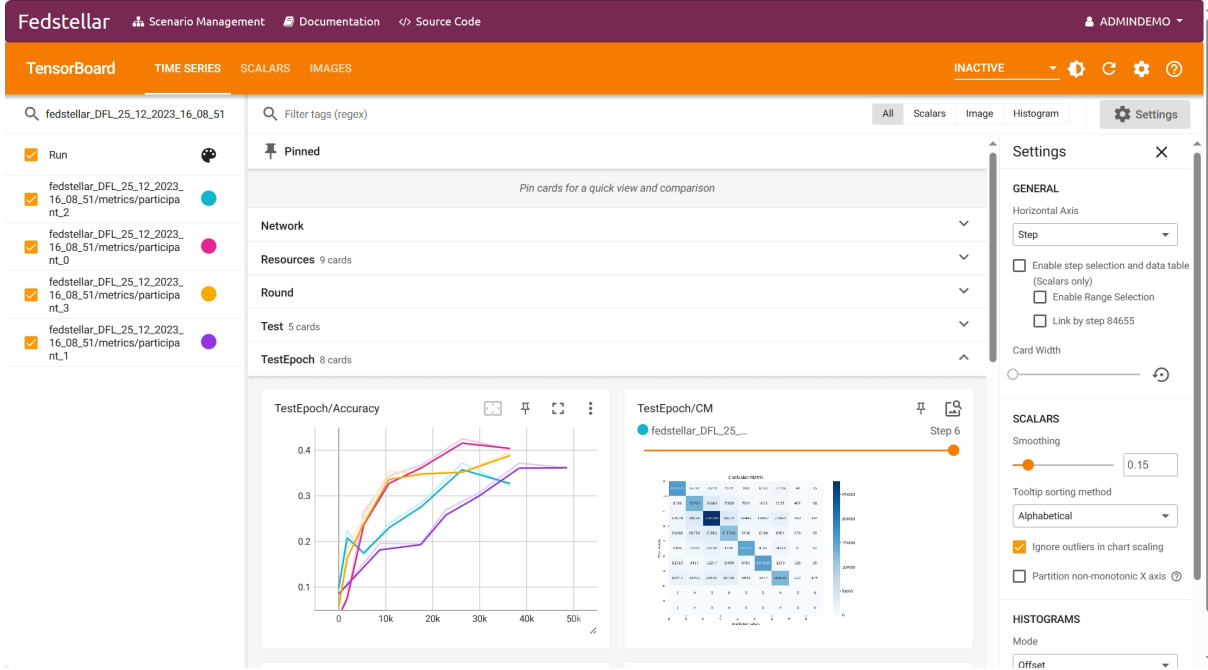


Figure 2.5: Fedstellar Real-time Monitoring Metrics

2. **Controller:** This component orchestrates the operations across the platform, ensuring that tasks are efficiently managed and coordinated.
3. **Core:** Deployed on each participating device, the core module handles the essential tasks of model training and communication between devices, forming the backbone of the platform's functionality.

Overall, Fedstellar offers a robust framework for federated learning, enabling effective and streamlined management of complex, distributed learning environments.

2.3 Environmental Sustainability

In DFL systems, where machine learning models are trained across a multitude of distributed devices, understanding and mitigating the environmental impact of these operations is crucial. The dispersed nature of DFL inherently involves multiple computing nodes, each contributing to the overall energy draw and heat output. To effectively manage and minimize the environmental footprint associated with such systems, it is essential to delve deeply into specific metrics that provide insight into the energy dynamics and emissions of these operations.

The following are conceptions related to environmental sustainability and the metrics used in this work to calculate the energy consumption and carbon emissions of model training in a DFL system.

2.3.1 Energy Consumption

In DFL, the energy consumption associated with model training encompasses the total energy required to operate machine learning models across numerous distributed devices. This includes not only the direct electrical consumption by all devices engaged in the training process but also the energy utilized by auxiliary systems necessary for operation, such as cooling mechanisms. Typically, the direct electrical consumption during model training includes the energy needed for both processing and communication: processing energy refers to the electricity used by devices when executing computational tasks, while communication energy pertains to the electricity required for transmitting data between devices. The key components are as follows:

1. **Computational Energy:** This is the primary type of energy consumption within DFL systems, encompassing the electricity consumed by CPUs, GPUs, or other processing units as they carry out data processing tasks. The level of energy used is intimately linked to the devices' performance capabilities, efficiency, and the complexity of the tasks they execute.
2. **Communication Energy:** Owing to the decentralized characteristics of DFL, there is a need for frequent synchronization of model parameters across various nodes, which incurs additional energy consumption. This energy usage is influenced by the configuration of the network, as well as the frequency and efficiency of the data transmission.
3. **System Auxiliary Energy:** Beyond the energy directly used for processing and communication, additional power is consumed by auxiliary devices necessary for maintaining system operations, including cooling systems, power management, and data storage solutions.

2.3.2 Carbon Emissions and Carbon Intensity

1. **Carbon Emissions:** Carbon emissions primarily refer to the carbon dioxide gas released during the energy consumption process, especially when the electricity used is derived from fossil fuels. In DFL systems, since model training involves extensive data processing and communication, devices consume a significant amount of electricity. If this electricity originates from fossil fuels, it directly results in CO₂ emissions.
2. **Carbon Intensity:** Carbon Intensity is a measure that quantifies the amount of carbon dioxide emissions produced per unit of energy generated or consumed, particularly in electricity production. Specifically, it is usually expressed in terms of the mass of carbon dioxide released per kilowatt-hour (kWh) of electricity generated, typically measured in grams or kilograms of CO₂ per kWh. For example, suppose a power plant emits 900 grams of carbon dioxide for every kilowatt-hour of electricity it generates while burning fossil fuels. In that case, the carbon intensity of that

power plant would be 900 grams CO₂/kWh. This metric not only reflects the efficiency of energy production but also indicates the level of environmental cleanliness. In the field of electricity production, different energy technologies, such as coal, natural gas, wind, and solar, have varying carbon intensity values, directly impacting their environmental effects.

2.3.3 Renewable Energy Ratio

1. **Renewable Energy:** Renewable energy refers to energy resources obtained from natural processes that are continually replenished and not depleted. These sources of energy can be constantly renewed. The main types include [17]:

- *Solar Energy:* Derived from the sun's light and heat, which can be converted into electricity using solar panels.
- *Wind Energy:* Uses wind turbines to convert the kinetic energy of the wind into electricity.
- *Hydraulic Energy:* Including hydroelectric and tidal energy, utilizes the flow of water (such as rivers and tides) to generate electricity.
- *Geothermal Energy:* Harnesses the heat from within the Earth to produce heat or electricity.
- *Biomass Energy:* Organic materials from plants and animals can be converted into energy, such as biofuels.

These energy sources are considered environmentally friendly because they typically produce fewer greenhouse gases during energy production, causing relatively less environmental damage.

2. **Non-renewable Energy:** Non-renewable energy refers to energy resources that cannot be replenished or restored within a human lifetime. Once these resources are depleted, they cannot be naturally restored over a short period. The main types include [17]:

- *Fossil Fuels:* Such as coal, oil, and natural gas. These fuels are formed from the remains of ancient plants and animals buried underground for millions of years under high pressure and temperature.
- *Nuclear Energy:* Although there is a significant stock of nuclear fuels like uranium, they turn into waste after use, requiring long-term management and disposal.

The use of non-renewable energy sources often accompanies the release of significant amounts of carbon dioxide and other greenhouse gases during production, which can have long-term negative effects on the environment and human health.

3. **Renewable Energy Ratio:** The Renewable Energy Ratio (RER) refers to the proportion of electricity that a DFL system obtains from renewable sources (such as

solar, wind, hydro, and biomass energy) compared to its total electricity consumption within a specific time period. This ratio is typically expressed as a percentage, and the formula to calculate it is as follows:

$$\text{RER} = \left(\frac{\text{Electricity Consumed from Renewable Sources}}{\text{Total Electricity Consumption}} \right) \times 100\% \quad (2.1)$$

This metric indicates the extent to which a DFL system utilizes environmentally friendly energy sources to power its operations.

2.3.4 Power Usage Effectiveness

Power Usage Effectiveness (PUE) is a ratio that measures the total amount of energy consumed by a computing facility relative to the energy used directly for computational activities. It is calculated as the total facility energy divided by the IT equipment energy. PUE is calculated using the formula:

$$\text{PUE} = \frac{\text{Total Facility Energy}}{\text{IT Equipment Energy}} \quad (2.2)$$

PUE is crucial for evaluating the efficiency of energy use within computing infrastructures, particularly those involved in DFL systems. A PUE value of 1.0 is ideal, indicating that all the energy consumed is used for computing purposes only, with no waste on cooling, lighting, or other non-computational functions. Reducing the PUE close to this ideal value means that less energy is wasted, which directly contributes to lower operational costs and reduced carbon emissions, especially significant when scaled across the dispersed environments typical of DFL. Efforts to optimize PUE in DFL can lead to more sustainable operations by minimizing the overall energy footprint.

2.3.5 Thermal Design Power

Thermal Design Power (TDP) represents the maximum amount of heat generated by a computer chip (CPU or GPU) under normal operating conditions, which the cooling system is expected to dissipate. It is expressed in watts and is a vital specification used in designing and testing thermal and cooling equipment.

In the process of training models within DFL systems, accurately measuring the power consumption of CPUs in real-time poses significant challenges. Due to the dynamic nature of DFL, where multiple devices across different nodes are engaged simultaneously, capturing precise energy usage data from CPUs during model training can be particularly complex. To address this issue, an approximate method is employed in this work, utilizing the TDP multiplied by the CPU utilization rate.

This approximation method leverages the TDP value, which represents the maximum amount of heat generated under typical operating conditions, as a baseline for maximum

potential energy consumption. By adjusting this value according to the CPU utilization rate, a metric indicating the percentage of time the CPU is actively processing during model training sessions, we can derive a more context-specific estimate of energy usage.

This methodology not only simplifies the measurement process but also provides a scalable solution for monitoring energy consumption across the varied and distributed environments typical of DFL systems. In practice, this means that when a CPU is running at full capacity, its energy consumption is assumed to be close to its TDP rating. Conversely, if the CPU utilization is at 50%, the energy consumption would be approximated as half of the TDP. This approach provides a practical framework for estimating power usage without the need for intricate and potentially disruptive measurement technologies.

Chapter 3

Related Work

This chapter provides a comprehensive review of the literature concerning the carbon footprint associated with traditional ML, CFL, and DFL. It then progresses to a detailed review of the two most significant and relevant papers in the field. In conclusion, this chapter outlines the key insights derived from the extensive literature review. These insights lay the groundwork and serve as the fundamental inspiration for the methodology and approach adopted in this thesis.

3.1 Energy Consumption and Carbon Footprint in Traditional ML, CFL, and DFL

The following Table 3.1 is a summary of the research findings on the environmental sustainability aspect of ML, CFL, and DFL. The initial four articles focus on quantitatively measuring energy usage and carbon emissions in ML and CFL. The fifth article offers a qualitative examination of the environmental impact of CFL. Lastly, the sixth article delves into the calculation and assessment of energy consumption and carbon footprints in ML, CFL, and DFL.

As shown in Table 3.1, [18] offers the first-ever systematic study of the carbon footprint of CFL. It proposes a rigorous model to quantify the carbon footprint, hence facilitating the investigation of the relationship between CFL design and carbon emissions. Besides, it compares the carbon footprint of CFL to traditional ML. [19] provides a methodology for calculating energy consumption and carbon emissions in ML, and proposes best practices, namely: increasing transparency in data centers, selecting the most environmentally friendly and efficient processors for machine learning to train on, and developing more efficient machine learning models. If best practices are adopted across the machine learning landscape, the amount of energy used and the total carbon emissions generated by training will be reduced. [20] presents a generalized methodology for calculating the carbon footprint across various CFL configurations and carries out experiments on actual CFL hardware, exploring different settings, models, strategies, and tasks. [21] suggests that the location of the servers used for training, the energy grid they utilize, the

duration of the training process, and even the brand and model of the hardware used for training significantly impact the carbon emissions produced. To approximate these emissions, this paper introduces a machine learning emissions calculator, which aids in better understanding the environmental impact of training ML models. [14] expands the trustworthiness evaluation framework for CFL by introducing a new sustainability pillar, thoroughly addressing all seven key requirements defined by the European Commission’s High-Level Expert Group on AI (AI-HLEG). The newly added sustainability pillar focuses on assessing the environmental impact of FL systems by qualitatively measuring indicators such as hardware efficiency, federation complexity, and the carbon intensity of the energy grid. [22] introduces a new framework for analyzing energy consumption and carbon emissions in ML, CFL, and DFL environments. It quantifies the energy footprints and carbon dioxide equivalent emissions associated with both standard FL methods and consensus-based fully decentralized approaches. This paper also explores optimal thresholds and operational parameters that facilitate environmentally friendly FL designs, thus enhancing their sustainability assessments.

Table 3.1: Comparison of Related Work Regarding Energy Consumption and Carbon Footprint in ML, CFL, and DFL

Work	Training Method	Measurement Method	Main Idea
[18] 2020	ML, CFL	Quantifying	Quantify the carbon footprint of CFL. Compare the carbon footprint of CFL and ML.
[19] 2022	ML	Quantifying	Estimate the energy and carbon footprint of ML. Energy consumption is not skyrocketing if follow best practices.
[20] 2020	CFL	Quantifying	Introduce a generalized methodology to compute the carbon footprint of many CFL settings. Conduct experiments on real CFL hardware under different settings, models, strategies, and tasks.
[21] 2019	ML	Quantifying	Present a tool, Machine Learning Emissions Calculator, to approximate carbon emissions.
[14] 2023	CFL	Qualitative	Qualitative assessment of the environmental impact of CFL systems. Incorporate three main aspects: hardware efficiency, federation complexity, and the carbon intensity.
[22] 2022	ML, CFL, DFL	Quantifying	Propose frameworks for the calculation of energy consumption and carbon footprints in ML, CFL, and DFL.

3.2 Detailed Review of Previous Works

This section conducts a detailed analysis of the papers most relevant to this thesis, specifically the last two papers [14] and [22] listed in the Table 3.1.

3.2.1 Related Work [14]

Unlike other articles that focus on quantitative frameworks for measuring energy consumption and carbon emissions, this article employs a relative scoring approach that evaluates emissions from three distinct perspectives: the carbon intensity of the energy grid used, the efficiency of the underlying hardware, and the complexity of the federation. The specific metrics for these three dimensions are detailed in Figure 3.1.

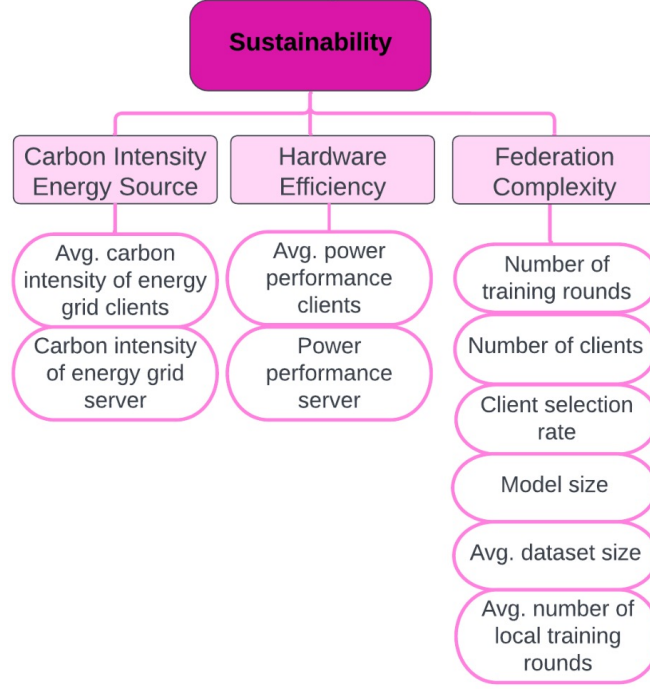


Figure 3.1: Detailed Metrics for Three Dimensions: Carbon Intensity Energy Source, Hardware Efficiency, and Federation Complexity [14]

1. **Carbon Intensity Energy Source:** The carbon intensity of electricity is different in different parts of the world and depends on the energy mix that is used to produce electricity. Key metrics include:
 - Average carbon intensity of energy grid clients: This measures the average carbon intensity associated with the energy used by clients within the grid.
 - Carbon intensity of the energy grid server: This metric focuses specifically on the carbon intensity of the server's energy source.
2. **Hardware Efficiency:** Divide the Mark through the TDP to evaluate the efficiency of the underlying hardware in terms of computing power per unit of power consumed:
 - Average power performance of clients: Evaluates the power efficiency metrics across all clients.
 - Power performance of the server: Assesses the power efficiency of the server hardware.

3. **Federation Complexity:** The complexity and size of the federation significantly influence the energy consumption and, consequently, the emissions generated. Generally, increased complexity and size of the model, along with a higher number of participants, lead to greater energy usage:

- Number of training rounds: Indicates the total number of training rounds undertaken.
- Number of clients: Reflects how many clients are involved in the federation.
- Client selection rate: Measures the frequency or criteria with which clients are selected for training.
- Model size: Considers the overall size of the machine learning model being utilized.
- Average dataset size: Looks at the average size of datasets used by the clients.
- Average number of local training rounds: Represents the average number of training rounds performed locally by clients.

This work evaluates and assigns scores to each of the aforementioned metrics, subsequently summing them to derive a comprehensive score that relatively reflects the environmental sustainability aspect. This approach allows for a nuanced assessment of each component's impact on the overall sustainability of the system, ensuring a balanced and reasoned evaluation of environmental performance.

3.2.2 Related Work [22]

The paper [22] presents a new framework for quantificationally assessing energy consumption and carbon emissions across ML, CFL, and DFL settings. In the DFL setting, the main formulas for energy consumption are as follows:

$$E_{\text{DFL}}(n) = E_{\text{DFL}}^{(L)}(n) + E_{\text{DFL}}^{(C)}(n) \quad (3.1)$$

$$E_{\text{DFL}}^{(L)}(n) = \sum_{t=1}^n \left(\sum_{k \in N_t} E_k^{(C)} + \sum_{k \notin N_t} E_{k,\text{sleep}}^{(C)} \right) \quad (3.2)$$

$$E_{\text{DFL}}^{(C)}(n) = \sum_{t=1}^n \sum_{k \in N_t} \sum_{h \in N_{k,t}} b(W_{k,t}) \cdot E_{k,h}^{(T)} \quad (3.3)$$

Definitions in formulas:

- n : The number of learning rounds.
- k : Client k .

- $E_k^{(C)}$: Energy consumption of client k during local training per learning round.
- $E_{k,\text{sleep}}^{(C)}$: Energy consumption of inactive clients per round.
- N_t : The subset of all clients k that are actively involved in the computation and communication processes during a specific learning round.
- $N_{k,t}$: All the neighboring clients to which client k sends or receives data during the communication phase of the learning round.
- $W_{k,t}$: The weight updates or gradients that are being shared among the clients as part of the federated learning algorithm.
- $b(W_{k,t})$: Converts the data size from a higher-level representation (like model parameters or gradients) into a quantifiable metric (bits) that can be used to calculate the communication energy.
- $E_{k,h}^{(T)}$: The energy consumed for transmission of the 1-bit parameter over the link (k, h) .
- $\sum_{h \in N_{k,t}} b(W_{k,t}) \cdot E_{k,h}^{(T)}$: The total energy spent by client k to diffuse the local model parameters to selected neighbors h in the set N_t at round t .

According to Equation 3.1, the total energy consumption is comprised of the energy used for learning (local training) and the energy expended for communication. It presents a method for calculating actual energy consumption and providing significant insights for this thesis.

3.3 Summary and Insights

3.3.1 Findings and Limitations

After reviewing the relevant literature, several key findings and limitations have emerged:

1. The majority of studies utilize a quantitative approach to evaluate energy consumption and carbon emissions. Nonetheless, the number of studies that specifically address these metrics in DFL is limited, with the bulk of research focusing on traditional ML and CFL.
2. While [14] efficiently scores each defined metric to reflect aspects of environmental sustainability, this work encounters certain limitations:
 - It is challenging to determine the precise weight for each metric, which is critical for accurate assessment.
 - The relative scoring approach hinders direct comparisons with carbon emissions from other systems, which could provide broader environmental insights.

3. Although [22] introduces a method to calculate actual energy consumption and carbon emissions, this study presents specific drawbacks:
 - It includes energy used in learning and communication processes. However, the learning energy consumption solely accounts for the energy used during training phases and neglects the energy consumed during the aggregation process.
 - The study lacks a specific formula for calculating the energy consumed during the learning process (both training and aggregation), which limits the precision and applicability of its findings.

3.3.2 Insights

Building on the insights from the reviewed articles, this thesis makes significant advancements in the understanding and assessment of the environmental impacts associated with DFL systems. Here are the core insights and enhancements derived:

1. **Quantitative Approach for Environmental Impact:** Inspired by the systematic quantitative methods discussed, this thesis adopts a refined approach to quantitatively evaluate energy consumption and carbon emissions in DFL systems. This methodology not only extends previous models but also integrates precise metrics to offer more reliable and measurable insights into sustainability.
2. **Inclusion of All Relevant Learning Phases:** Addressing gaps identified in the existing literature [22], this work comprehensively covers all crucial phases of DFL. This includes training, aggregation, and communication, ensuring a thorough evaluation of each component's energy impact.
3. **Enhanced Methodology for Calculation:** Drawing on the limitations noted in the previous study [22], such as the lack of specific formulas for aggregation energy consumption, this thesis develops detailed methodologies for calculating energy consumption and carbon emissions across different DFL processes.

Through these improvements, this thesis aims to provide a more holistic and detailed calculation framework for assessing the environmental sustainability of DFL systems, thereby contributing to more informed and effective green computing practices in environmental sustainability.

Chapter 4

Architecture

This chapter delves into the detailed methodologies for calculating energy consumption and carbon emissions within DFL environments. It systematically breaks down the analysis into three critical phases of DFL operations: local training, aggregation, and communication. Each phase is examined for its energy consumption and carbon emissions, providing a comprehensive insight into the environmental impact of DFL systems.

In DFL, the division of energy consumption and carbon emission calculations into local training, aggregation, and communication is essential due to the distinct characteristics of each phase:

1. **Training Energy Consumption and Carbon Emissions:** This phase involves the local computation by DFL clients, where models are trained on distributed nodes using local data. This is the most computationally intensive phase, often resulting in significant energy use and associated emissions.
2. **Aggregation Energy Consumption and Carbon Emissions:** Aggregation occurs when trained models from multiple clients are combined to update the global model. This phase involves less computation but significant data transmission across the network, impacting the system's overall energy and carbon footprint.
3. **Communication Energy Consumption and Carbon Emissions:** Communication refers to exchanging information among clients in fully decentralized setups. This phase is critical as it involves data transmission over potentially long distances and may contribute substantially to the network's energy consumption and emissions.

Figure 4.1 presents the comprehensive architecture used to calculate energy consumption and carbon emissions across training, aggregation, and communication phases within DFL environments. To gain a clear and structured understanding of the calculation processes involved in DFL, the following discussion provides a detailed analysis of the energy consumption and carbon emissions framework, as shown in the architectural diagram. Section 4.1 and subsequent sections carefully examine each component of the framework, detailing the energy consumption and carbon emissions during the training, aggregation, and communication phases.

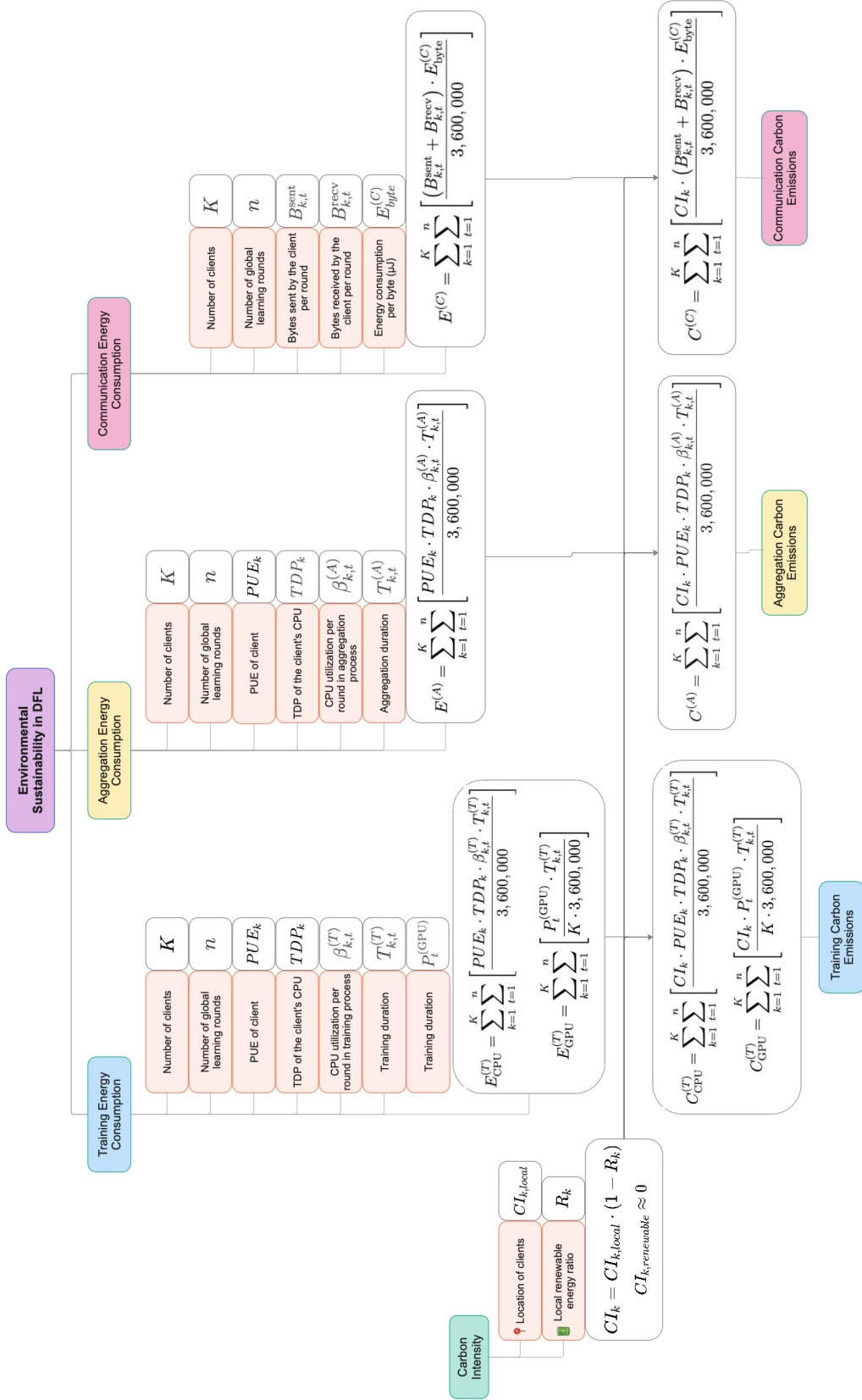


Figure 4.1: Architecture of Environmental Sustainability Module in DFL

4.1 Calculation of Energy Consumption

4.1.1 Training Energy Consumption

The calculation of energy consumption for local training in a node varies based on the type of accelerator used, such as CPUs or GPUs. During the actual training process, nodes may choose to train using only CPUs or a combination of CPUs and GPUs. The total energy consumption for local training is simply the sum of the energy consumed by both CPUs and GPUs (If GPUs are not in use, they consume zero energy). The formulas for calculating the energy consumed by CPUs and GPUs training are provided in 5.1 and 5.2, respectively and the variables in these formulas are defined as follows:

$$E_{\text{CPU}}^{(T)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{PUE_k \cdot TDP_k \cdot \beta_{k,t}^{(T)} \cdot T_{k,t}^{(T)}}{3,600,000} \right] \quad (4.1)$$

$$E_{\text{GPU}}^{(T)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{P_t^{(\text{GPU})} \cdot T_{k,t}^{(T)}}{K \cdot 3,600,000} \right] \quad (4.2)$$

- K : Number of nodes.
- n : Number of global learning rounds.
- PUE_k : Power Usage Effectiveness of each node k , which measures the efficiency of power usage by the node.
- TDP_k : Thermal Design Power of each node k , in watts (W). It indicates the maximum amount of heat generated under the worst-case operational conditions.
- $\beta_{k,t}^{(T)}$: CPU utilization rate in training process of each node k in each round t . It reflects how much of the CPU's capacity is used.
- $T_{k,t}^{(T)}$: Duration of local training for each node k in each round t , in seconds (s).
- $P_t^{(\text{GPU})}$: Power consumption of the GPU in each round t , in watts (W). This assumes a single GPU's power draw is shared across nodes when they are performing parallel computations.
- The factor K in the denominator distributes the total GPU power consumption equally among all nodes, assuming equal workload distribution among them.
- The denominator 3,600,000 converts the result from watts-seconds to kilowatt-hours, making the energy consumption units more readable and comparable to typical energy measurements.

Formula 5.1 calculates the total energy consumption of all nodes across all training rounds using CPU resources. For each node k and each training round t , the energy consumption for a single node during a single training round is calculated using the expression $PUE_k \cdot TDP_k \cdot \beta_{k,t}^{(T)} \cdot T_{k,t}^{(T)}$, where PUE_k measures the power usage effectiveness of the node, TDP_k is the thermal design power of the node's CPU, $\beta_{k,t}^{(T)}$ is the CPU utilization rate indicating the active level of the CPU during the round, and $T_{k,t}^{(T)}$ is the duration of the training round. The resultant energy consumption is normalized to KWh by dividing by 3,600,000.

Due to the difficulty of precisely measuring the actual power consumption of each CPU at every moment without specialized equipment, this approach uses TDP and CPU utilization to estimate CPU energy consumption. It provides a reasonably accurate estimate of energy use, allowing for energy consumption monitoring and management at a lower cost without specialized hardware support. Although this method may not be as precise as direct measurement, it is sufficient for analyzing energy consumption trends, estimating costs, and managing energy efficiency in large-scale deployments.

Formula 5.2 calculates the total energy consumption by all nodes using the GPU across all training rounds. For each training round t and each node k , the GPU energy consumption is computed using the expression $\frac{P_t^{(GPU)} \cdot T_{k,t}^{(T)}}{K}$, where $P_t^{(GPU)}$ represents the power consumption of the GPU during that round, and $T_{k,t}^{(T)}$ is the duration of training for that node. Due to the use of only one graphics card for cost considerations, all nodes fundamentally share the GPU resources fairly. This division of $P_t^{(GPU)}$ by the number of nodes K ensures that the calculated energy consumption is an approximate value that reflects the shared use of GPU resources among all nodes. The result is then converted from joules to kWh by dividing by 3,600,000 to provide a practical measure of the energy contribution across the network.

4.1.2 Aggregation Energy Consumption

The model aggregation phase, while less computationally intensive than local training, involves significant coordination and data transmission across the network, potentially incurring substantial energy costs. The computation method for aggregating the energy consumption during the model aggregation phase is analogous to the calculation used for CPU energy consumption during training. The formula for calculating the energy consumed by aggregation is provided in 5.3 and the variables in this formula are defined as follows:

$$E^{(A)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{PUE_k \cdot TDP_k \cdot \beta_{k,t}^{(A)} \cdot T_{k,t}^{(A)}}{3,600,000} \right] \quad (4.3)$$

- K : Number of nodes.
- n : Number of global learning rounds.
- PUE_k : Power Usage Effectiveness of each node k .

- TDP_k : Thermal Design Power of each node k , in watts (W).
- $\beta_{k,t}^{(A)}$: CPU utilization rate in aggregation process of each node k in each round t .
- $T_{k,t}^{(A)}$: Duration of aggregation for each node k in each round t , in seconds (s).
- The denominator 3,600,000 converts the result from watts-seconds to kilowatt-hours.

Equation 5.3 quantifies the energy consumption during the aggregation process across all nodes k over n global learning rounds in a DFL system. It calculates the energy consumed by each node for aggregation by multiplying the Power Usage Effectiveness (PUE_k), which gauges the power efficiency, with the Thermal Design Power (TDP_k) in watts, which indicates the maximum heat generation under operational use, the CPU utilization rate ($\beta_{k,t}^{(A)}$) during aggregation, and the duration ($T_{k,t}^{(A)}$) of the aggregation in seconds. The result, expressed in watt-seconds, is then converted to kWh by dividing by 3,600,000.

4.1.3 Communication Energy Consumption

To accurately account for the total energy consumed during DFL, it is essential to consider not only the local computation but also the communication phase. Communication is pivotal in FL as it involves transferring updated models and data between nodes, which can significantly impact the network's energy consumption. In the formula 5.4 for calculating communication energy consumption in a DFL environment, each variable is defined as follows:

$$E^{(C)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{(B_{k,t}^{\text{sent}} + B_{k,t}^{\text{recv}}) \cdot E_{\text{byte}}^{(C)}}{3,600,000} \right] \quad (4.4)$$

- K : Number of nodes.
- n : Number of global learning rounds.
- $B_{k,t}^{\text{sent}}$ and $B_{k,t}^{\text{recv}}$: The total bytes sent and received by node k during round t , respectively. These values quantify the data transmitted and received, contributing to the calculation of communication energy.
- $E_{\text{byte}}^{(C)}$: Energy consumption per byte transmitted, in joule (J). A critical factor that depends on the network technology and the efficiency of the communication protocols used.
- The denominator 3,600,000 converts the result from watts-seconds to kilowatt-hours.

The formula 5.4 quantifies the energy used during the communication phase across all nodes and learning rounds. It calculates this by summing the product of the bytes sent ($B_{k,t}^{\text{sent}}$) and received ($B_{k,t}^{\text{recv}}$) by each node in each round, multiplied by the energy per byte $E_{\text{byte}}^{(C)}$, and then normalizing this value to kilowatt-hours by dividing by 3,600,000.

For $E_{\text{byte}}^{(C)}$, WiFi is chosen as the communication mode between nodes in this thesis. WiFi, a widely recognized wireless networking technology, uses radio waves to provide high-speed internet and network connections, allowing devices such as computers, smartphones, and smart home devices to connect to the internet and communicate with each other without the need for physical connections. Due to the difficulty in measuring or obtaining this energy consumption data directly, this thesis estimates the values by referencing WiFi energy consumption data from within the same technological field.

The paper [23] elaborates on the energy consumption for WiFi wireless network, specifically including a detailed data for power consumption of a WiFi radio at different data production rates and different transmission intervals, as shown in Figure 4.2.

$T_{\text{transmission}}$ (sec)	Data Production Rate (bytes/sec)					
	75	100	150	300	600	1200
	Power Consumption (mW)					
$T_{\text{transmission}} \leq 15$	317.1725	317.34	317.675	318.68	320.69	324.71
30	167.1692	167.3367	167.6717	168.6767	170.6867	174.7067
60	83.83583	84.00333	84.33833	85.34333	87.35333	91.37333
120	42.16917	42.33667	42.67167	43.67667	45.68667	49.70667
300	17.16917	17.33667	17.67167	18.67667	20.68667	24.70667
600	8.835833	9.003333	9.338333	10.34333	12.35333	16.37333
1200	4.669167	4.836667	5.171667	6.176667	8.186667	12.20667

Figure 4.2: Power Consumption of a WiFi Radio at Different Data Production Rates and Different Transmission Intervals [23]

The power consumption (mW) from the table for different transmission times and data production rates is first converted to watts (W). The energy consumption for each scenario is calculated using the formula Energy (J) = Power (W) $\times$ Time (s). Next, based on the transmission time and data production rate, the amount of data transmitted is determined, and the total energy consumption is divided by the amount of data transmitted to obtain the energy consumed per byte. Finally, the energy consumed per byte across all scenarios is averaged, resulting in an average energy consumption of approximately 0.0005506J per byte transmitted. The calculation process is shown in the code excerpt 4.1. This approach provides an estimate of the energy consumption associated with transmitting a single byte in the WiFi environment, giving an insight into the energy cost per byte.

```

1 import pandas as pd
2 import numpy as np

```

```

3
4 data = {
5     "Data Production Rate (bytes/sec)": [75, 100, 150, 300, 600, 1200],
6     "T_transmission <=15": [317.1725, 317.34, 317.675, 318.68, 320.69,
7     324.71],
8     "T_transmission 30": [167.1692, 167.3367, 167.6717, 168.6767,
9     170.6867, 174.7067],
10    "T_transmission 60": [83.83583, 84.00333, 84.33833, 85.34333,
11    87.35333, 91.37333],
12    "T_transmission 120": [42.16917, 42.33667, 42.67167, 43.67667,
13    45.68667, 49.70667],
14    "T_transmission 300": [17.16917, 17.33667, 17.67167, 18.67667,
15    20.68667, 24.70667],
16    "T_transmission 600": [8.835833, 9.003333, 9.338333, 10.34333,
17    12.35333, 16.37333],
18    "T_transmission 1200": [4.669167, 4.836667, 5.171667, 6.176667,
19    8.186667, 12.20667]
20 }
21
22 df = .DataFrame(data)
23 transmission_times = [15, 30, 60, 120, 300, 600, 1200]
24 energy_per_byte = []
25
26 for rate in df["Data Production Rate (bytes/sec)"]:
27     for time in transmission_times:
28         power_mW = df[df["Data Production Rate (bytes/sec)"] == rate].
29         iloc[0, transmission_times.index(time) + 1]
30         power_W = power_mW / 1000
31         energy_J = power_W * time
32         data_transmitted = rate * time
33         energy_per_byte.append(energy_J / data_transmitted)
34
35 average_energy_per_byte = .mean(energy_per_byte)
36 average_energy_per_byte

```

Listing 4.1: Calculation of the Average Energy Consumption per Byte in the WiFi Environment

4.2 Calculation of Carbon Intensity

Calculating carbon intensity is crucial because it quantifies the carbon dioxide emissions produced per unit of energy consumed. Carbon intensity varies according to the geographic location of the computing nodes, as different regions may depend on various energy sources with distinct carbon footprints. The energy grid used by a node can be determined by its location, typically defined by the latitude and longitude, aligning with the energy mix of the country where the node operates.

Besides, carbon intensity also depends on the renewable energy ratio of the place where the clients are located. The higher the renewable energy ratio, the lower the carbon intensity of the region. In the formula 5.5 for calculating carbon intensity for every node, each variable is defined as follows:

$$CI_k = CI_{k,local} \cdot (1 - R_k) \quad (4.5)$$

$$CI_{k,renewable} \approx 0 \quad (4.6)$$

- CI_k : The carbon intensity (g CO₂/kWh) at node k . This value quantifies the carbon dioxide emissions per unit of energy consumed specifically by node k , adjusted for its energy sourcing.
- $CI_{k,local}$: The local carbon intensity (g CO₂/kWh) for node k . This value is typically determined by the average emissions factor of the energy mix in the region where the node is located. It reflects the average amount of CO₂ emitted per unit of energy generated, taking into account the proportion of each type of energy source (fossil fuels, nuclear, renewables, etc.) used in that region.
- R_k : This is the renewable energy ratio for node k . It indicates the percentage of energy consumed by the node that comes from renewable sources. This ratio directly influences the reduction of the carbon intensity; the higher the ratio, the lower the carbon intensity, as renewable sources generally contribute less to carbon dioxide emissions.
- $(1 - R_k)$: This term modifies the local carbon intensity by the proportion of non-renewable energy used. It effectively scales down the local carbon intensity based on how much of the node's energy consumption is offset by renewables.
- $CI_{k,renewable} \approx 0$: This expression states that if a node uses energy entirely sourced from renewable technologies, its carbon intensity is considered to be nearly zero. This reflects the ideal scenario where the energy use does not add carbon dioxide to the atmosphere, thus minimizing the environmental impact.

The formula 5.5 provided, $CI_k = CI_{k,local} \cdot (1 - R_k)$, calculates the adjusted carbon intensity for a node k . Here, $CI_{k,local}$ represents the local carbon intensity based on the regional energy mix, and R_k is the proportion of renewable energy utilized. A higher proportion of renewable energy results in a lower effective carbon intensity, making the operation more environmentally sustainable. The second formula 4.6, $CI_{k,renewable} \approx 0$, indicates that the carbon intensity approaches zero when the energy used is entirely from renewable sources, emphasizing the benefits of sustainable energy use.

4.3 Calculation of Carbon Emissions

Having calculated the energy consumption and determined the carbon intensity, the next step involves translating these figures into actual carbon emissions (g CO₂). The formulas 4.7, 4.8, and 4.9 displayed following enable this calculation by integrating the energy consumption for training, aggregation, and communication with the specific carbon intensity values applicable to each node.

$$C_{\text{CPU}}^{(T)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{CI_k \cdot PUE_k \cdot TDP_k \cdot \beta_{k,t}^{(T)} \cdot T_{k,t}^{(T)}}{3,600,000} \right] \quad (4.7)$$

$$C_{\text{GPU}}^{(T)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{CI_k \cdot P_t^{(\text{GPU})} \cdot T_{k,t}^{(T)}}{K \cdot 3,600,000} \right] \quad (4.8)$$

$$C^{(A)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{CI_k \cdot PUE_k \cdot TDP_k \cdot \beta_{k,t}^{(A)} \cdot T_{k,t}^{(A)}}{3,600,000} \right] \quad (4.9)$$

$$C^{(C)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{CI_k \cdot (B_{k,t}^{\text{sent}} + B_{k,t}^{\text{recv}}) \cdot E_{\text{byte}}^{(C)}}{3,600,000} \right] \quad (4.10)$$

1. Training Carbon Emissions using CPUs: This is captured in formula 4.7, where the total energy consumed by the CPUs across all nodes and rounds is multiplied by the carbon intensity specific to each node, thus providing a measure of the carbon emissions attributable to CPU usage.
2. Training Carbon Emissions using GPU: Similar to CPUs, the GPU energy consumption for all nodes is accounted for in formula 4.8. The power consumed by GPUs during each training round is adjusted by dividing by the number of nodes to distribute the GPU energy usage equally among all participating nodes, then multiplied by the carbon intensity.
3. Aggregation Carbon Emissions: As shown in formula 4.9, this segment calculates the carbon emissions during the model aggregation phase by applying the carbon intensity to the energy consumed for data aggregation across nodes.
4. Communication Carbon Emissions: Lastly, formula 4.10 addresses the emissions from data communication between nodes. It incorporates the amount of data sent and received, multiplies this by the energy per byte, and applies the carbon intensity to determine the emissions resulting from network traffic.

Together, these calculations provide a comprehensive overview of the carbon footprint associated with each phase of DFL, thus offering insights into potential areas for energy optimization and carbon reduction.

Chapter 5

Implementation

In this chapter, the environmental sustainability module from Chapter 4 will be directly developed to calculate energy consumption and carbon emissions in a DFL system. Since this environmental sustainability module is intended for integration into the Fedstellar platform, this chapter first provides an overview of Fedstellar and its functionalities in Section 5.1. Following the introduction of the Fedstellar platform, the chapter covers the implementation of the environmental sustainability module, including the implementation of Frontend and Backend in Sections 5.2, and 5.3. Section 5.3 details the Backend implementation, including code snippets and visualizations for each stage of development.

5.1 Fedstellar Platform

This section provides insights into the design, architecture, and important functionalities of Fedstellar platform used in the environmental sustainability module.

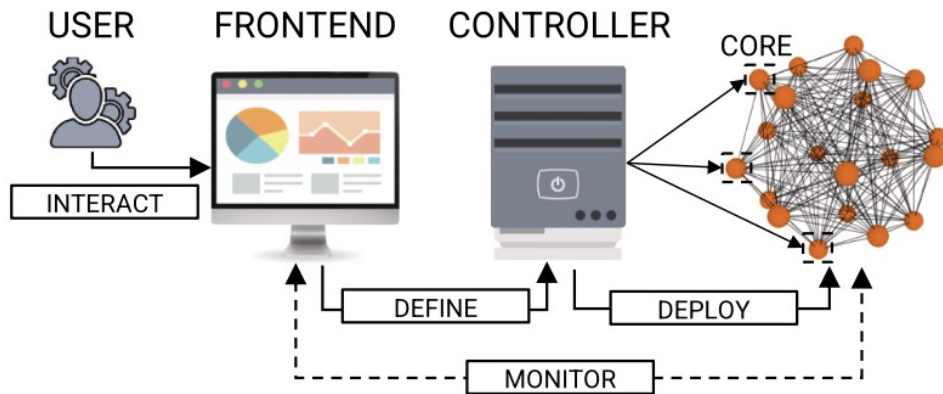


Figure 5.1: Components of the Fedstellar Platform: User, Frontend, Controller, and Core [15]

Fedstellar [24] is a groundbreaking platform that enables the training of FL models in a distributed manner across numerous physical and virtual devices. Fedstellar’s architecture

integrates advanced data processing, efficient training, and communication protocols to deploy FL frameworks on physical and virtual devices. Python, as the foundational language, supports diverse network topologies, efficient communication, and a rich ML library ecosystem. The architecture includes four main components: User, Frontend, Controller, and Core, managing system configuration, training, and performance monitoring. The architecture of the four main components is shown in Figure 5.1.

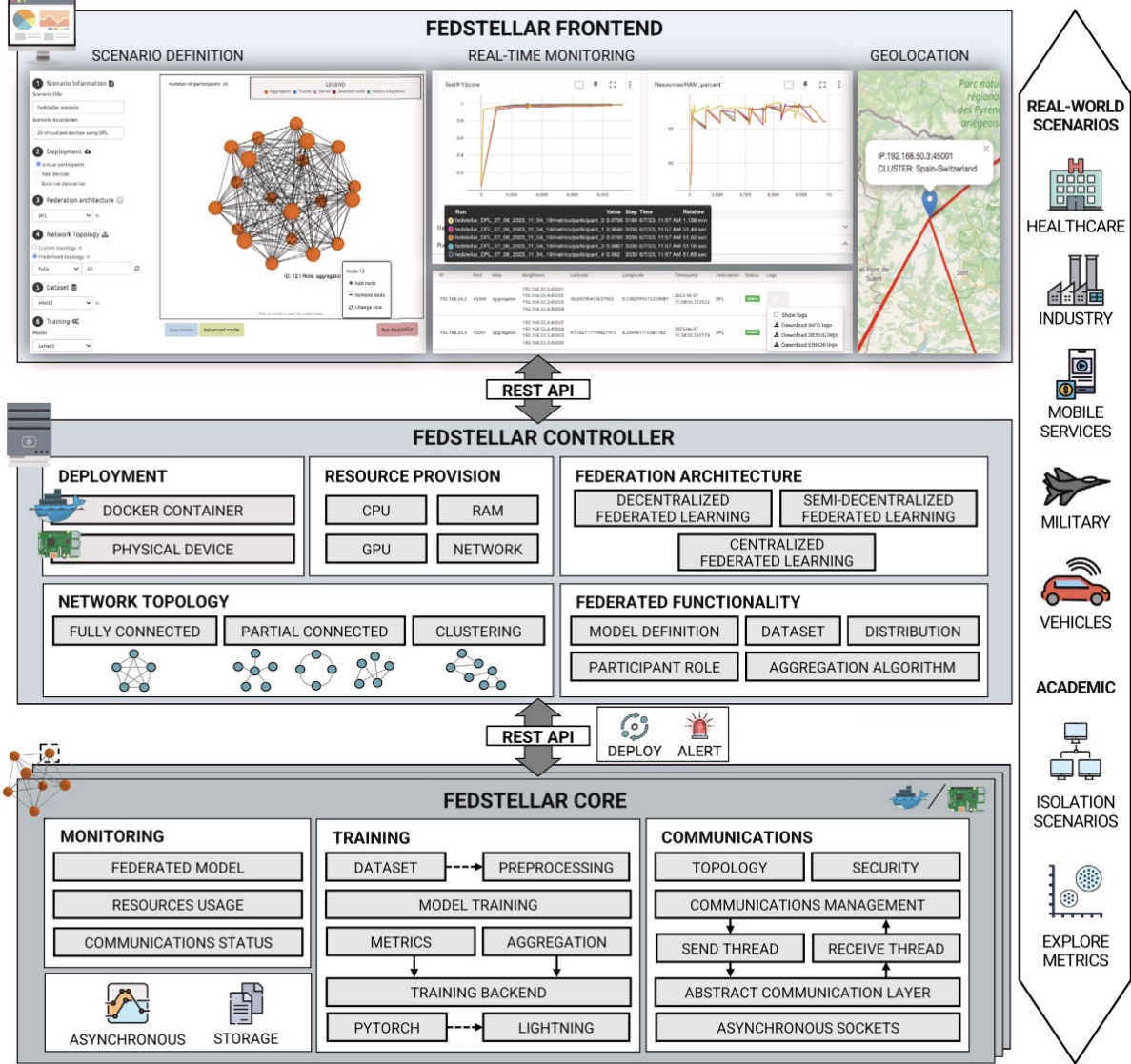


Figure 5.2: Overall Architecture of the Fedstellar platform [15]

1. **Fedstellar Frontend:** It is designed to provide a user-friendly and intuitive experience, serving as the primary point of interaction with Fedstellar. As shown in the first level of Figure 5.2, users can define federated scenarios, monitor progress, and track the physical locations of devices within the topology. Additionally, a REST API allows smooth communication with the controller, offering customizable endpoints for these functionalities.
2. **Fedstellar Controller:** It is an essential component that acts as a bridge, facilitating

seamless interaction between the frontend and core modules deployed in physical or virtualized devices. As illustrated in the second level of Figure 5.2, it encompasses deployment and resource provision, network topology, federation architecture, and federated functionality, each playing a crucial role in orchestrating the FL process. Additionally, a REST API ensures robust connectivity between the controller and each deployed participant.

3. Fedstellar Core: It is a fundamental component of the Fedstellar platform, deployed into physical or virtualized devices by the controller, as shown in the third level of Figure 5.2. This component is responsible for managing most of the computation and communication required for the successful operation of FL. It consists of the monitoring module for establishing the operational environment, the training module for handling the learning process, and the communications module for facilitating interactions between federated participants and the frontend.

5.2 Frontend Implementation

The current version of the Fedstellar platform is designed to support only virtual participants for conducting virtual DFL training sessions. Consequently, the locations of these nodes are assigned either randomly or through manual configuration. To enhance future development and optimize the system, this work introduces frontend interfaces. These interfaces allow users to conveniently set the variables R_k , PUE_k , and $E_{\text{byte}}^{(C)}$, which are critical for the efficient functioning of the DFL framework. The following Subsection 5.2.1 and Subsection 5.2.2 present the frontend interfaces implementation for variables R_k , PUE_k , and $E_{\text{byte}}^{(C)}$, respectively.

5.2.1 Variables R_k and PUE_k

For R_k and PUE_k , the default value of R_k is set to 0%, meaning that the proportion of renewable energy used by the data center where the node is located is 0%. Therefore, its carbon intensity is exactly the same as the carbon intensity of the energy grid in the country where the node is located. The default value of PUE_k is set to 1.0, representing perfect energy efficiency with no additional energy overhead. These default values will also be used in the subsequent experiments.

- R_k : This is the renewable energy ratio for node k . It indicates the percentage of energy consumed by the node that comes from renewable sources. This ratio directly influences the reduction of the carbon intensity; the higher the ratio, the lower the carbon intensity, as renewable sources generally contribute less to carbon dioxide emissions.
- PUE_k : Power Usage Effectiveness of each node k , which measures the efficiency of power usage by the node.

Their specific positions within the overall architecture are depicted in Figure 5.3.

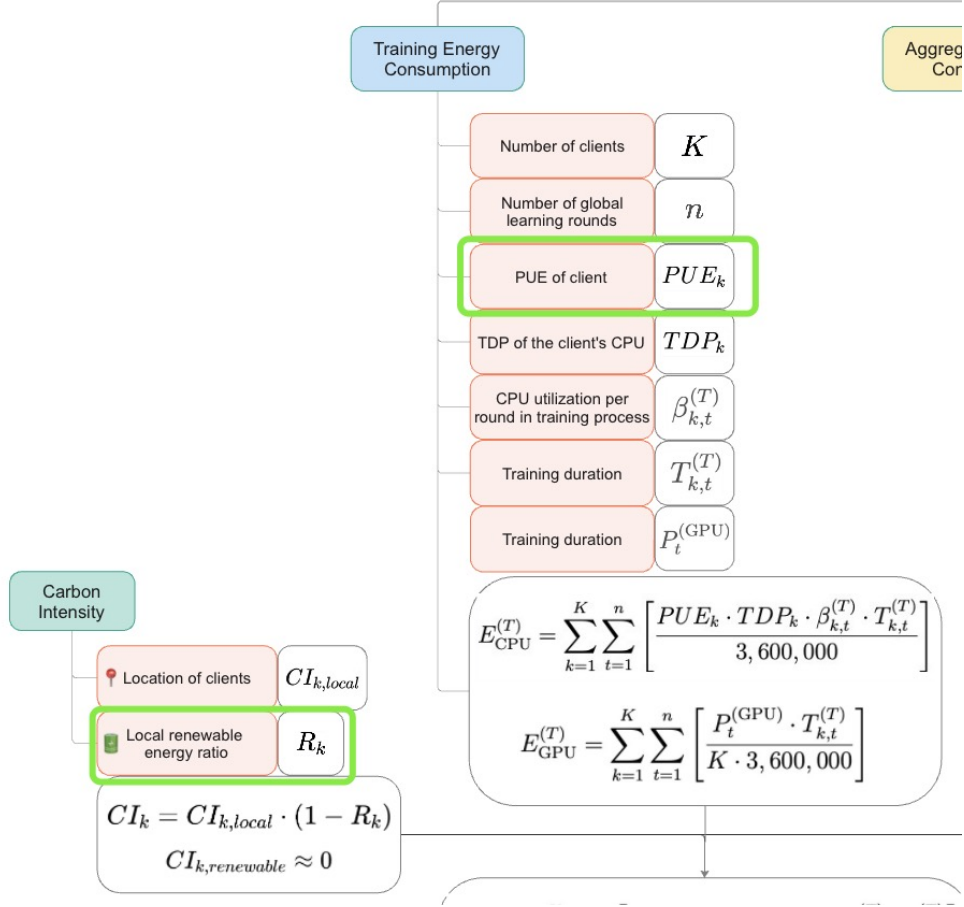


Figure 5.3: The Positions of R_k and PUE_k in the Overall Architecture

The frontend JavaScript code is shown in the following code snippet 5.1. This JavaScript code is primarily used to dynamically generate input forms for participants, allowing the input and management of the virtual DFL participants' variables R_k and PUE_k . It also ensures that these forms are automatically updated when the number of participants changes.

```

1 <script>
2   function generateParticipantSustainabilityInputs() {
3     var participantItems = document.getElementById("participant-
4 sustainability-items");
5     participantItems.innerHTML = "";
6     var n_nodes = document.getElementById("info-participants-number").
7 innerHTML;
8     for (var i = 0; i < n_nodes; i++) {
9       var participantItem = document.createElement("div");
10      participantItem.classList.add("participant-card");
11      participantItem.style.display = "grid";
12      participantItem.style.gridTemplateColumns = "repeat(auto-fit,
13 minmax(100px, 1fr))";

```

```

12     participantItem.style.gridGap = "10px";
13
14     var participantTitle = document.createElement("h5");
15     participantTitle.innerHTML = "Participant " + i;
16     participantItem.appendChild(participantTitle);
17
18     var renewableLabel = document.createElement("h6");
19     renewableLabel.innerHTML = "Percent of local renewable energy";
20     participantItem.appendChild(renewableLabel);
21
22     var renewableInput = document.createElement("input");
23     renewableInput.type = "number";
24     renewableInput.classList.add("form-control");
25     renewableInput.id = "renewable-energy-" + i;
26     renewableInput.placeholder = "Percent of local renewable energy"
    ;
27     renewableInput.min = "0";
28     renewableInput.max = "100";
29     renewableInput.value = "0";
30     renewableInput.style.display = "inline";
31     renewableInput.style.width = "100%";
32     participantItem.appendChild(renewableInput);
33
34     var renewableSmall = document.createElement("small");
35     renewableSmall.classList.add("form-text");
36     renewableSmall.classList.add("text-muted");
37     renewableSmall.innerHTML = "%";
38     renewableSmall.style.lineHeight = "3em";
39     participantItem.appendChild(renewableSmall);
40
41     var pueLabel = document.createElement("h6");
42     pueLabel.innerHTML = "Power usage effectiveness (PUE)";
43     participantItem.appendChild(pueLabel);
44
45     var pueInput = document.createElement("input");
46     pueInput.type = "number";
47     pueInput.classList.add("form-control");
48     pueInput.id = "pue-" + i;
49     pueInput.placeholder = "Power usage effectiveness (PUE)";
50     pueInput.min = "1.0";
51     pueInput.step = "0.01";
52     pueInput.value = "1.0";
53     pueInput.style.display = "inline";
54     pueInput.style.width = "100%";
55     participantItem.appendChild(pueInput);
56
57     participantItems.appendChild(participantItem);
58 }
59 }
60
61 generateParticipantSustainabilityInputs();
62
63 var observer = new MutationObserver(function(mutations) {
64     mutations.forEach(function(mutation) {
65         if (mutation.type === 'childList' || mutation.type === '
characterData' || mutation.type === 'subtree') {

```

```

66         generateParticipantSustainabilityInputs();
67     }
68 });
69 });
70
71 var targetNode = document.getElementById('info-participants-number');
72 var config = { childList: true, characterData: true, subtree: true };
73
74 observer.observe(targetNode, config);
75 </script>

```

Listing 5.1: Frontend Interfaces Implementation for Variables R_k and PUE_k

The HTML part of the code is not shown here. This part of the interface design is consistent with the original Fedstellar user interface style. The frontend interface screenshot is shown as Figure 5.4.

The screenshot shows a web interface titled '16 Environmental Sustainability'. It contains three rows, each representing a participant. Each row has two input fields: 'Percent of local renewable energy' (with a value of 0) and 'Power usage effectiveness (PUE)' (with a value of 1.0). The interface is clean and modern, with a light gray background and white input boxes.

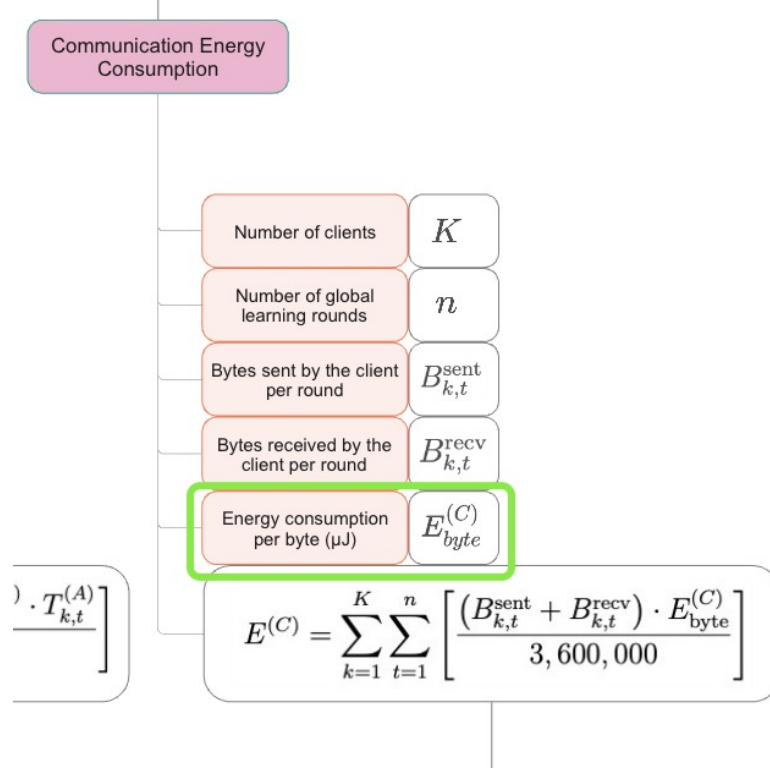
Figure 5.4: The Frontend Interface Screenshot for Variables R_k and PUE_k

5.2.2 Variable $E_{\text{byte}}^{(C)}$

For $E_{\text{byte}}^{(C)}$, WiFi is chosen as the communication mode between nodes in the development and the following evaluation. This decision ensures a stable and widely-used method for data transfer. However, the flexibility of the system allows for the integration of additional communication methods, such as Bluetooth, ZigBee, and Ethernet, in future developments. This capability provides room for a more diverse selection of communication modes, enhancing the adaptability and functionality of the system to meet various needs and preferences.

- $E_{\text{byte}}^{(C)}$: Energy consumption per byte transmitted, in joule (J). A critical factor that depends on the network technology and the efficiency of the communication protocols used.

Its specific position within the overall architecture is depicted in Figure 5.5.

Figure 5.5: The Position of $E_{byte}^{(C)}$ in the Overall Architecture

The frontend HTML code is shown in the following code snippet 5.2. In the existing code, a new “communication-method” dropdown is added to the “Advanced Communications” section, allowing users to select the communication method. This addition includes a step title “Communication method” and a dropdown menu with an option for WiFi, specifying the energy consumption per byte as 0.0005506J. This enhancement provides users with the ability to configure the communication method used between nodes, which is essential for evaluating different communication technologies and their energy consumption impact.

```

1 <!-- Advanced Communications -->
2 <div class="col-md-12 container-shadow tiny grey">
3   <div class="form-group">
4     <h4 class="step-number">Advanced Communications <i class="fa fa-
5       wifi"></i></h4>
6     <div id="docker-options">
7       <h5 class="step-title">Network address</h5>
8       <input type="text" class="form-control" id="network-subnet"
9       style="width: 80%" value="192.168.50.0/24">
10      <h5 class="step-title">Network gateway</h5>
11      <input type="text" class="form-control" id="network-gateway"
12      style="width: 80%" value="192.168.50.1">
13      <h5 class="step-title">Communication method</h5>
14      <select class="form-control" id="communication-method" name=
15      "communication-method" style="display: inline; width: 80%">
16        <option value="0.0005506">WiFi [
17        Energy_consumption_per_byte = 0.0005506J]</option>
18      </select>
19    </div>
20  </div>

```

```

15 </div>
16 </div>

```

Listing 5.2: Frontend Interface Implementation for Variable $E_{\text{byte}}^{(C)}$

This part of the interface design is consistent with the original Fedstellar user interface style. The frontend interface screenshot is shown as Figure 5.6.

11 Advanced Communications

Network address

192.168.50.0/24

Network gateway

192.168.50.1

Communication method

WiFi [Energy_consumption_per_byte = 0.0005506J]

Figure 5.6: The Frontend Interface Screenshot for Variable $E_{\text{byte}}^{(C)}$

5.3 Backend Implementation and Visualization

This section demonstrates how to obtain the remaining key variables in the energy consumption calculation formulas for the three stages: training (using CPU and GPU), aggregation, and communication. It then illustrates how to integrate the energy consumption of these three stages for each node, along with visualizing the energy consumption and carbon emission results for each node. Finally, it consolidates the total energy consumption and carbon emissions across all nodes, representing the final energy consumption and carbon emissions throughout the entire DFL process.

5.3.1 Implementation of Calculations at Three Phases

In the training process (using CPU), the calculation of energy consumption is shown in Formula 5.1:

$$E_{\text{CPU}}^{(T)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{PUE_k \cdot TDP_k \cdot \beta_{k,t}^{(T)} \cdot T_{k,t}^{(T)}}{3,600,000} \right] \quad (5.1)$$

- For TDP_k , it is necessary to first obtain the CPU model of the device used by each node. Subsequently, this model should be matched with its corresponding TDP using a database [25], which can be downloaded in CSV format. The implementation of functions to obtain the TDP of the node's CPU is shown in Code Snippet 5.3.

```

1 def normalize_cpu_string(cpu_string):
2     cpu_string = cpu_string.replace("(R)", "").replace("(TM)", "").
        replace("CPU", "")
3     normalized_string = re.sub(r'[\w\s]', '', cpu_string)
4     normalized_string = normalized_string.lower()
5     normalized_string = ' '.join(normalized_string.split())
6     return normalized_string
7
8 current_dir = os.path.dirname(os.path.abspath(__file__))
9 csv_file_path = os.path.join(current_dir, 'CPU_benchmark_v4.csv')
10 cpu_tdp_df = pd.read_csv(csv_file_path)
11 cpu_tdp_df['cpuName'] = cpu_tdp_df['cpuName'].apply(
        normalize_cpu_string)
12 cpu_tdp_dict = dict(zip(cpu_tdp_df['cpuName'], cpu_tdp_df['TDP']))
13
14 def get_cpu_tdp(cpu_model):
15     cpu_model = normalize_cpu_string(cpu_model)
16     tdp = cpu_tdp_dict.get(cpu_model)
17     logging.info(f"cpu tdp {tdp}")
18     logging.info(f"cpu device {cpu_model}")
19
20     if tdp is None:
21         logging.warning(f"TDP for CPU model '{cpu_model}' not found
        . Using average TDP value.")
22         tdp = 62.38
23     return tdp
24
25 def get_cpu_models():
26     if platform.system() == "Windows":
27         return platform.processor()
28     elif platform.system() == "Linux":
29
30         command = "grep 'model name' /proc/cpuinfo | head -n 1 |
        awk -F ':' '{ print $2 }'"
31         try:
32             result = subprocess.run(command, shell=True, check=True
        , stdout=subprocess.PIPE, text=True)
33             brand=result.stdout.strip()
34         except Exception:
35             brand="Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz"
36         return brand
37
38     elif platform.system() == "Darwin":
39         command = "sysctl -n machdep.cpu.brand_string"
40         model = subprocess.check_output(command, shell=True).strip
        ().decode()
41         return model
42     return "Unknown CPU"
43

```

Listing 5.3: Functions to Obtain the TDP of the Node's CPU

- For $\beta_{k,t}^{(T)}$, it is necessary to allocate specific CPU cores for each node. This helps in better managing CPU resources in a multi-node system, ensuring that each node processes tasks on its assigned cores, thereby facilitating the monitoring of CPU

core utilization for each node. Then the CPU utilization can be obtained using the “cpu_percent()” function from the “psutil” library. The code to allocate specific CPU cores for each node is shown in Code Snippet 5.4.

```

1 n_nodes = self.config.participant["scenario_args"]["n_nodes"]
2 self.process = psutil.Process()
3 devicecount = psutil.cpu_count()//self.config.participant["
  scenario_args"]["n_nodes"]
4 self.cpu_index = [i for i in range(self.idx * devicecount, self.idx
  * devicecount + devicecount)]
5 self.process.cpu_affinity(self.cpu_index)
6

```

Listing 5.4: Code to Allocate CPU Cores for Each Node

The following Figure 5.7 shows an example of the cumulative energy consumption for each node during training process (using CPU).

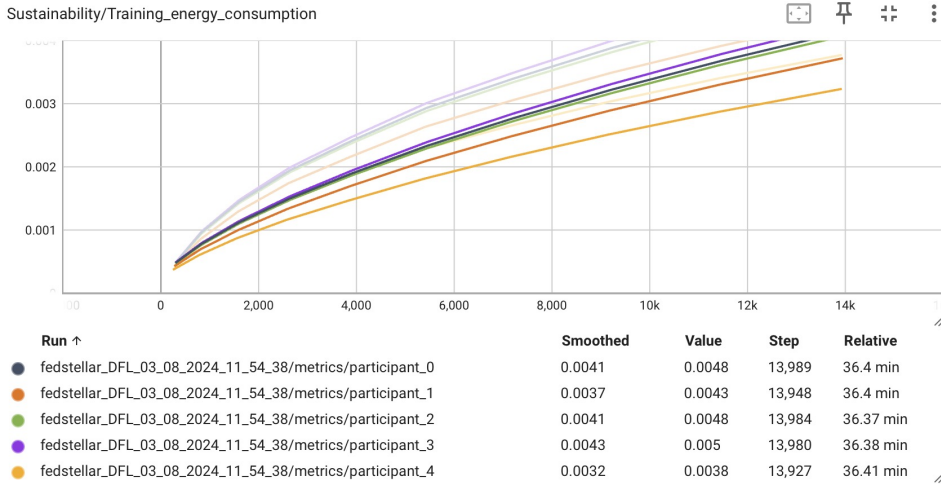


Figure 5.7: Example of Energy Consumption for Each Node in Training (Using CPU)

In the training process (using GPU), the calculation of energy consumption is shown in Formula 5.2:

$$E_{\text{GPU}}^{(T)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{P_t^{(\text{GPU})} \cdot T_{k,t}^{(T)}}{K \cdot 3,600,000} \right] \quad (5.2)$$

- For $P_t^{(\text{GPU})}$, when GPU usage for training is detected, the code iterates over all GPU devices (one GPU device in the following experiments), retrieves and accumulates the power consumption of each device, and then calculates the total energy consumption based on the training time and the number of participating nodes. This value is accumulated into “self.gpu_energy_consumption” to achieve real-time

monitoring and tracking of GPU energy consumption. The code to obtain GPU power and calculate training energy consumption for each node is shown in Code Snippet 5.5.

```

1 if isgpu:
2     gpu_powers=0
3     for i in range(devices):
4         handle = pynvml.nvmlDeviceGetHandleByIndex(i)
5         gpu_power = pynvml.nvmlDeviceGetPowerUsage(handle) / 1000.0
6         gpu_powers+=gpu_power
7     self.gpu_energy_consumption += train_last_time * gpu_powers /
8     self.config.participant["scenario_args"]["n_nodes"] / 3600000

```

Listing 5.5: Code to Obtain GPU Power and Calculate Training Energy Consumption for Each Node

The following Figure 5.8 shows an example of the cumulative energy consumption for each node during training process (using GPU).

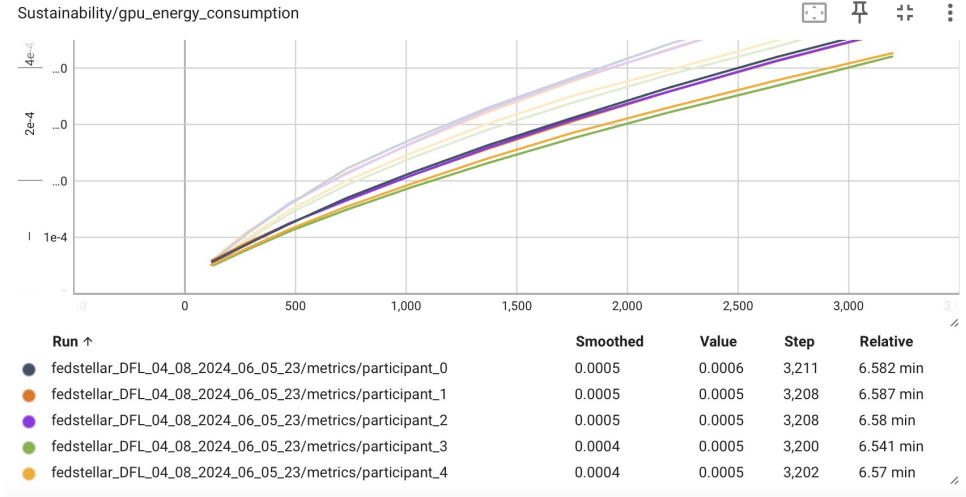


Figure 5.8: Example of Energy Consumption for Each Node in Training (Using GPU)

In the aggregation process, the calculation of energy consumption is shown in Formula 5.3:

$$E^{(A)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{PUE_k \cdot TDP_k \cdot \beta_{k,t}^{(A)} \cdot T_{k,t}^{(A)}}{3,600,000} \right] \quad (5.3)$$

- For $\beta_{k,t}^{(A)}$, the method of obtaining it is similar to that of $\beta_{k,t}^{(T)}$.

The following Figure 5.9 shows an example of the cumulative energy consumption for each node during aggregation process.

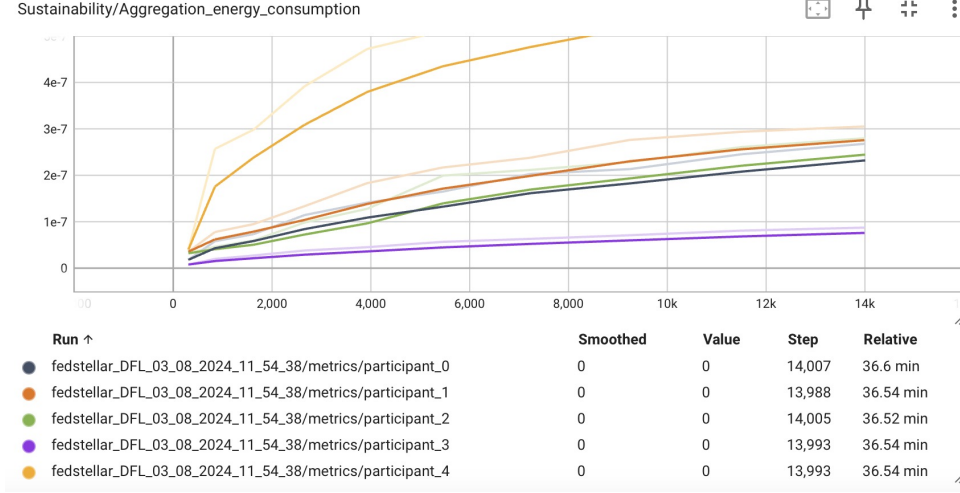


Figure 5.9: Example of Energy Consumption for Each Node in Aggregation

In the communication process, the calculation of energy consumption is shown in Formula 5.4:

$$E^{(C)} = \sum_{k=1}^K \sum_{t=1}^n \left[\frac{(B_{k,t}^{\text{sent}} + B_{k,t}^{\text{recv}}) \cdot E_{\text{byte}}^{(C)}}{3,600,000} \right] \quad (5.4)$$

- For $(B_{k,t}^{\text{sent}} + B_{k,t}^{\text{recv}})$, using the “psutil” library to obtain current network I/O statistics, then extracts the number of bytes sent and received by each node in each round, and calculates the total energy consumption for communication based on the energy consumption per byte. The code to obtain the number of bytes sent and received by each node in each round and calculate communication energy consumption for each node is shown in Code Snippet 5.6.

```

1 net_io_counters = psutil.net_io_counters()
2 bytes_sent = net_io_counters.bytes_sent
3 bytes_recv = net_io_counters.bytes_recv
4 communication_energy_consumption = (bytes_sent + bytes_recv) * self
  .Energy_consumption_per_byte / 3600000
5
```

Listing 5.6: The Code to Obtain the Number of Bytes Sent and Received by Each Node

The following Figure 5.10 shows an example of the cumulative energy consumption for each node during communication process.

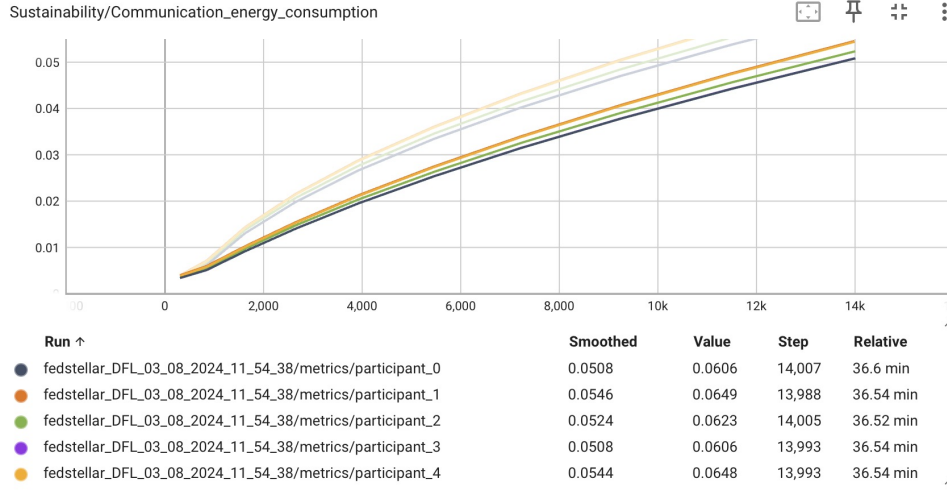


Figure 5.10: Example of Energy Consumption for Each Node in Communication

The previous implementation has already determined the energy consumption of each node during three phases, including training, aggregation, and communication. To calculate the corresponding carbon emissions, the carbon intensity of each node must be obtained. By multiplying the energy consumption by the carbon intensity, the carbon emissions can be calculated. The calculation of carbon intensity for each node is shown in Formula 5.1:

$$CI_k = CI_{k,local} \cdot (1 - R_k) \quad (5.5)$$

- For $CI_{k,local}$, to locate the country based on latitude and longitude data using the Natural Earth dataset [26], and then obtain the corresponding carbon intensity data from the “global_energy_mix.json” file in the CodeCarbon [27] package. The code to obtain the carbon intensity of country where each node located is shown in Code Snippet 5.7.

The code reads a JSON file containing global carbon intensity data and a Shapefile from the specified directory to calculate carbon intensity based on geographical location. It opens and parses the “global_energy_mix.json” file to get the carbon intensity data, then uses “geopandas” to read the “ne_10m_admin_0_countries.shp” file to load country boundary data. The defined “get_carbon_intensity” function locates the country based on the latitude and longitude coordinates and returns the corresponding carbon intensity value.

```

1 with open(os.path.join(current_dir, 'ne_10m_admin_0_countries/
  global_energy_mix.json'), 'r', encoding='utf-8') as file:
2     carbon_intensity_data = json.load(file)
3 world_map = gpd.read_file(os.path.join(current_dir, '
  ne_10m_admin_0_countries/ne_10m_admin_0_countries.shp'))
4
5 def get_carbon_intensity(longitude, latitude):
6     point = Point(longitude, latitude)
7     print(f"longitude{longitude,latitude}")
8     logging.info(f"longitude{longitude,latitude}")

```

```

9     for _, row in world_map.iterrows():
10         try:
11             if row['geometry'].contains(point):
12                 return carbon_intensity_data[row['ISO_A3']]['carbon_intensity"]
13         except Exception as e:
14             print(f"error {e}")
15             logging.error(f"error {e}")
16             return 100
17     return 500
18
19 carbon_intensity = get_carbon_intensity(float(self.config.
    participant["mobility_args"]["longitude"]), float(self.config.
    participant["mobility_args"]["latitude"]))

```

Listing 5.7: The Code to Obtain the Carbon Intensity of Country Where Each Node Located

The following Figures 5.11, 5.12, 5.13, and 5.14 show examples of the cumulative carbon emissions for each node during training, aggregation, and communication processes.

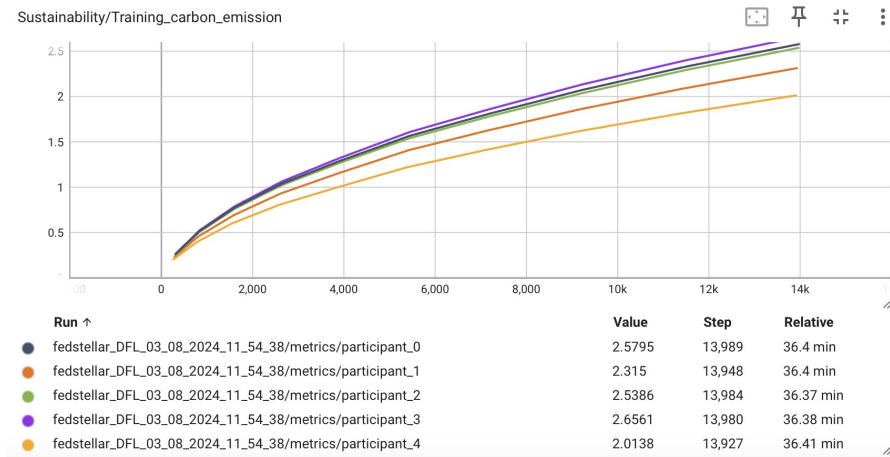


Figure 5.11: Example of Carbon Emissions for Each Node in Training (Using CPU)

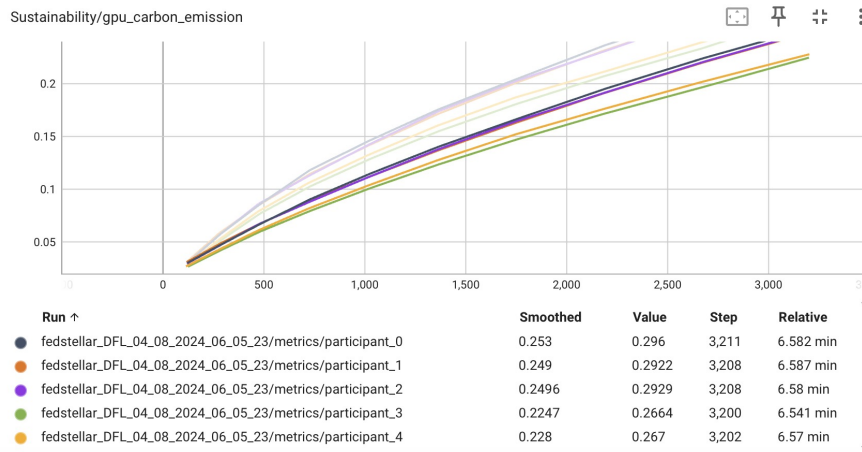


Figure 5.12: Example of Carbon Emissions for Each Node in Training (Using GPU)

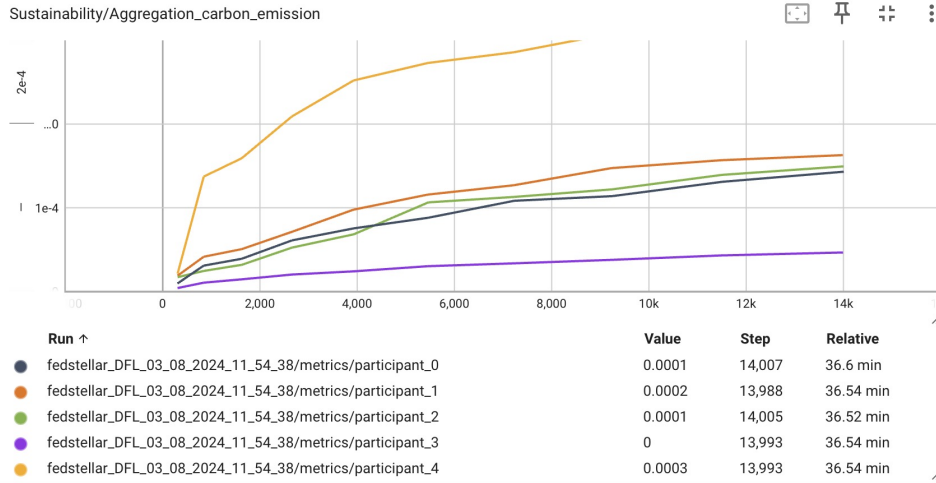


Figure 5.13: Example of Carbon Emissions for Each Node in Aggregation

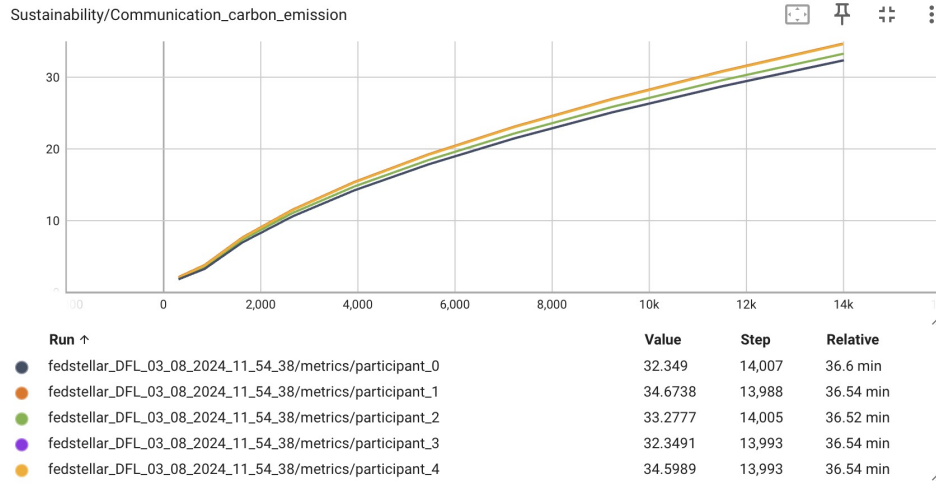


Figure 5.14: Example of Carbon Emissions for Each Node in Communication

5.3.2 Integration of Three Phases

The previous implementation has determined the energy consumption and carbon emissions of each node during three phases. Next, it integrates these three phases for each node. The calculation is shown in Code Snippet 5.8.

```

1 total_energy_consumption = self.train_cpu_energy_consumption + self.
  aggregation_cpu_energy_consumption + communication_energy_consumption
  + self.gpu_energy_consumption
2 total_carbon_emission = total_energy_consumption * carbon_intensivity

```

Listing 5.8: Energy Consumption and Carbon Emissions of Each Node

The following Figure 5.15 and 5.16 show examples of energy consumption and carbon emissions for each node, respectively.

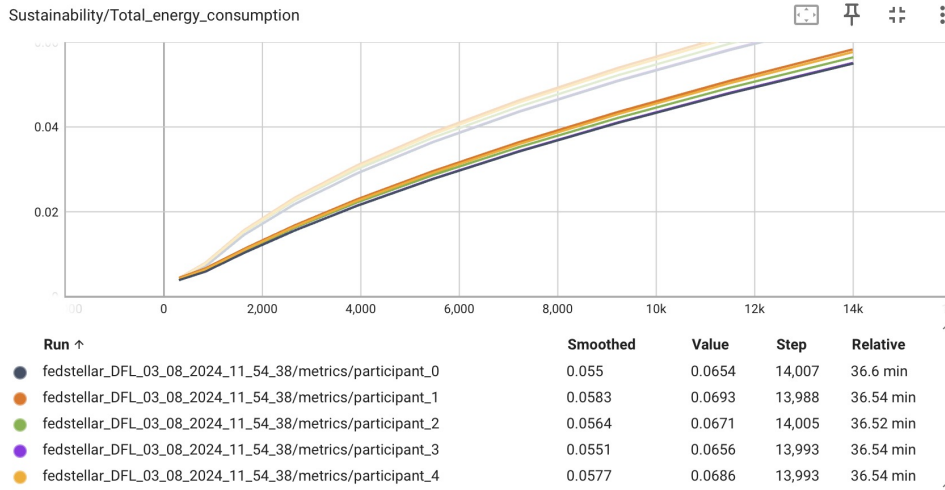


Figure 5.15: Example of Energy Consumption for Each Node

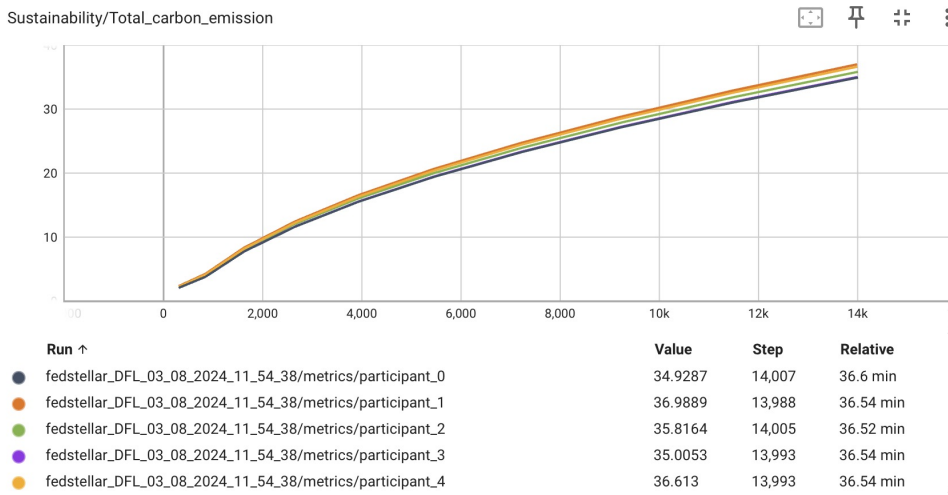


Figure 5.16: Example of Carbon Emissions for Each Node

5.3.3 Integration of All Nodes

Finally, by summing the energy consumption of all individual nodes, the total energy consumption for the entire DFL process is obtained. Likewise, by summing the carbon emissions of all nodes, the total carbon emissions for the entire DFL process are calculated. The functions are shown in Code Snippet 5.9. In the “aggregate_nodes_data” function, it aggregates the total energy consumption and total carbon emissions of all nodes, determines the latest step from either energy or carbon data, and finally logs these aggregated metrics.

```

1 def read_tensorboard_data(log_dir, attribute='Resources/
  total_energy_consumption'):
2     total_energy = 0

```

```

3     overall_latest_step = -1
4
5     for subdir in os.listdir(log_dir):
6         subdir_path = os.path.join(log_dir, subdir)
7         event_acc = EventAccumulator(subdir_path)
8         event_acc.Reload()
9
10        if attribute in event_acc.scalars.Keys():
11            energy_events = event_acc.Scalars(attribute)
12            if energy_events:
13                latest_event = max(energy_events, key=lambda e: e.step)
14                total_energy += latest_event.value
15                if latest_event.step > overall_latest_step:
16                    overall_latest_step = latest_event.step
17
18        return total_energy, overall_latest_step
19
20 def aggregate_nodes_data(self):
21     Total_nodes_energy_consumption, step_energy = read_tensorboard_data(
22         os.path.join(self._root_dir, self._name), "Sustainability/
23         Total_energy_consumption")
24     Total_nodes_carbon_emission, step_carbon = read_tensorboard_data(os.
25         path.join(self._root_dir, self._name), "Sustainability/
26         Total_carbon_emission")
27
28     if step_carbon < step_energy:
29         step = step_energy
30     else:
31         step = step_carbon
32     metrics = {"Sustainability/Total_nodes_energy_consumption":
33         Total_nodes_energy_consumption, "Sustainability/
34         Total_nodes_carbon_emission": Total_nodes_carbon_emission}
35     super().log_metrics(metrics, step)

```

Listing 5.9: Integrate Energy Consumption and Carbon Emissions of All Nodes

The following Figure 5.17 and 5.18 show examples of energy consumption and carbon emissions for each node, respectively.

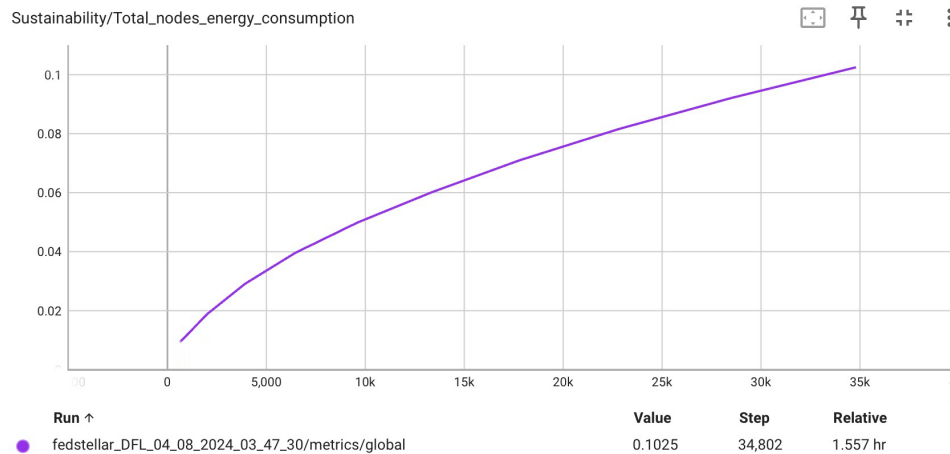


Figure 5.17: Example of Total Energy Consumption

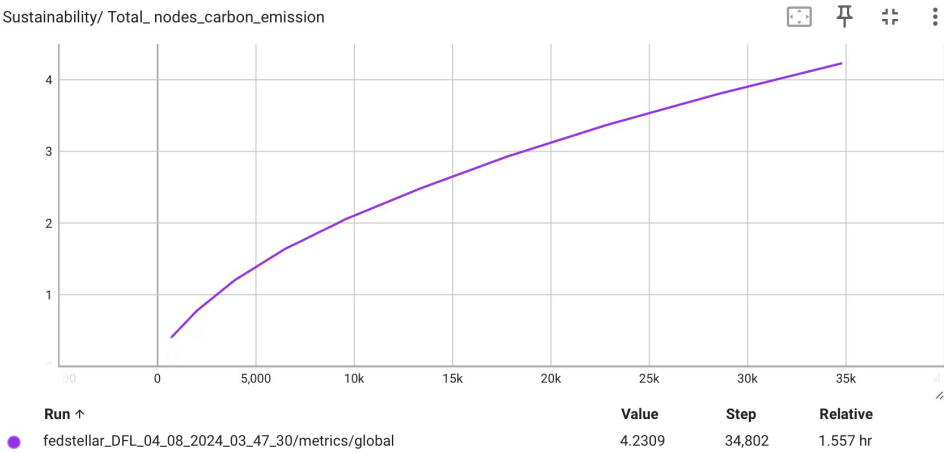


Figure 5.18: Example of Total Carbon Emissions

Chapter 6

Evaluation

This chapter provides a comprehensive evaluation of the energy consumption and carbon emissions associated with various experimental setups in a DFL environment. The evaluation focuses on key parameters and configurations, including federation complexity, the number of nodes, node locations, and accelerator types. By conducting 16 different experiments with diverse combinations of datasets (MNIST, FashionMNIST, CIFAR10) and models (MLP, CNN, fastermobilenet, CNN_V3), this study aims to identify the factors that significantly influence the overall energy efficiency and environmental footprint of the DFL process. The chapter is structured into several sections, except for Section ??, each addressing a specific scenario and its impact on energy consumption and carbon emissions.

Section 6.2 compares energy consumption and carbon emissions during training, aggregation, and communication phases. Section 6.3 examines how different aggregation algorithms and dataset-model combinations affect energy use and emissions. Section 6.4 explores the impact of varying the number of nodes in the federated learning setup. Section 6.5 investigates how the geographical location of nodes influences energy consumption and emissions. Section 6.6 analyzes the differences in energy and emissions when using CPU versus GPU accelerators.

6.1 Experimental Configurations

This evaluation studied the energy consumption and carbon emissions under various parameter conditions in a federated learning setup. A total of 16 experiments were conducted, involving different combinations of datasets such as MNIST, FashionMNIST, and CIFAR10, and models including MLP, CNN, fastermobilenet, and CNN_V3. These experiments aimed to explore the impact of model complexity (including aggregation algorithms, dataset and model combinations), the number of nodes, node locations, and accelerators on the overall energy efficiency and environmental footprint of the federated learning process.

The key parameters of the experimental configurations included both fixed and variable parameters. The 16 experiments used the same fixed parameters, including federation approach, deployment type, topology, dataset type, number of rounds, number of epochs, communication method, renewable energy ratio, and PUE. The fixed parameter settings are shown in Table 6.1.

Table 6.1: Fixed Parameters in Experimental Setup

Parameter	Setting
Federation approach	DFL
Deployment type	Virtual participants
Dataset type	Non-IID
Topology	Fully connected
Communication method	WiFi
Renewable energy ratio	0%
PUE	1.0
Number of round	10
Number of epoch	3

The variable parameters varied across the experiments and included the dataset, model, aggregation algorithm, number of nodes, node locations, and accelerators, as detailed in Table 6.2.

Experiment	Dataset	Model	Aggregation Algorithm	Number of Nodes	Location	Accelerator
1	MNIST	MLP	Krum	5	China	CPU
2	MNIST	MLP	FedAvg	5	China	CPU
3	MNIST	CNN	FedAvg	5	China	CPU
4	FashionMNIST	MLP	FedAvg	5	China	CPU
5	FashionMNIST	CNN	FedAvg	5	China	CPU
6	CIFAR10	fastermobilenet	FedAvg	5	China	CPU
7	CIFAR10	CNN_V3	FedAvg	5	China	CPU
8	MNIST	MLP	FedAvg	3	China	CPU
9	FashionMNIST	MLP	FedAvg	3	China	CPU
10	CIFAR10	fastermobilenet	FedAvg	3	China	CPU
11	MNIST	MLP	FedAvg	5	Switzerland	CPU
12	FashionMNIST	MLP	FedAvg	5	Switzerland	CPU
13	CIFAR10	fastermobilenet	FedAvg	5	Switzerland	CPU
14	MNIST	MLP	FedAvg	5	China	GPU
15	FashionMNIST	MLP	FedAvg	5	China	GPU
16	CIFAR10	fastermobilenet	FedAvg	5	China	GPU

Table 6.2: Variable Parameters for Experiments 1 through 16

The following Table 6.3 shows the estimated model parameter size for each dataset MNIST, FashionMNIST, and CIFAR10. Specifically, for the MNIST and FashionMNIST datasets, MLP and CNN models are selected. The MLP model has a parameter size

of 234.2K, while the CNN model has a parameter size of 6.47M, indicating a significant difference in parameter sizes between these two models. For the CIFAR10 dataset, among the models CNN, CNN_V2, CNN_V3, fastermobilenet, and simplemobilenet, the CNN_V3 and fastermobilenet models are chosen. The CNN_V3 model has a parameter size of 1.39M, which is the largest among these models, while the fastermobilenet model has a parameter size of 9.35K, making it the smallest among these models.

Table 6.3: Estimated Model Parameter Size

Dataset	Model	Estimated Model Parameter Size
MNIST	MLP	234.2K
	CNN	6.47M
FashionMNIST	MLP	234.2K
	CNN	6.47M
CIFAR10	CNN	552.65K
	CNN_V2	1.10M
	CNN_V3	1.39M
	fastermobilenet	9.35K
	simplemobilenet	136.6K

6.2 Scenario 1: Comparison Across Three Phases

This scenario explores the comparison of energy consumption and carbon emissions across three phases: training, aggregation, and communication. For this analysis, experiments 2 and 10 have been selected to assess their energy consumption and carbon emissions in each of the three stages. The specific experimental parameters are detailed in Table 6.4.

Experiment	Dataset	Model	Aggregation Algorithm	Number of Nodes	Location	Accelerator
2	MNIST	MLP	FedAvg	5	China	CPU
10	CIFAR10	fastermobilenet	FedAvg	3	China	CPU

Table 6.4: Details of Experiment 2 and 10

6.2.1 Analysis of Experiment 2 Across Three phases

The following presents two sets of bar charts reflecting energy consumption and carbon emissions in Experiment 2 across three phases: training, aggregation, and communication.

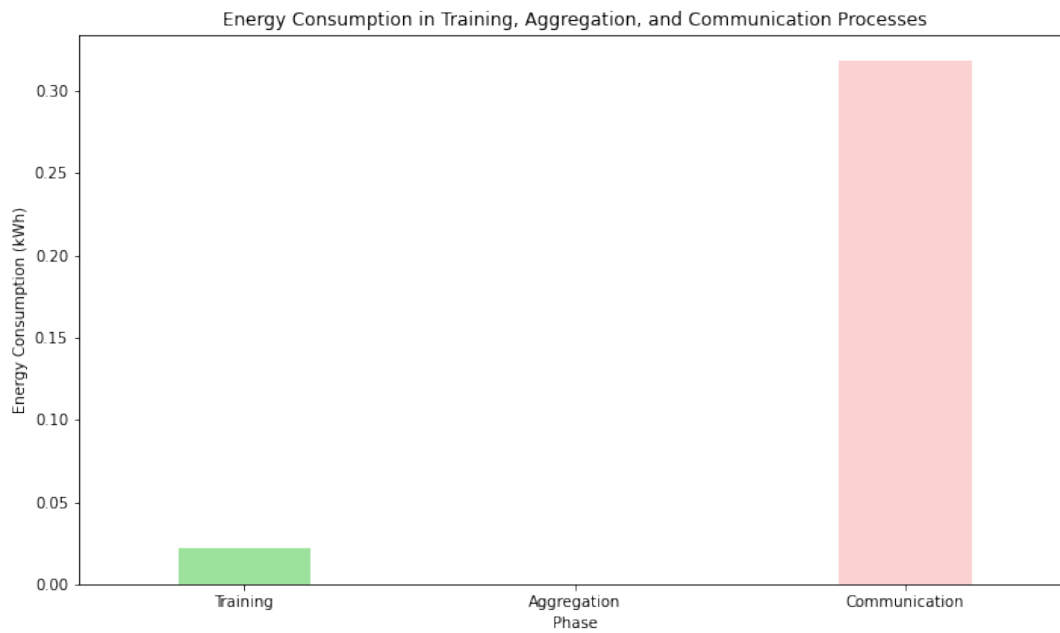


Figure 6.1: Comparison of Energy Consumption Across Three Phases in Experiment 2

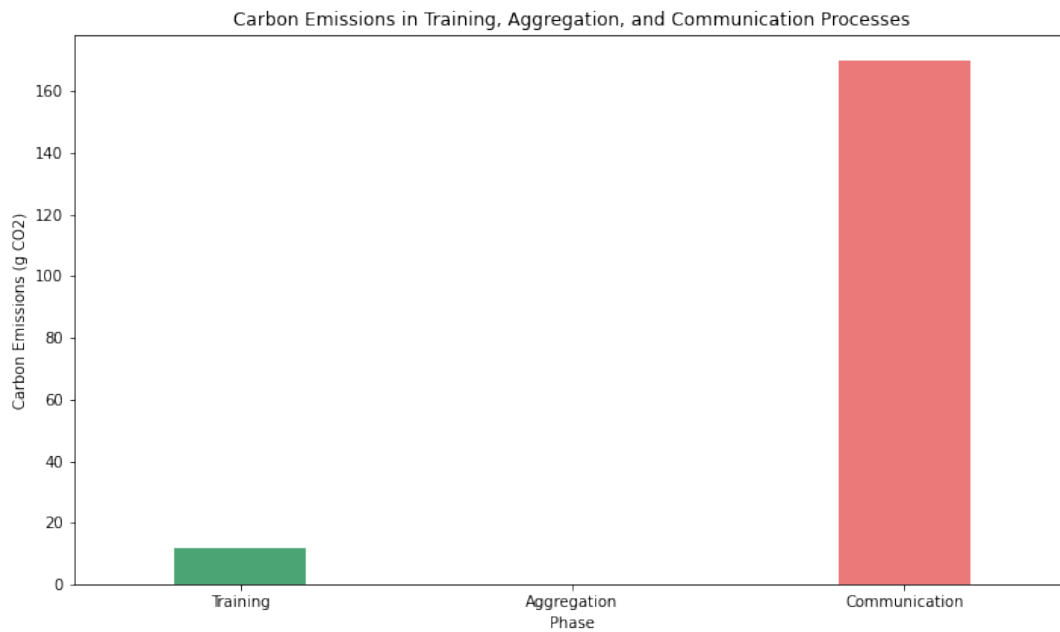


Figure 6.2: Comparison of Carbon Emissions Across Three Phases in Experiment 2

Figure 6.1 shows the energy consumption across three phases (training, aggregation, and communication) in Experiment 2. The energy consumption during the training phase is around 0.02 kWh, which is relatively low. This could be because, although the training phase requires substantial computation, the use of a relatively simple MLP model and the MNIST dataset results in lower energy consumption. The aggregation phase's energy consumption is 0 kWh, indicating almost no energy consumption during this phase,

consistent with the fact that the aggregation phase involves minimal computation and data transmission. The communication phase has an energy consumption of around 0.3 kWh, the highest among the three phases. This suggests that in a Distributed Federated Learning (DFL) environment, frequent data exchanges between nodes, especially with a Non-IID dataset, lead to higher communication load and significant energy consumption.

Figure 6.2 shows the carbon emissions across the three phases in Experiment 2. The training phase's carbon emission is approximately 12 g CO₂, which, despite low energy consumption, is significant due to the high carbon intensity of electricity in China (533.983 g CO₂/kWh). The aggregation phase's carbon emission is approximately 0.001 g CO₂, almost negligible, aligning with its zero energy consumption. The communication phase's carbon emission is around 170 g CO₂, the highest among the three phases. This is because the high energy consumption (around 0.3 kWh) during the communication phase, combined with the high carbon intensity in China, results in substantial carbon emissions.

From the analysis above, it is evident that the communication phase is the main source of energy consumption and carbon emissions in Experiment 2. Although the training phase involves some computational load, its energy consumption and carbon emissions are relatively low due to the simplicity of the model and dataset used. The aggregation phase has negligible energy consumption and carbon emissions due to minimal computation and data transmission. The communication phase, due to frequent data exchanges and high carbon intensity of electricity, emerges as the primary source of energy consumption and carbon emissions.

This scenario underscores the importance of optimizing communication efficiency and reducing communication load in a DFL environment to lower overall energy consumption and carbon emissions. Additionally, selecting electricity sources with lower carbon intensity can effectively reduce carbon emissions. By optimizing communication strategies and using green energy, the environmental sustainability of DFL systems can be significantly enhanced.

6.2.2 Analysis of Experiment 10 Across Three phases

The following presents two sets of bar charts reflecting energy consumption and carbon emissions in Experiment 10 across three phases: training, aggregation, and communication.

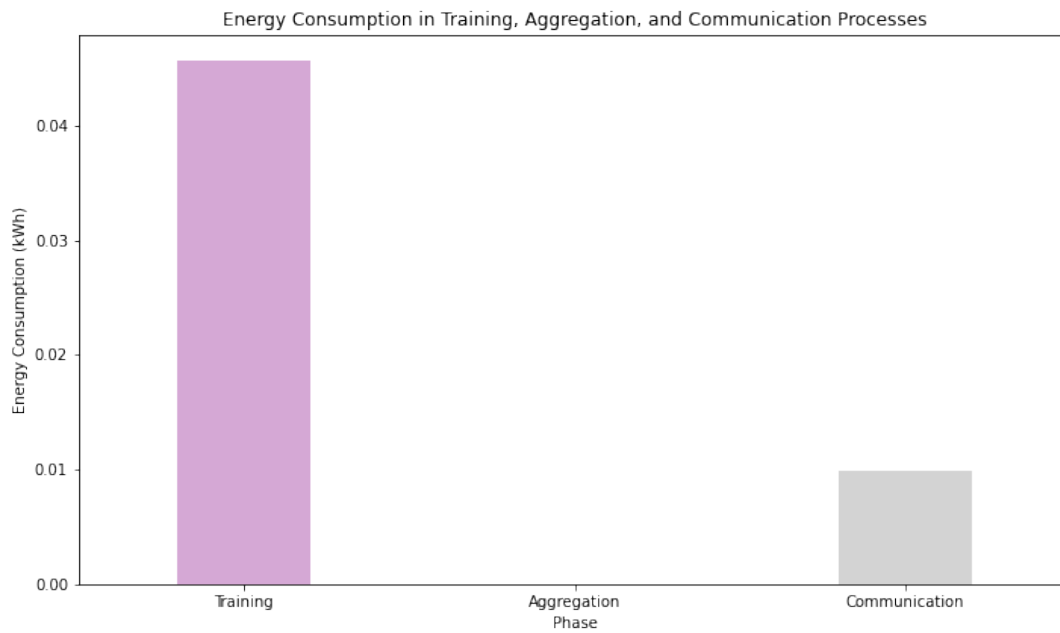


Figure 6.3: Comparison of Energy Consumption Across Three Phases in Experiment 10

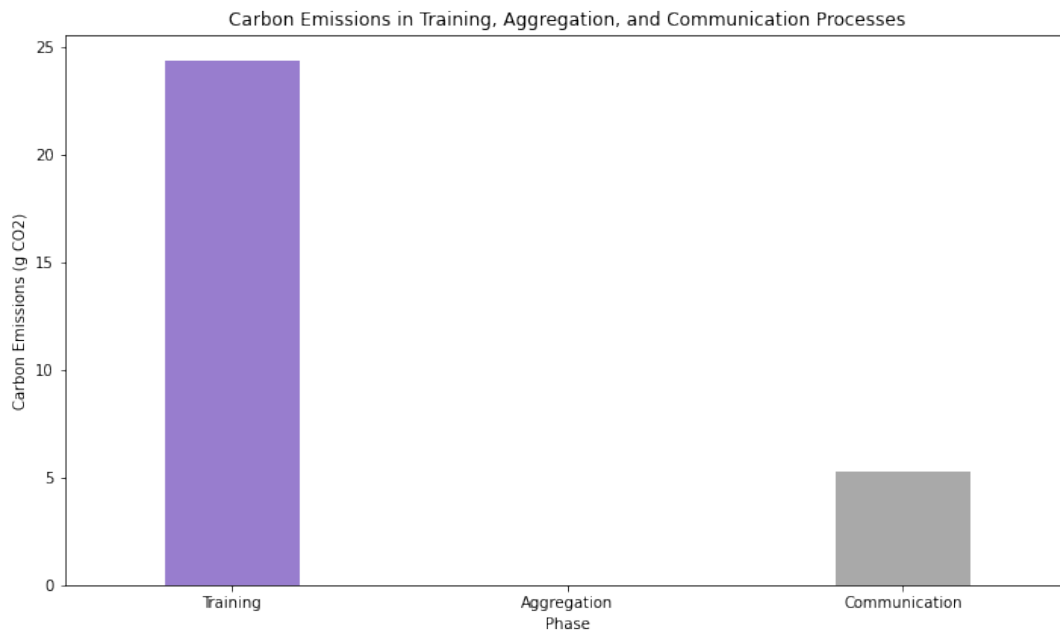


Figure 6.4: Comparison of Carbon Emissions Across Three Phases in Experiment 10

In Experiment 10, the energy consumption and carbon emissions during the training, aggregation, and communication phases using the CIFAR10 dataset and the fastermobilenet model were observed. The results show that the energy consumption and carbon emissions are significantly higher during the training phase compared to the communication and aggregation phases.

First, the energy consumption during the training phase is the highest, approximately 0.04 kWh. This is because the training process involves extensive computational operations, including adjusting and optimizing model parameters. Although the fastermobilenet model has relatively small parameter size (9.35K), processing the CIFAR10 dataset still requires substantial computational resources, leading to significantly higher energy consumption and carbon emissions (about 24 g CO₂) during the training phase compared to other phases.

Second, the energy consumption during the communication phase is about 0.01 kWh, with carbon emissions of approximately 5 g CO₂. The communication phase mainly involves data transmission between nodes. Although frequent data transmissions are required, the amount of data transferred each time is relatively small, resulting in lower overall energy consumption and carbon emissions.

Lastly, the energy consumption and carbon emissions during the aggregation phase are negligible. This phase primarily involves averaging model parameters from different nodes, which is computationally less intensive, leading to minimal energy consumption and carbon emissions.

In summary, the energy consumption and carbon emissions during the training phase in Experiment 10 are significantly higher than during the communication phase. This is mainly due to the higher computational complexity and data processing requirements during training. In contrast, the communication phase primarily involves transmitting model parameters, which has lower computational demands, resulting in lower energy consumption and carbon emissions.

6.3 Scenario 2: Federation Complexity

This scenario focuses on the impact of federation complexity on energy consumption and carbon emissions. It examines how different aspects of federation complexity, such as the choice of aggregation algorithm and the combinations of dataset and model, influence energy consumption and carbon emissions.

6.3.1 Scenario a: Aggregation Algorithm

Scenario a explores the impact of aggregation algorithm on energy consumption and carbon emissions. The specific experimental parameters are shown in Table 6.5. The scenario compares the energy consumption and carbon emissions of the MNIST + MLP combination using two different aggregation algorithms: Krum and FedAvg.

Experiment	Dataset	Model	Aggregation Algorithm	Number of Nodes	Location	Accelerator
1	MNIST	MLP	Krum	5	China	CPU
2	MNIST	MLP	FedAvg	5	China	CPU

Table 6.5: Details of Experiments 1 and 2

The following presents two sets of bar charts reflecting energy consumption and carbon emissions in experiments using different aggregation algorithms (Krum and FedAvg).

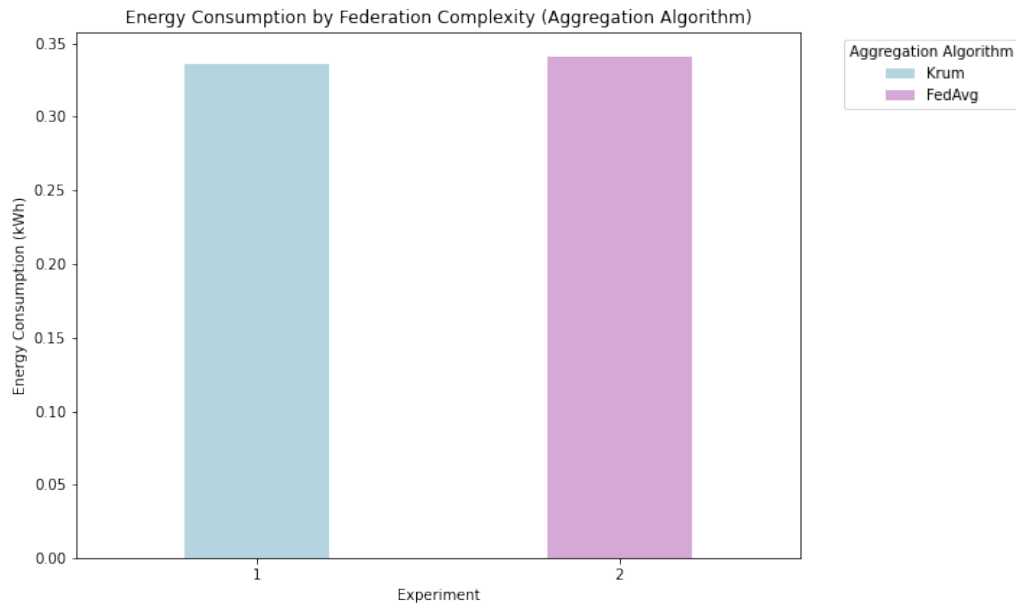


Figure 6.5: Impact of Aggregation Algorithms on Energy Consumption in Experiments 1 and 2

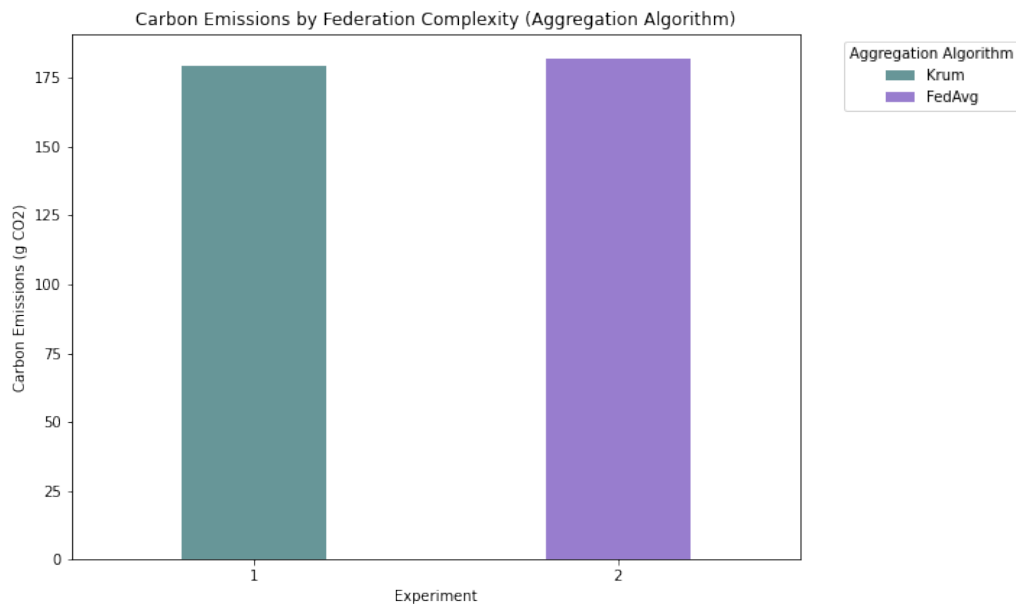


Figure 6.6: Impact of Aggregation Algorithms on Carbon Emissions in Experiments 1 and 2

Figure 6.5 shows that both aggregation algorithms (Krum and FedAvg) have approximately the same energy consumption, around 0.33 kWh. The chart indicates that there is no significant difference in energy consumption between the two aggregation algorithms.

This suggests that under the same experimental conditions (such as the same dataset, model, number of nodes, location, and accelerator), the choice of aggregation algorithm does not significantly impact energy consumption.

Figure 6.6 reveals that the carbon emissions for both algorithms are approximately 175 grams of CO₂. Similarly, this implies that under these specific conditions, the choice of aggregation algorithm does not significantly affect environmental impact, at least in terms of carbon emissions. Given the identical experimental setups, these results suggest that other factors, such as the energy source and hardware efficiency, play a more critical role in determining environmental impact rather than the aggregation algorithm itself.

The similarity in energy consumption and carbon emissions can be attributed to several reasons. Firstly, the experimental setup is identical, including the number of nodes (5), location (China), and accelerator type (CPU). This consistency eliminates differences attributable to these factors, focusing solely on the impact of the aggregation algorithm. Secondly, the use of the same model (MLP) and dataset (MNIST) in both experiments ensures that the computational workload and data characteristics are identical, leading to similar energy consumption and carbon emissions. Additionally, under a non-IID (non-independent and identically distributed) data setting, Krum and FedAvg may exhibit similar computational efficiencies when applied to the MLP model, meaning they use CPU resources almost identically, resulting in the same energy consumption.

Furthermore, the PUE is set to 1.0, indicating that the data center’s energy usage is entirely efficient with no energy wastage. This perfect efficiency results in equal energy consumption for both algorithms. Lastly, the communication method is WiFi, which might introduce some variability in energy consumption. However, the consistent environmental conditions likely minimize any significant differences arising from this factor.

In summary, the results of the charts indicate that under the controlled experimental conditions provided, the choice between Krum and FedAvg does not impact energy consumption or carbon emissions. The findings suggest that factors such as hardware and operational efficiency play a more pivotal role in determining energy and environmental impact in these scenarios, rather than the aggregation algorithm itself.

6.3.2 Scenario b: Dataset and Model

Scenario b explores the impact of different dataset and model combinations on energy consumption and carbon emissions. The specific experimental parameters are shown in Table 6.6. The scenario compares the energy consumption and carbon emissions of six combinations: MNIST + MLP, MNIST + CNN, FashionMNIST + MLP, FashionMNIST + CNN, CIFAR10 + fastermobilenet, and CIFAR10 + CNN_V3.

Experiment	Dataset	Model	Aggregation Algorithm	Number of Nodes	Location	Accelerator
2	MNIST	MLP	FedAvg	5	China	CPU
3	MNIST	CNN	FedAvg	5	China	CPU
4	FashionMNIST	MLP	FedAvg	5	China	CPU
5	FashionMNIST	CNN	FedAvg	5	China	CPU
6	CIFAR10	fastermobilenet	FedAvg	5	China	CPU
7	CIFAR10	CNN_V3	FedAvg	5	China	CPU

Table 6.6: Details of Experiments 2 through 7

The following presents two sets of bar charts reflecting energy consumption and carbon emissions in experiments using different dataset and model combinations.

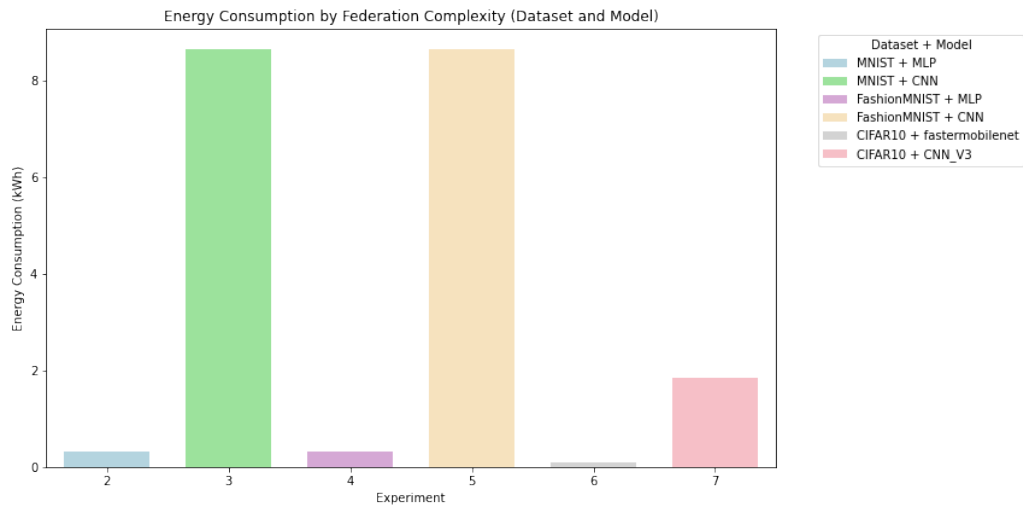


Figure 6.7: Impact of Combinations of Dataset and Model on Energy Consumption in Experiments 2 through 7

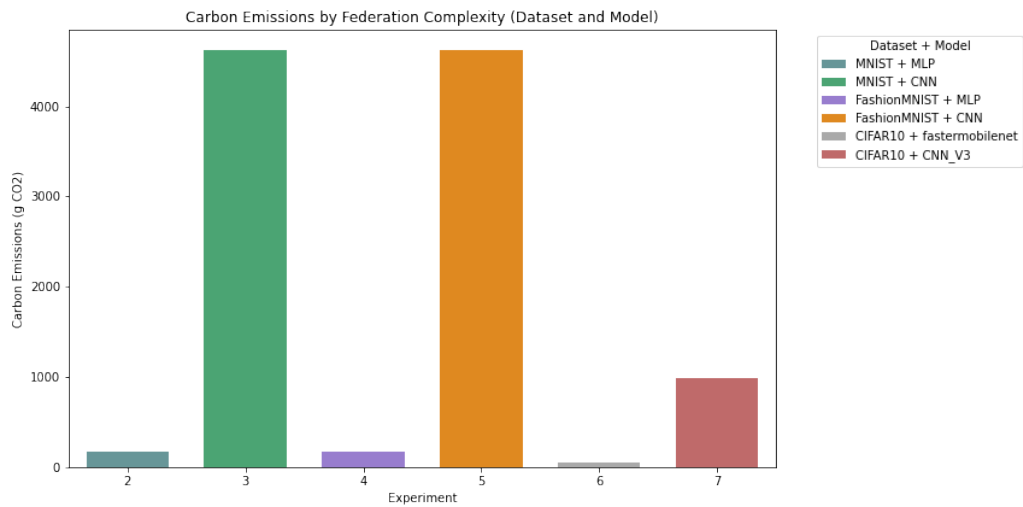


Figure 6.8: Impact of Combinations of Dataset and Model on Carbon Emissions in Experiments 2 through 7

Figure 6.7 shows that the total energy consumption for Experiment 3 (MNIST + CNN) and Experiment 5 (FashionMNIST + CNN) is significantly higher, both around 8.6 kWh, primarily due to their high communication energy consumption. In contrast, the total energy consumption for Experiment 2 and Experiment 4, which use MLP models, is much lower, both around 0.33 kWh. Other models, such as fastermobilenet and CNN_V3 on the CIFAR10 dataset, also have lower energy consumption, approximately 0.1 kWh and 1.8 kWh, respectively.

Figure 6.8 shows that in terms of carbon emissions, the total emissions for Experiment 3 and Experiment 5 are also significantly higher, approximately 4600 g CO₂, which is directly related to their high energy consumption, particularly from communication. In contrast, the carbon emissions for Experiment 2 and Experiment 4 are lower, at around 180 g CO₂. Other models, such as fastermobilenet and CNN_V3, also have lower carbon emissions, at approximately 55 g CO₂ and 990 g CO₂, respectively.

These experimental results indicate that the choice of model and dataset combination significantly impacts energy consumption and carbon emissions. The CNN models chosen for training on the MNIST and FashionMNIST datasets consume more energy and produce more carbon emissions due to their larger number of parameters and higher computational complexity. MLP models, with fewer parameters and simpler computations, have lower energy consumption and carbon emissions. Optimized models like fastermobilenet and CNN_V3, with smaller parameter sizes, show potential for significantly reducing energy consumption and carbon emissions while maintaining performance.

6.4 Scenario 3: Number of Nodes

This scenario explores the impact of the number of nodes on energy consumption and carbon emissions. The specific experimental parameters are shown in Table 6.7. The experiment compares three different combinations: MNIST + MLP, FashionMNIST + MLP, and CIFAR10 + fastermobilenet, and tests them under conditions with 3 nodes and 5 nodes respectively.

Experiment	Dataset	Model	Aggregation Algorithm	Number of Nodes	Location	Accelerator
2	MNIST	MLP	FedAvg	5	China	CPU
8	MNIST	MLP	FedAvg	3	China	CPU
4	FashionMNIST	MLP	FedAvg	5	China	CPU
9	FashionMNIST	MLP	FedAvg	3	China	CPU
6	CIFAR10	fastermobilenet	FedAvg	5	China	CPU
10	CIFAR10	fastermobilenet	FedAvg	3	China	CPU

Table 6.7: Details of Experiments 2,4,6 and 8,9,10

The following presents two sets of bar charts reflecting energy consumption and carbon emissions in experiments using different number of nodes (3 and 5).

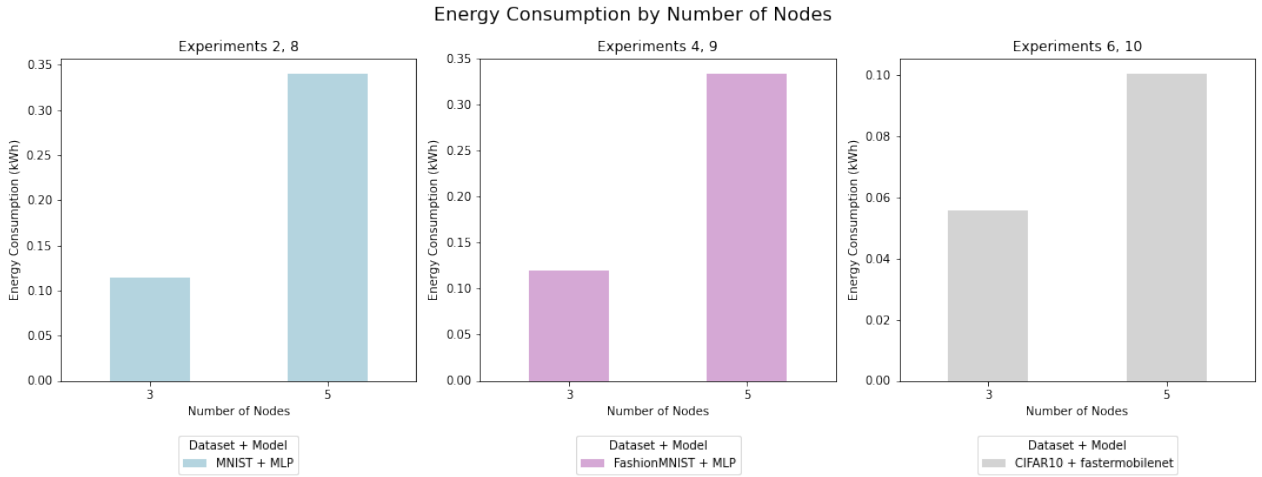


Figure 6.9: Impact of the Number of Nodes on Energy Consumption in Experiments 2 and 8, 4 and 9, and 6 and 10

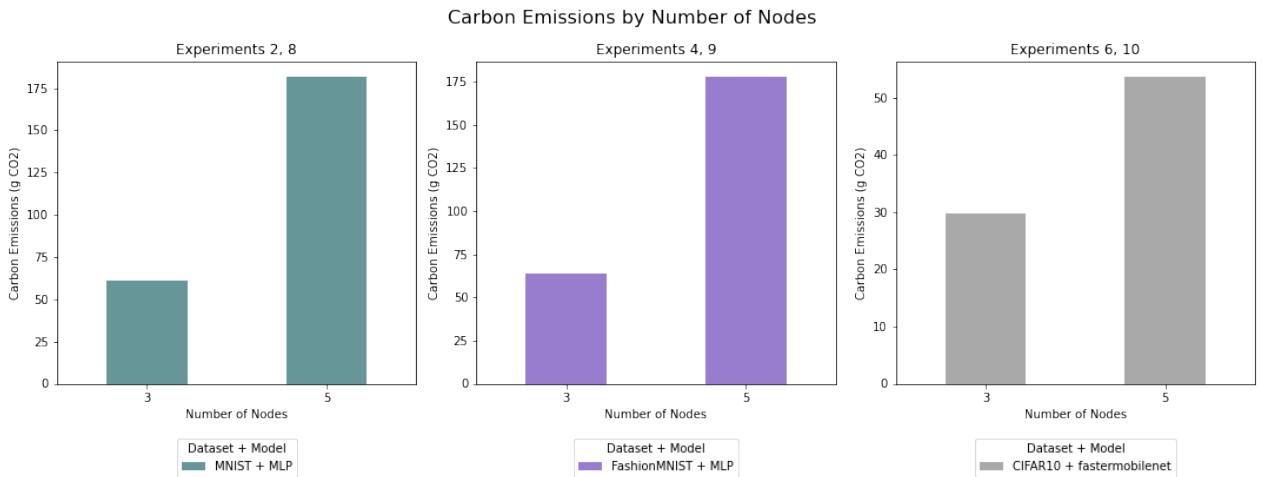


Figure 6.10: Impact of the Number of Nodes on Carbon Emissions in Experiments 2 and 8, 4 and 9, and 6 and 10

Figure 6.9 shows that the number of nodes has a significant impact on energy consumption. As the number of nodes increases from 3 to 5, the energy consumption of all combinations increases significantly. For example, the energy consumption of the MNIST + MLP and FashionMNIST + MLP combinations is approximately 0.34 kWh with 5 nodes, compared to only 0.1 kWh with 3 nodes. Similarly, the energy consumption of the CIFAR10 + fastermobilenet combination increases from approximately 0.05 kWh to 0.1 kWh. This indicates that more nodes require more communication and synchronization, thereby increasing overall energy consumption.

Figure 6.10 shows that the number of nodes also has a very significant impact on carbon emissions. As the number of nodes increases, the carbon emissions of all combinations also increase correspondingly. For instance, the carbon emissions of the MNIST + MLP

and FashionMNIST + MLP combinations are approximately 180 g CO₂ with 5 nodes, compared to only 60 g CO₂ with 3 nodes. Although the carbon emissions of the CIFAR10 + fastermobilenet combination are lower, they also increase from 30 g CO₂ to 53 g CO₂. This further demonstrates that more nodes lead to higher energy consumption, thereby increasing carbon emissions.

6.5 Scenario 4: Location

This scenario investigates the impact of node locations on energy consumption and carbon emissions. The specific experimental parameters are shown in Table 6.8. The experiment compares the energy consumption and carbon emissions of three combinations: MNIST + MLP, FashionMNIST + MLP, and CIFAR10 + fastermobilenet, under the locations of China and Switzerland.

Experiment	Dataset	Model	Aggregation Algorithm	Number of Nodes	Location	Accelerator
2	MNIST	MLP	FedAvg	5	China	CPU
11	MNIST	MLP	FedAvg	5	Switzerland	CPU
4	FashionMNIST	MLP	FedAvg	5	China	CPU
12	FashionMNIST	MLP	FedAvg	5	Switzerland	CPU
6	CIFAR10	fastermobilenet	FedAvg	5	China	CPU
13	CIFAR10	fastermobilenet	FedAvg	5	Switzerland	CPU

Table 6.8: Details of Experiments 2,4,6 and 11,12,13

The following presents two sets of bar charts reflecting energy consumption and carbon emissions in experiments changing locations of nodes (China and Switzerland).

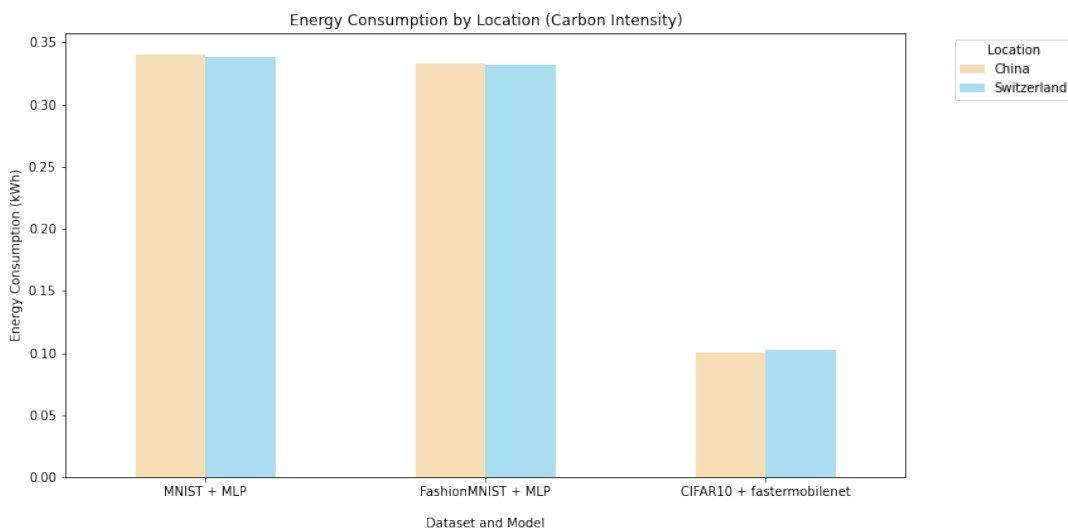


Figure 6.11: Impact of Location on Energy Consumption in Experiments 2 and 11, 4 and 12, and 6 and 13

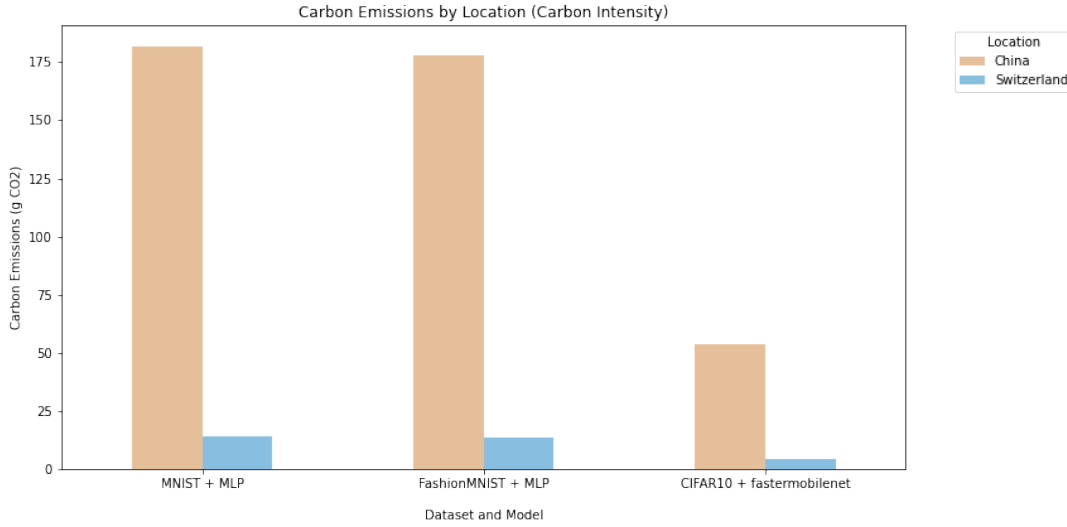


Figure 6.12: Impact of Location on Carbon Emissions in Experiments 2 and 11, 4 and 12, and 6 and 13

Figure 6.11 shows that, from the perspective of energy consumption, the impact of location on the energy consumption of different combinations is minimal. Whether in China or Switzerland, the energy consumption of the MNIST + MLP and FashionMNIST + MLP combinations is approximately 0.33 kWh. The CIFAR10 + fastermobilenet combination consumes about 0.1 kWh in both locations. This indicates that geographical location has an insignificant impact on energy consumption, likely because the experimental conditions and hardware configurations remain consistent.

However, Figure 6.12 shows that, from the perspective of carbon emissions, the impact of location is very significant. The MNIST + MLP and FashionMNIST + MLP combinations have carbon emissions of about 175 g CO₂ in China, compared to about 15 g CO₂ in Switzerland. The CIFAR10 + fastermobilenet combination emits about 50 g CO₂ in China, compared to about 5 g CO₂ in Switzerland. This indicates that, despite similar energy consumption, the carbon intensity of different countries has a significant impact on total carbon emissions.

Overall, the experimental results show that geographical location has a minor impact on energy consumption but a significant impact on carbon emissions. This is due to differences in energy structures and carbon intensities between countries.

6.6 Scenario 5: Accelerator

This scenario explores the impact of accelerator types (CPU and GPU) on energy consumption and carbon emissions. The specific experimental parameters are shown in Table 6.9. Under consistent conditions, the combinations of MNIST + MLP, FashionMNIST + MLP, and CIFAR10 + fastermobilenet are compared using both CPU and GPU.

Experiment	Dataset	Model	Aggregation Algorithm	Number of Nodes	Location	Accelerator
2	MNIST	MLP	FedAvg	5	China	CPU
4	FashionMNIST	MLP	FedAvg	5	China	CPU
6	CIFAR10	fastermobilenet	FedAvg	5	China	CPU
14	MNIST	MLP	FedAvg	5	China	GPU
15	FashionMNIST	MLP	FedAvg	5	China	GPU
16	CIFAR10	fastermobilenet	FedAvg	5	China	GPU

Table 6.9: Details of Experiments 2,4,6 and 14,15,16

The following presents two sets of bar charts reflecting energy consumption and carbon emissions in experiments changing accelerator (CPU and GPU).

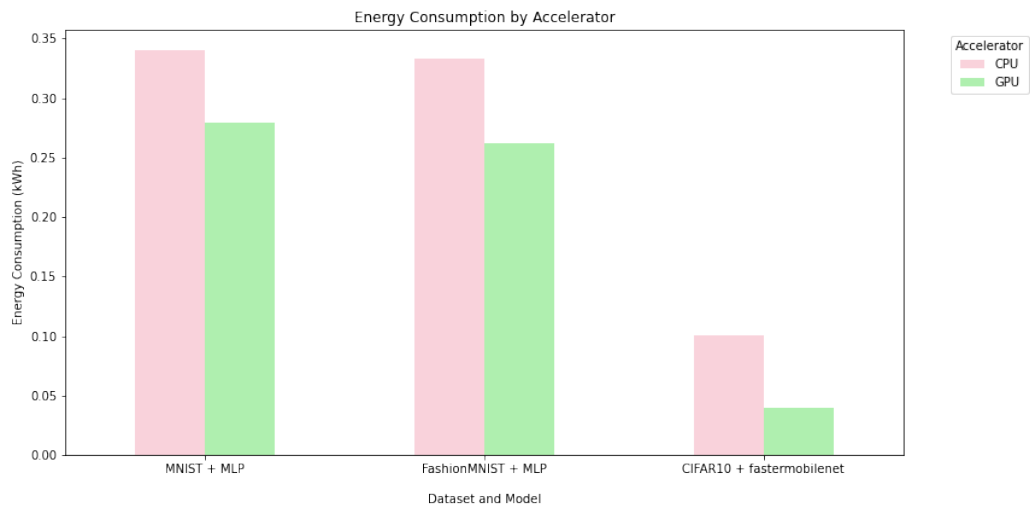


Figure 6.13: Impact of Accelerator on Energy Consumption in Experiments 2 and 14, 4 and 15, and 6 and 16

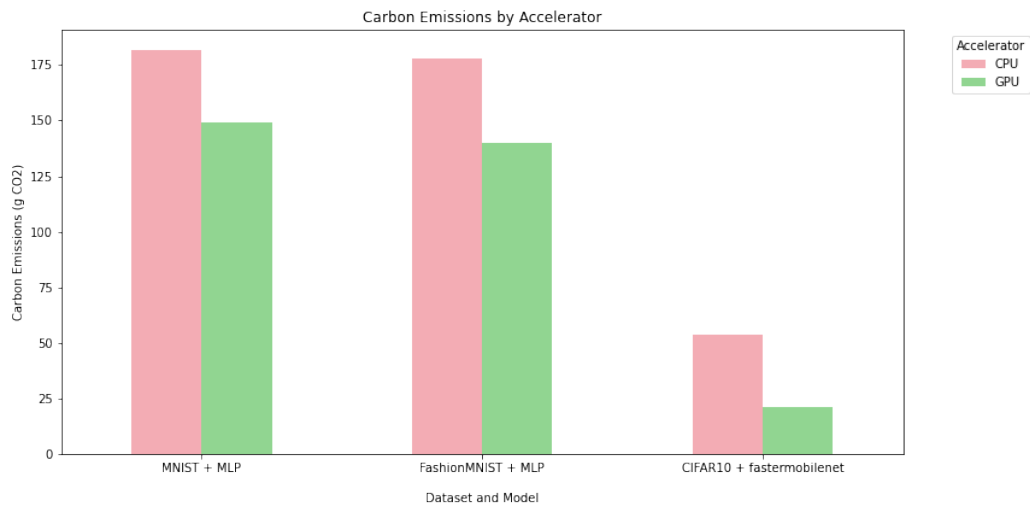


Figure 6.14: Impact of Accelerator on Carbon Emissions in Experiments 2 and 14, 4 and 15, and 6 and 16

Figure 6.13 shows that from the perspective of energy consumption, the type of accelerator has a significant impact on energy consumption. When using a CPU, the energy consumption of the MNIST + MLP and FashionMNIST + MLP combinations is about 0.33 kWh, while using a GPU reduces the energy consumption to about 0.25 kWh. Similarly, for the CIFAR10 + fastermobilenet combination, the energy consumption is about 0.1 kWh when using a CPU and about 0.05 kWh when using a GPU. It can be seen that the energy consumption with a GPU is lower than that with a CPU for all three combinations.

Figure 6.14 shows that from the perspective of carbon emissions, the type of accelerator also has a significant impact. The carbon emissions of the MNIST + MLP and FashionMNIST + MLP combinations are about 180 g CO₂ when using a CPU, compared to about 140 g CO₂ when using a GPU. For the CIFAR10 + fastermobilenet combination, the carbon emissions are about 50 g CO₂ when using a CPU and about 20 g CO₂ when using a GPU. This indicates that while GPUs may consume less energy than CPUs in some cases, their carbon emissions are still significantly lower than those of CPUs.

Overall, these experimental results show that the type of accelerator has a significant impact on both energy consumption and carbon emissions. GPUs generally consume less energy and emit less carbon when performing the same tasks. This may be due to the stronger parallel computing capabilities of GPUs, which can more efficiently handle large-scale computational tasks, thereby reducing energy consumption and carbon emissions.

Chapter 7

Summary and Conclusions

This thesis presents a comprehensive analysis and implementation of an environmental sustainability module for DFL systems, integrated into the Fedstellar platform. By focusing on the energy consumption and carbon emissions associated with various phases of DFL operations, valuable insights are gained into the factors that significantly influence the environmental impact of DFL systems.

7.1 Conclusion

- **Energy Consumption and Carbon Emissions Across Phases:** The communication phase emerged as the primary contributor to energy consumption and carbon emissions in DFL, especially in scenarios involving frequent data exchanges between nodes. The training phase also showed significant energy consumption, particularly when using complex models and larger datasets.
- **Impact of Federation Complexity:** The choice of aggregation algorithm (e.g., Krum vs. FedAvg) showed minimal impact on energy consumption and carbon emissions under controlled experimental conditions. The combination of dataset and model greatly influenced energy consumption and carbon emissions, with simpler models (e.g., MLP) generally consuming less energy compared to more complex models (e.g., CNN).
- **Effect of Number of Node:** Increasing the number of nodes led to higher energy consumption and carbon emissions due to the additional communication and synchronization required among the nodes.
- **Geographical Influence:** While geographical location did not significantly affect energy consumption, it had a substantial impact on carbon emissions due to variations in the carbon intensity of electricity across different regions.
- **Accelerator Types:** GPUs generally consumed less energy and produced lower carbon emissions compared to CPUs, likely due to their superior parallel processing capabilities.

7.2 Future Work

Building on the findings of this thesis, several avenues for future research and development are identified:

- **Advanced Optimization Techniques:** Investigate advanced optimization techniques, such as adaptive communication strategies and dynamic model selection, to further reduce the energy consumption and carbon emissions of DFL systems.
- **Real-World Deployment:** Conduct large-scale real-world deployments of the Fedstellar platform to validate the findings under diverse operational conditions and gather more comprehensive data on energy consumption and carbon emissions.
- **Diverse Communication Methods:** Explore the integration of various communication methods (e.g., Bluetooth, ZigBee, Ethernet) into the Fedstellar platform and assess their impact on energy efficiency and environmental sustainability.
- **Enhanced Monitoring and Reporting:** Develop enhanced monitoring and reporting tools within the Fedstellar platform to provide real-time feedback on energy consumption and carbon emissions, enabling more informed decision-making for optimizing DFL operations.

In conclusion, this thesis has laid a solid foundation for understanding and optimizing the environmental sustainability of DFL systems. The insights gained and the methodologies developed pave the way for future advancements in green computing, contributing to the broader goal of achieving sustainable and efficient AI deployments in the face of growing computational demands.

Bibliography

- [1] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, 2017. arXiv: 1712.01815 [cs.AI]. [Online]. Available: <https://arxiv.org/abs/1712.01815>.
- [2] Wikipedia contributors, *Google assistant* — *Wikipedia, the free encyclopedia*, [Online; accessed 30-July-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Google_Assistant&oldid=1236708270.
- [3] Wikipedia contributors, *Cortana (virtual assistant)* — *Wikipedia, the free encyclopedia*, [Online; accessed 30-July-2024], 2024. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Cortana_\(virtual_assistant\)&oldid=1235741000](https://en.wikipedia.org/w/index.php?title=Cortana_(virtual_assistant)&oldid=1235741000).
- [4] P. Covington, J. Adams, and E. Sargin, “Deep neural networks for youtube recommendations”, in *Proceedings of the 10th ACM Conference on Recommender Systems*, ser. RecSys ’16, Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 191–198, ISBN: 9781450340359. DOI: 10.1145/2959100.2959190. [Online]. Available: <https://doi.org/10.1145/2959100.2959190>.
- [5] Wikipedia contributors, *Deepart* — *Wikipedia, the free encyclopedia*, [Online; accessed 30-July-2024], 2024. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=DeepArt&oldid=1230840008>.
- [6] Wikipedia contributors, *Ibm watson* — *Wikipedia, the free encyclopedia*, [Online; accessed 30-July-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=IBM_Watson&oldid=1236620087.
- [7] Wikipedia contributors, *Hirevue* — *Wikipedia, the free encyclopedia*, [Online; accessed 30-July-2024], 2024. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=HireVue&oldid=1232852058>.
- [8] Wikipedia contributors, *Geolitica* — *Wikipedia, the free encyclopedia*, [Online; accessed 30-July-2024], 2024. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Geolitica&oldid=1220433353>.
- [9] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, “A review of applications in federated learning”, *Computers Industrial Engineering*, vol. 149, p. 106 854, 2020, ISSN: 0360-8352. DOI: <https://doi.org/10.1016/j.cie.2020.106854>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835220305532>.
- [10] N. Bouacida and P. Mohapatra, “Vulnerabilities in federated learning”, *IEEE Access*, vol. 9, pp. 63 229–63 249, 2021. DOI: 10.1109/ACCESS.2021.3075203.

- [11] E. T. Martínez Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys and Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023, ISSN: 2373-745X. DOI: 10.1109/comst.2023.3315746. [Online]. Available: <http://dx.doi.org/10.1109/COMST.2023.3315746>.
- [12] W. Henshall, *4 charts that show why ai progress is unlikely to slow down*, Accessed: 31.07.2024, 2023. [Online]. Available: <https://time.com/6300942/ai-progress-charts/>.
- [13] A. S. G. Andrae and T. Edler, “On global electricity usage of communication technology: Trends to 2030”, *Challenges*, vol. 6, no. 1, pp. 117–157, 2015, ISSN: 2078-1547. DOI: 10.3390/challe6010117. [Online]. Available: <https://www.mdpi.com/2078-1547/6/1/117>.
- [14] L. Zumtaugwald, “Designing and implementing an advanced algorithm to measure the trustworthiness level of federated learning models”, M.Sc. Thesis, University of Zurich, Faculty of Economics, 2023.
- [15] E. T. Martínez Beltrán, Á. L. Perales Gómez, C. Feng, *et al.*, “Fedstellar: A Platform for Decentralized Federated Learning”, *Expert Systems with Applications*, vol. 242, p. 122861, 2024, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2023.122861.
- [16] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys & Tutorials*, 2023.
- [17] L. Edwards. “The difference between renewable and non-renewable resources”. (2023), [Online]. Available: <https://www.nesfircroft.com/resources/blog/the-difference-between-renewable-and-non-renewable-resources/> (visited on 08/01/2023).
- [18] X. Qiu, T. Parcollet, D. J. Beutel, T. Topal, A. Mathur, and N. D. Lane, *Can federated learning save the planet?*, 2021. arXiv: 2010.06537 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2010.06537>.
- [19] D. Patterson, J. Gonzalez, U. Hölzle, *et al.*, *The carbon footprint of machine learning training will plateau, then shrink*, 2022. arXiv: 2204.05149 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2204.05149>.
- [20] X. Qiu, T. Parcollet, J. Fernandez-Marques, *et al.*, *A first look into the carbon footprint of federated learning*, 2023. arXiv: 2102.07627 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2102.07627>.
- [21] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, *Quantifying the carbon emissions of machine learning*, 2019. arXiv: 1910.09700 [cs.CY]. [Online]. Available: <https://arxiv.org/abs/1910.09700>.
- [22] S. Savazzi, V. Rampa, S. Kianoush, and M. Bennis, “An energy and carbon footprint analysis of distributed and federated learning”, *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 1, pp. 248–264, Mar. 2023, ISSN: 2473-2400. DOI: 10.1109/tgcn.2022.3186439. [Online]. Available: <http://dx.doi.org/10.1109/TGCN.2022.3186439>.

- [23] A. Das, “A study on energy consumption of different wireless devices”, *International Journal of Engineering Research Technology IJERT*, Mar. 2020, ISSN: 2278-0181.
- [24] E. T. Martínez Beltrán, Á. L. Perales Gómez, C. Feng, *et al.*, *Enriquetomasmb/fedstellar*, 2023. [Online]. Available: <https://github.com/enriquetomasmb/fedstellar>.
- [25] M. PassMark, *Cpu benchmarks compilation*, Accessed: 2023-08-04, 2022. [Online]. Available: <https://www.kaggle.com/datasets/alanjo/cpu-benchmarks?resource=download>.
- [26] T. P. Nathaneiel V. KELSO, *Nvkelso/natural-earth-vector*, Accessed: 2023-08-04, 2022. [Online]. Available: <https://github.com/nvkelso/natural-earth-vector/tree/master>.
- [27] V. from Mila and the DataForGoodFR, *Mlco2/codecarbon*, Accessed: 2023-08-04, 2023. [Online]. Available: <https://github.com/mlco2/codecarbon>.

Abbreviations

AI	Artificial Intelligence
AI-HLEG	High-Level Expert Group on AI
CFL	Centralized Federated Learning
CPU	Central Processing Unit
CPUs	Central Processing Units
CSV	Comma-Separated Values
DFL	Decentralized Federated Learning
FL	Federated Learning
GPU	Graphics Processing Unit
GPUs	Graphics Processing Units
HTML	HyperText Markup Language
IID	Independent and Identically Distributed
I/O	Input/Output
IP	Internet Protocol
JSON	JavaScript Object Notation
kWh	kilowatt-hour
ML	Machine Learning
MLP	MultiLayer Perceptron
Non-IID	Non-Independent and Identically Distributed
PUE	Power Usage Effectiveness
RER	Renewable Energy Ratio
REST API	Representational State Transfer Application Programming Interface
TDP	Thermal Design Power
WiFi	Wireless Fidelity

List of Figures

1.1	Amount of Computing Used to Train Notable AI Models [12]	2
1.2	Global Electricity Consumption Ratio of Data Centers and All Other Industries [13]	3
1.3	Global Greenhouse Gas Emissions Ratio of Data Centers and All Other Industries [13]	3
2.1	Traditional ML (left) and FL (right)	8
2.2	CFL (left) and DFL (right) [16]	9
2.3	Fedstellar User Interface	11
2.4	Fedstellar Advanced Mode	11
2.5	Fedstellar Real-time Monitoring Metrics	12
3.1	Detailed Metrics for Three Dimensions: Carbon Intensity Energy Source, Hardware Efficiency, and Federation Complexity [14]	19
4.1	Architecture of Environmental Sustainability Module in DFL	24
4.2	Power Consumption of a WiFi Radio at Different Data Production Rates and Different Transmission Intervals [23]	28
5.1	Components of the Fedstellar Platform: User, Frontend, Controller, and Core [15]	33
5.2	Overall Architecture of the Fedstellar platform [15]	34
5.3	The Positions of R_k and PUE_k in the Overall Architecture	36
5.4	The Frontend Interface Screenshot for Variables R_k and PUE_k	38
5.5	The Position of $E_{\text{byte}}^{(C)}$ in the Overall Architecture	39
5.6	The Frontend Interface Screenshot for Variable $E_{\text{byte}}^{(C)}$	40

5.7	Example of Energy Consumption for Each Node in Training (Using CPU) .	42
5.8	Example of Energy Consumption for Each Node in Training (Using GPU)	43
5.9	Example of Energy Consumption for Each Node in Aggregation	44
5.10	Example of Energy Consumption for Each Node in Communication	45
5.11	Example of Carbon Emissions for Each Node in Training (Using CPU) . .	46
5.12	Example of Carbon Emissions for Each Node in Training (Using GPU) . .	46
5.13	Example of Carbon Emissions for Each Node in Aggregation	47
5.14	Example of Carbon Emissions for Each Node in Communication	47
5.15	Example of Energy Consumption for Each Node	48
5.16	Example of Carbon Emissions for Each Node	48
5.17	Example of Total Energy Consumption	49
5.18	Example of Total Carbon Emissions	50
6.1	Comparison of Energy Consumption Across Three Phases in Experiment 2	54
6.2	Comparison of Carbon Emissions Across Three Phases in Experiment 2 . .	54
6.3	Comparison of Energy Consumption Across Three Phases in Experiment 10	56
6.4	Comparison of Carbon Emissions Across Three Phases in Experiment 10 .	56
6.5	Impact of Aggregation Algorithms on Energy Consumption in Experiments 1 and 2	58
6.6	Impact of Aggregation Algorithms on Carbon Emissions in Experiments 1 and 2	58
6.7	Impact of Combinations of Dataset and Model on Energy Consumption in Experiments 2 through 7	60
6.8	Impact of Combinations of Dataset and Model on Carbon Emissions in Experiments 2 through 7	60
6.9	Impact of the Number of Nodes on Energy Consumption in Experiments 2 and 8, 4 and 9, and 6 and 10	62
6.10	Impact of the Number of Nodes on Carbon Emissions in Experiments 2 and 8, 4 and 9, and 6 and 10	62
6.11	Impact of Location on Energy Consumption in Experiments 2 and 11, 4 and 12, and 6 and 13	63

6.12 Impact of Location on Carbon Emissions in Experiments 2 and 11, 4 and 12, and 6 and 13	64
6.13 Impact of Accelerator on Energy Consumption in Experiments 2 and 14, 4 and 15, and 6 and 16	65
6.14 Impact of Accelerator on Carbon Emissions in Experiments 2 and 14, 4 and 15, and 6 and 16	65

List of Tables

3.1	Comparison of Related Work Regarding Energy Consumption and Carbon Footprint in ML, CFL, and DFL	18
6.1	Fixed Parameters in Experimental Setup	52
6.2	Variable Parameters for Experiments 1 through 16	52
6.3	Estimated Model Parameter Size	53
6.4	Details of Experiment 2 and 10	53
6.5	Details of Experiments 1 and 2	57
6.6	Details of Experiments 2 through 7	60
6.7	Details of Experiments 2,4,6 and 8,9,10	61
6.8	Details of Experiments 2,4,6 and 11,12,13	63
6.9	Details of Experiments 2,4,6 and 14,15,16	65

Appendix A

Installation Guidelines

This chapter provides a guideline for installing and deploying Fedstellar platform. Fedstellar is a modular, adaptable and extensible platform for creating centralized and decentralized architectures using Federated Learning. Also, the platform enables the creation of a standard approach for developing, deploying, and managing federated learning applications. The platform enables developers to create distributed applications that use federated learning algorithms to improve user experience, security, and privacy. It provides features for managing data, managing models, and managing federated learning processes. It also provides a comprehensive set of tools to help developers monitor and analyze the performance of their applications.

A.1 Prerequisites

Prerequisites

- Python 3.8 or higher (3.11 recommended)
- pip3
- Docker Engine 24.0.4 or higher
(24.0.7 recommended, <https://docs.docker.com/engine/install/>)
- Docker Compose 2.19.0 or higher
(2.19.1 recommended, <https://docs.docker.com/compose/install/>)

A.2 Deploy a virtual environment

Virtualenv is a tool to build isolated Python environments.

It's a great way to quickly test new libraries without cluttering your global site-packages or run multiple projects on the same machine which depend on a particular library but not the same version of the library.

Since Python version 3.3, there is also a module in the standard library called `venv` with roughly the same functionality.

A.2.1 Create virtual environment

In order to create a virtual environment called e.g. `fedstellar` using `venv`, run:

```
1 $ python3 -m venv fedstellar-venv
```

A.2.2 Activate the environment

Once the environment is created, you need to activate it. Just change directory into it and source the script `Scripts/activate` or `bin/activate`.

With bash:

```
1 $ cd fedstellar-venv
2 $ . Scripts/activate
3 (fedstellar-venv) $
```

With csh/tcsh:

```
1 $ cd fedstellar-venv
2 $ source Scripts/activate
3 (fedstellar-venv) $
```

Notice that the prompt changes once you are activate the environment. To deactivate it just type `deactivate`:

```
1 (fedstellar-venv) $ deactivate
2 $
```

After you have created the environment, you can install `fedstellar` following the instructions below.

A.3 Building from source

A.3.1 Obtaining the platform

You can obtain the source code from <https://github.com/enriquetomasmb/fedstellar>

Or, if you happen to have git configured, you can clone the repository:

```
1 git clone https://github.com/enriquetomasmb/fedstellar.git
```

Now, you can move to the source directory:

```
1 cd fedstellar
```

A.3.2 Dependencies

Fedstellar requires the additional packages in order to be able to be installed and work properly.

You can install them using pip:

```
1 pip3 install -r requirements.txt
```

A.3.3 Checking the installation

Once the installation is finished, you can check by listing the version of the Fedstellar with the following command line:

```
1 python app/main.py --version
```

A.4 Building the fedstellar participant

A.4.1 Docker image (CPU version)

You can build the docker image using the following command line in the root directory:

```
1 docker build -t fedstellar -f Dockerfile-cpu .
```

A.4.2 Docker image (GPU version)

You can build the docker image using the following command line in the root directory:

```
1 docker build -t fedstellar-gpu -f Dockerfile-gpu .
```

Also, you have to follow the instructions in the following link to install nvidia-container-toolkit:

<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html>

A.5 Checking the docker images

You can check the docker images using the following command line:

```
1 docker images
```

A.6 Running Fedstellar

To run Fedstellar, you can use the following command line:

```
1 python app/main.py [PARAMS]
```

The first time you run the platform, the fedstellar-frontend docker image will be built. This process can take a few minutes.

You can show the PARAMS using:

```
1 python app/main.py --help
```

The frontend will be available at <http://127.0.0.1:5000> (by default)

To change the default port of the frontend, you can use the following command line:

```
1 python app/main.py --webport [PORT]
```

To change the default port of the statistics endpoint, you can use the following command line:

```
1 python app/main.py --statsport [PORT]
```

A.7 Fedstellar Frontend

You can login with the following credentials:

```
1 - User: admindemo
2 - Password: dfl-fedstellar
```

If not working the default credentials, send an email to Enrique Tomás Martínez Beltrán to get the credentials.

A.8 Stop Fedstellar

To stop Fedstellar, you can use the following command line:

```
1 python app/main.py --stop
```

Be careful, this command will stop all the containers related to Fedstellar: frontend, controller, and participants.

Appendix B

Contents of the CD

The CD contains all the documents, thesis source code, and installation instructions for this thesis.