Impact Analysis of Different Consensus, Participant Selection and Scoring Algorithms in Blockchain-based Federated Learning Systems Using a Modular Framework

Coelho Dias, Henrique Afonso

Award date:
2022

Department of Mathematics and Computer Science

Impact Analysis of Different Consensus, Participant Selection and Scoring Algorithms in Blockchain-based Federated Learning Systems Using a Modular Framework

Master Thesis

Henrique Afonso Coelho Dias

Supervisor

prof. dr. ir. N. Meratnia

Eindhoven, September 2022

Abstract

Federated Learning allows multiple distributed clients to collaborate on training the same Machine Learning model. Blockchain-based Federated Learning has emerged in recent years to improve its transparency and information safety issues. On the one hand, it eliminates the need for a central orchestrator, removing the single point of failure in the network. On the other hand, it facilitates aspects such as traceability, auditability, authentication and persistency, that, together, improve the transparency and safety of the federated training process and accommodate new verification algorithms in order to detect malicious agents.

In these systems, it is common to score each client's model update in order to determine if it is a good contribution to the global model. With Blockchain-based Federated Learning being increasingly adopted in IoT networks, where low powered devices with low resources are the norm, it is important to ensure that the system consumes the least amount of resources. The current literature has very little information regarding how different algorithms impact the resource usage of the system. Additionally, there is no publicly available framework that can be used to implement a Blockchain-based Federated Learning system.

In this thesis, we design and implement the first modular open-source framework for Blockchain-based Federated Learning using Ethereum and TensorFlow. This framework can be easily adapted to support multiple architectures, as well as different scoring, aggregation, and privacy algorithms. With this framework, we proceed to do the first known analysis of how different aspects of Blockchain-based Federated Learning, such as consensus, participation selection and scoring algorithms, impact the accuracy, execution time and communication and computation costs. Additionally, the same analysis will be done per each scoring algorithm to analyze the impact of the number of clients and privacy mechanisms on the aforementioned aspects. Finally, we also provide a proof of concept of how the framework can be adapted to support, not only the Horizontal Federated Learning, but also the Vertical Federated Learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine Learning (ML) has revolutionized the way we use and work with data, opening ways to new techniques and explorations. ML models can be powerful tools to predict things that would otherwise require high amounts of human effort. For example, an image recognition model may be created to help medical professionals diagnose diseases. Similarly, a model may be created to detect abnormal heart rate or walking patterns based on smart watch sensor data. Even though models such as these can be very helpful, they have to be trained with high amounts of good quality data in order for the model to perform accurately [30]. To address this issues, there are techniques that allow multiple parties to collaboratively train the same models.

In 2016, Google researchers attempted to build communication-efficient deep neural networks in decentralized settings [51]. The result of this work was the introduction of a new way of collaboratively training Machine Learning models, which they termed Federated Learning. Federated Learning (FL) allows multiple clients, in different locations, to collaborate on the training of a global Machine Learning model without sharing their own data with each other. Instead of sharing the raw data, clients only share model parameters, such as weights. This brings some benefits. The first benefit is that, by not sharing raw data, models can preserve data privacy, allowing them to be trained on sensitive data. In addition, since model parameters are usually much smaller than the raw data, this leads to less data being transported over the networks. Finally, since the data is distributed among different clients, a single powerful server is not required to train the model, as usually training models with smaller amounts of data is less computationally expensive.

1.1 Motivation

Currently, most FL networks include a central server that coordinates the federated training process and aggregates the model weights from each of the clients into a single model. This central coordinator is a single point of failure in the network, since it is always required to be online and behave correctly [40, 83]. To address this, Blockchain-based Federated Learning (BFL) techniques have been proposed.

By combining Blockchain to Federated Learning, not only can the central orchestrator

be eliminated, but also the federated training process can be made more transparent. In the blockchain, each transaction is recorded in the distributed ledger. These transactions record information such as local updates, scores, aggregations, among others. Having this information in a public ledger allows for a transparent training process and reward distribution [40]. The following are some aspects that Blockchain can bring to Federated Learning when combined:

- *Traceability and Auditability.* Due to the structure of the blockchain, it is possible to trace transactions to their original source, which can be useful for auditability purposes [7, 83].
- *Data Immutability and Persistency.* Once transactions are added to the distributed ledger, it is nearly impossible to revert them or change their information [7, 70]. This ensures that data is not changed and it can be retrieved after the fact.
- *Decentralization.* The involvement of a central orchestrator is eliminated and the processing of the aggregation is replaced by multiple servers [48, 59, 70, 83]. This improves the resilience and availability of the system.
- *Authentication.* Blockchain ensures the authentication of data and messages due to the verification mechanisms in place, such as the usage of private keys to sign transactions [70].

Blockchain platforms can implement different consensus algorithms, which can lead to very different resource consumption [60], as well as different degrees of latency [3]. Two other important aspects in BFL systems are the participant selection and the scoring algorithms. While the former indicates how the participants are chosen to submit their updates in each round, the latter aids on scoring each client's model update in order to identify which ones are the best and should be included in the final aggregation. All this algorithms influence the accuracy, convergence, and resource consumption. At the same time, each algorithm performs differently with different amounts of devices, as well as with different degrees of privacy, leading to different accuracy results, as well as resource consumption.

1.2 Problem Statement

To the best of our knowledge, there is very little literature on the impact of different algorithms of BFL systems, namely consensus, participant selection and scoring algorithms, on the execution time, accuracy, convergence, communication and computation costs of the system. In addition, there is no research on how different scoring algorithms are impacted in terms of the aforementioned aspects when different amounts of clients and privacy degrees are used. With BFL being increasingly adopted in IoT networks, where low powered devices with low resources are the norm, it is important to ensure the system consumes the least amount of resources, especially regarding the clients.

Additionally, even though there is literature on designing BFL frameworks, none of them are open source, or modular enough to support different algorithms. Such framework could be used to empower future research on new algorithms, as well as help those who want to implement their own BFL system.

1.3 Research Questions

Taking into account the motivation and problem statement, this work will focus on answering the following main research question:

What is the impact of different consensus, participant selection and scoring algorithms in a Blockchain-based Federated Learning system on execution time, convergence and accuracy, as well as communication and computation costs?

This research question can be further sub-divided into four sub-questions:

1. *How to design a modular framework that allows easy customization of different algorithms related to different parts of the Blockchain-based Federated Learning system?*

2. *How do consensus, participant selection, and scoring algorithms influence execution time, convergence, accuracy, and communication and computation costs of the system?*

3. *How does the number of clients, as well as degrees of privacy impact the different scoring algorithms?*

4. *How can we build a Blockchain-based Federated Learning framework that supports different data partition formats, such as vertical and horizontal?*

1.4 Contributions and Outline

The contributions of this thesis are as follows: (i) design and implementation of the first open-source modular framework for BFL that can be easily adapted to support new scoring, aggregation, and privacy algorithms. This framework can be used to empower future research; (ii) the first comparative study of how different algorithms of BFL, namely consensus, participant selection, and scoring algorithms, impact the execution time, transaction costs, transaction latency, model accuracy and convergence, communication costs, and computation costs of the system; (iii) the first comparative study of how the number of clients and different degrees of privacy impact the accuracy, execution time and communication and computation costs of different scoring algorithms; and (iv) a proof-of-concept of a Blockchain-based Vertical Federated Learning.

The remainder of the thesis is structured as follows. Chapter 2 provides definitions and fundamental concepts about BFL, as well as background information of the algorithms and mechanisms that will be explored. Chapter 3 reviews the existing work regarding algorithms and mechanisms used in BFL systems. Chapter 4 explains the design and implementation of the framework. Chapter 5 provides information regarding the experimental setup of the experiments. Chapter 6 provides the impact analysis of using different consensus algorithms. Chapter 7 provides the impact analysis of using different scoring algorithms, as well as how they behave with different number of clients and privacy degrees. Chapter 8 provides analysis of the proof of concept of Vertical Federated Learning applied in a BFL system. Finally, Chapter 9 discusses the results, contributions and provides directions for future works.

Chapter 2

Background

This chapter gives an overview of the fundamental concepts and the algorithms important for this thesis.

2.1 Machine Learning

Machine Learning is a sub-field of Artificial Intelligence that builds models based on statistical and algorithmic concepts in order to detect relevant patterns based on previously-seen data [27]. There are four categories of learning, i.e., *supervised*, *semi-supervised*, *unsupervised* and *reinforcement learning* [73]. Each category performs different types of tasks on different types of data. This study focuses only on supervised learning.

In supervised learning, algorithms build mathematical models from labeled data, which is data in the format (X, y), where X is the input sample and y is the expected output, or label. During training, algorithms provide the model with the input samples and improve the model by comparing its output with the expected outputs (the so called ground truth). Supervised learning problems can be divided into *regression problems*, if the output is a continuous variable, or *classification problems*, if the output is a discreet variable [73].

2.1.1 Federated Learning

Federated Learning is a ML technique, in which different distributed clients collaboratively train a model under the supervision of a centralized server. Clients are distributed heterogeneous devices with their own computing resources and they are responsible for producing and maintaining their own data [41]. The data is assumed to be non independent or non identically distributed (*non-iid*).

Since clients are heterogeneous and distributed, having different communication costs and response times are normal. In addition, some clients may operate under constrained networks with either low or limited bandwidth. Therefore, it is important that new FL techniques ensure that communication and resource usage is minimized.

During the training process, the raw data never leaves the clients and only the model parameters, such as model weights, are exchanged with the server in order to compute

the global model. However, model weights can be target of inference attacks and leak secret information [88]. Consequently, new Federated Learning architectures and algorithms must be compatible with techniques that guarantee privacy, such as differential privacy, homomorphic encryption, secure multiparty computation or other cryptographic protocols [88].

After each round of training, the central server aggregates the local updates. Usually, this is done using a mathematical formula, for example, Federated Averaging (*FedAvg*) [51], which calculates the weighted average of all clients.

2.1.2 Categories of Federated Learning

According to [65, 88], with respect to the different data partition among the clients, the federated Learning techniques can be broadly divided into three main categories, i.e., (i) horizontal, (ii) vertical, and (iii) federated Transfer Learning. We focus on the first two categories.

In *Horizontal Federated Learning* (HFL), clients with the same data structure collaborate to build a single model. In other words, the different data sets in the different clients share the same feature space, but not the sample space. For example, two banks branches operating in different cities have similar businesses (feature space), but different clients (sample space). The architecture of HFL, depicted in Figure 2.1, consists of multiple clients training a model, while the central server performs the aggregation of all local updates.

In *Vertical Federated Learning* (VFL), clients share an intersecting sample space, but different feature spaces. For example, two different banks with different products operating in the same city have a similar client base (sample space), but different information about each client (feature space). The architecture of VFL, depicted in Figure 2.2, is similar to the one of HFL. However, it requires an additional step, calculating the Private Set Intersection (PSI) [84], since the clients do not share the exact same sample space. PSI is a protocol by which multiple clients can calculate the common samples without sharing their raw data [84].

2.2 Blockchain

A blockchain is an immutable distributed ledger, which is a database of transactions maintained by several computers, also known as nodes, linked through a peer-to-peer network. The concept of blockchain was first introduced by Stuart Haber and W. Scott Stornetta in 1991 [76], being popularized by Satoshi Nakamoto in 2008 with the introduction of the cryptocurrency Bitcoin [57].

In a blockchain, the data is structured as blocks, as can be seen in Figure 2.3. Each block contains a certain number of transactions and links to the previous block via a cryptographic hash, forming a chain. This guarantees fidelity and trust without requiring a trusted third party, which is why it is called a *trustless* system. In addition, since the record is immutable and decentralized, all transactions can be transparently viewed by others.

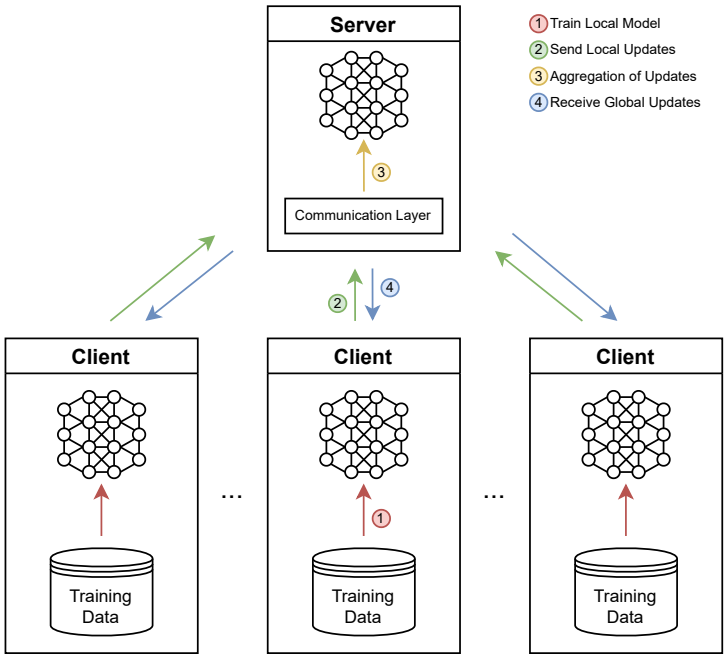


Figure 2.1: Horizontal Federated Learning Architecture

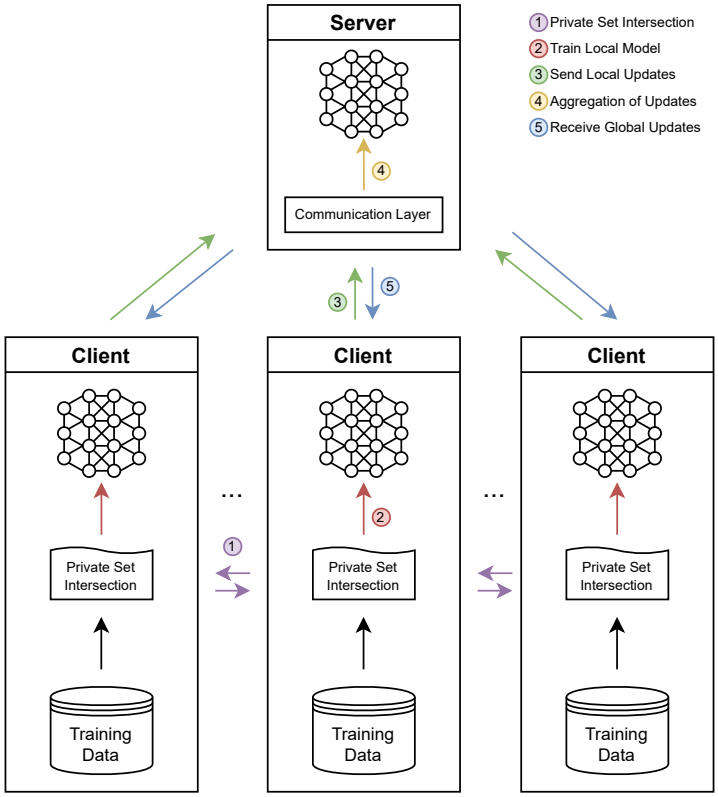


Figure 2.2: Vertical Federated Learning Architecture

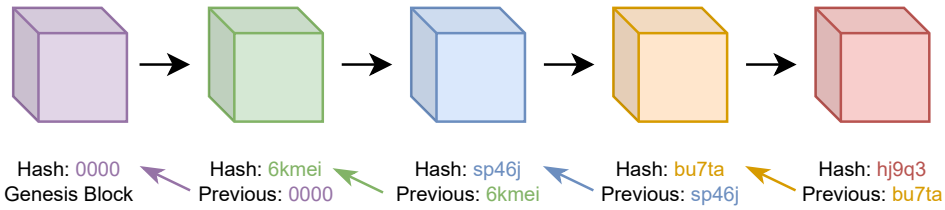


Figure 2.3: Blockchain Representation

As mentioned beforehand, a blockchain is maintained by several nodes in a peer-to-peer network. As transactions come in, nodes compete in order to generate the next block. Since it is a decentralized process, multiple nodes will try to create the next block of the chain in parallel. In order to reach an agreement between the nodes, a consensus algorithm is used. The consensus algorithm allows to reach an agreement between multiple decentralized nodes without requiring a singular node to be in charge.

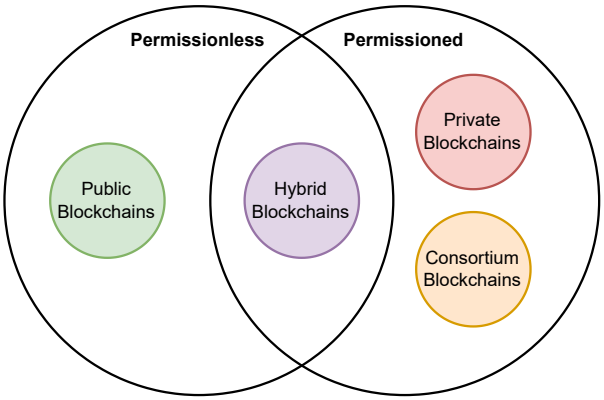


Figure 2.4: Blockchain Types

Figure 2.4 illustrates four different types of blockchain, i.e., public, private, consortium, and hybrid. Some are permissionless, which means that anyone can join the network, while some are permissioned, which means that only allowed parties can join the network.

- *Public Blockchains* are permissionless and therefore anyone can join and participate in the network. There is no central authority.
- *Private Blockchains*, in contrary to the public blockchains, private blockchains are permissioned with a single central authority. They can only be accessed by allowed parties and they are usually used within organizations.
- *Consortium Blockchains*, similarly to the private blockchains, are permissioned. However, instead of being controlled by a single authority, they are controlled by a group of different authorities.
- *Hybrid Blockchains* have features of both permissioned and permissionless blockchain systems. They, on one hand, are usually controlled by a single authority, while on the other hand, have a mixed usage of permissioned and permissionless protocols running in parallel for different use cases.

2.2.1 Smart Contracts

Smart contracts [82] are small computer programs that live within the blockchain and automatically run when predetermined conditions are met. As they live in the blockchain, they are trustless and are typically used to automate the execution of agreements. This way, every party involved in the agreement is certain that it will be honored once the conditions of the agreement are met. Some blockchain platforms, such as the Ethereum [87], provide functionality for smart contracts.

2.2.2 Blockchain Platforms

As explained in section 2.2, blockchain platforms allow developers to build applications on top of the blockchain technologies. Even though all platforms are based on the concept of blockchain, they all have different characteristics and restrictions, as well as different sets of features.

As it can be seen in Table 3.1, around half of the implementations used an already existing platform, where the remaining preferred to implement their own blockchain platform. Among already existing platforms, Ethereum is the most popular. When implementing a custom platform, it is easier to overcome certain restrictions such as limits on data per block [9, 35].

2.2.3 Consensus Algorithm

The consensus algorithm is one of the most important components of a Blockchain platform. Consensus is the process of reaching an agreement on a single value among different distributed processes [68]. These algorithms are designed to be reliable even on networks that have unreliable blockchain nodes. In blockchain, the consensus algorithm is used to reach consensus on the next block of the chain [66]. The following consensus algorithms will be compared in our work:

- The *Proof of Work (PoW)* [26] consensus algorithm was first introduced in the context of blockchain platforms by Satoshi Nakamoto in Bitcoin [57]. It works by means of computation effort proofs, where a set of virtual miners race in solving a complex, yet feasible, mathematical problem. The winner of the race generates a cryptographic proof based on the solution of the problem that can be easily verified by others. Then, the winner adds a new block containing the newly verified transactions to the blockchain. In addition, the winner is rewarded according to some pre-determined rules.

  On the one hand, it is a simple algorithm, for which proofs are hard to create, but easy to verify [40]. Not only is it robust and proven to work, but the cost of attacking a PoW blockchain is extremely high [40]. For an attack to be successful, it needs to control more than half of the network [40]. On the other hand, PoW consumes extreme amounts of energy and it is hard to scale [20, 40, 60].

- The *Proof of Authority (PoA)* [77] consensus algorithm is a reputation-based consensus algorithm that is most commonly used in private blockchain networks. It works by having a set of validator nodes that are responsible for validating new transactions. In order to ensure the correctness of the validator nodes, they have to

stake their own reputation. In addition, the validators are known trusted entities that are manually chosen by the network owner.

On the one hand, PoA provides high throughput and high scalability [2]. On the other hand, the main criticism of PoA is that it is usually has a small number of validators, that are manually chosen. Therefore, it has a lesser degree of decentralization. Consequently, PoA is not as common in public blockchain networks as it is for private networks [2].

- The *QBFT Byzantine Fault Tolerance (QBFT)* [54] consensus algorithm is similar to the Practical Byzantine Fault Tolerance (PBFT) [10] algorithm, which is a three-phase protocol that allows a network with $3f + 1$ nodes, where f is the maximum amount of faulty nodes, to reach consensus. The different between QBFT and PBFT is that in the former the set of validators is dynamic, while in the latter, it is static. The network reaches a consensus once $2f + 1$ nodes agree.

  One the one hand, QBFT allows for high consensus efficiency in high throughput networks [40]. On the other hand, it will stop working properly if only 33% or less blockchain nodes are running and it can also have high communication costs due to its three-phase protocol nature [40].

2.3 Blockchain-based Federated Learning

Recently, the idea of applying blockchain to FL has emerged. This is motivated by the fact that FL architectures are highly dependent on a single central server, leading to a central point of failure, that can be either overloaded or compromised. With blockchain, the central server can be replaced by multiple decentralized servers that operate the blockchain nodes. In addition, blockchain can provide authentication, traceability, auditability and data preservation.

As stated in [83], so far three different main architecture groups have been proposed for BFL systems: *Fully Coupled BFL*, *Flexibly Coupled BFL*, and *Loosely Coupled BFL*. In Fully Coupled BFL, the distributed clients are the only devices in the network and act as both clients and servers, performing the training and aggregation, as well as running the blockchain nodes. In Loosely Coupled BFL, the central server still exists and blockchain is only used to manage the reputation of clients. Flexibly Coupled BFL provides a middle ground between both, where there are servers and clients in the network. The servers are responsible for maintaining the blockchain nodes and use the blockchain to store information such as aggregations, scores, among others. The architecture is depicted in Figure 2.5.

2.3.1 Participants Selection Algorithms

Participant selection algorithms are algorithms that are used to decide how many and which clients are selected to participate in each round. There are two main participant selection algorithms:

- *Random selection*, where both the number of participants, as well as the participants themselves are selected randomly before the start of each round.

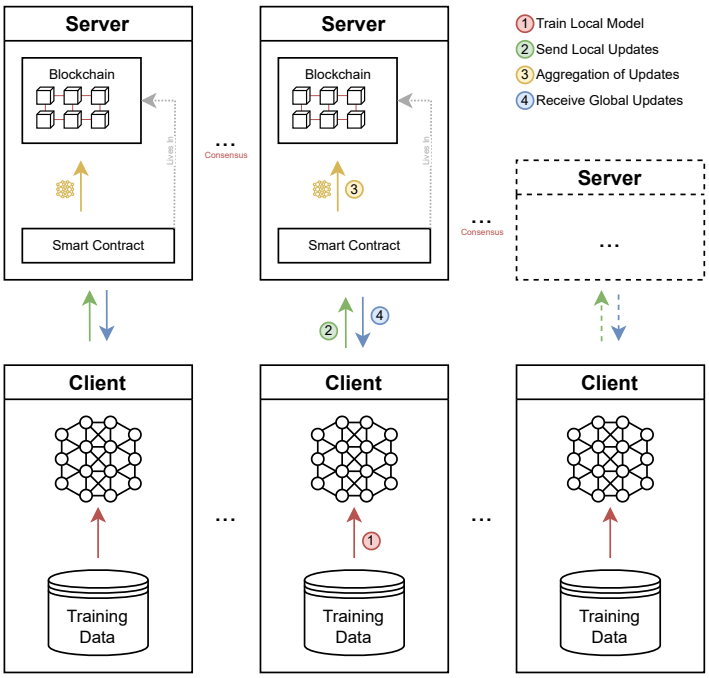


Figure 2.5: Blockchain-based Federated Learning Architecture

- *First-come first-served*, where the number of participants n is chosen randomly before the start of each round. Then, the first n clients to take initiative to join the round will participate.

2.3.2 Scoring and Aggregation Algorithms

During the training process, each client produces its model parameter updates and communicates them to the servers through the blockchain. These parameters are then aggregated. However, there are different security aspects that should be taken into account here as the parameter updates creates a possibility for performing different attacks such as poisoning attacks [69] and plagiarism attacks [48].

- *Poisoning attacks* happen when clients willingly send parameter updates that decrease the quality of the model. They may have been generated using an unreliable data set, or done on purpose. To avoid other participants to provide unreliable data to degrade the model performance, there are dynamic verification techniques [83, 92] that allow to ignore low quality data.

- *Plagiarism attacks* happen when lazy clients plagiarize other client's models updates without really training their models. These attacks can be addressed via pseudo-noise algorithms [59]. In addition, plagiarism attacks within the same round can be avoided by secure communication methods, such as differential privacy [48], through which plagiarism attacks where a client reuses parameters from a previous round can be avoided by simply comparing the different client's updates.

To mitigate the poisoning and plagiarism attacks, different scoring algorithms were developed. Scoring algorithms are used to give each client, or its model update, a score. Based on this score, the submission may have more or less impact on the aggregation, if any at all. In addition, scores are used for reward mechanisms in systems where public

models are being trained and clients need some sort of incentive to keep participating [5, 32, 50, 78]. In this section, we explain briefly how each of the scoring algorithms works and if they influence the aggregation algorithm.

2.3.2.1 BlockFlow Score

The BlockFlow algorithm [55] work by giving each submission a score and, based on that score, do the aggregation. In this algorithm, each client a gives each other client k a score $s_{a,k} \in [0.0, 1.0]$, which can be based on a's validation set accuracy using k's submission. Based on this scores, a median score and an evaluation quality scores are calculated. The overall scores will then be the minimum between the scaled median score and the scaled least accurate evaluation score.

With the final scores, the aggregation is calculated using the scores as weights in the Federated Averaging algorithm, instead of the number of samples. More details regarding the algorithm specifics can be found in the original paper.

2.3.2.2 Marginal Gain Score

The Marginal Gain algorithm [8], also known as contributivity score, is calculated by summing the marginal performance gains of all the client's model updates so far. Similarly to BlockFlow scoring, each client has to give each other clients' submission a score. The formula of the client c's submission score $S(c)$ is calculated as follows:

$$S(c) = \sum_{r} (v(M_r) - v(M^c_{r+1})) \tag{2.1}$$

, where v is a performance metric, such as accuracy, and m is the model and r the round. These scores are used as weights in the Federated Averaging Algorithm. If the submission's score is equal or below 0, the submission is ignored.

2.3.2.3 Multi-KRUM Score

The Multi-KRUM algorithm [64, 74, 91] works by giving each submission a score and eliminating dubious submissions based on their score. This scores are calculated by the servers and are based on the Euclidean distances between the different client c's submissions. The score of each client is denoted as $S(c)$ and calculated as follows:

$$S(c) = \sum_{c \to k} ||\Delta w_c - \Delta w_k||^2 \tag{2.2}$$

, where Δw is a submission and $c \to k$ are the clients k whose submission Δw_k are the $R - f - 2$ closest to Δw_c. In this formula, R is the total number of submissions, while f represents the amount of Byzantine clients. After giving each submission a score, the $R - f$ clients with the lowest scores are chosen and the remaining are rejected. Please note that Byzantine fault tolerant systems behave correctly when no more than f out of $3f + 1$ replicas fail.

2.3.3 Privacy Mechanisms

Even though FL is already more secure than centralized ML in the sense that the raw data is never shared, the model weights can be exploited via inference attacks [88]. Inference attacks are attacks in which the weights are used to reverse-engineer the original data.

Additionally, in BFS systems, the weights are visible to all other client or participant since the blockchain provides an immutable, traceable and auditable record of the whole process. Consequently, it is important to reduce the surface for attacks when it comes to the model parameters.

2.3.3.1 (Local) Differential Privacy

Differential Privacy (DP) [80] is a set of mathematical constrains that algorithms must observe in order to ensure a certain degree of privacy. In other words, some data is differentially private if, by looking at it, we cannot retrieve identifiable information about the original source.

Local Differential Privacy [48, 58] is a model of Differential Privacy that ensures a specific degree of privacy, usually by using a randomized algorithm A to apply noise to the original data [84]. We say that A provides ϵ-local differential privacy if, and only if, for all subsets S of the image of A, and for all pairs of private data x and x':

$$\Pr[A(x) \in S] \leq e^{\epsilon} \times \Pr[A(x') \in S] \tag{2.3}$$

, where the probability is taken from the randomized algorithm. The lower the ϵ, the higher the degree of privacy.

Chapter 3

Related Work

In Blockchain-based Federated Learning (BFL) systems, there are different algorithms that need to be taken into consideration when building the system. In this chapter, we go over these algorithms and their variations used in the literature in order to find the gap that we will try to fill in this thesis.

3.1 Consensus Algorithms

As it can be seen from Table 3.1, various consensus algorithms have been used for BFL systems. Below is a summary of each of these consensus algorithms. In most works, authors chose to use an already existing consensus algorithm, such as Proof of Work [23, 25, 35, 55, 56, 66, 69, 86, 90], Proof of Stake [12, 16, 25, 45–47, 56, 91] and Proof of Authority [37, 50, 71, 89]. In addition, in the majority of the existing works that used an already existing consensus algorithm, the BFL system is built on top of an already existing blockchain platform.

There are works such as [38, 44, 68] that integrate the consensus algorithm of the blockchain with the model training process in order to preserve energy and resource consumption [42]. However, they are either not publicly available or cannot be easily applied to already existing blockchain platforms because they require internal changes.

Finally, even though there are some works that analyze the resource and energy consumption of blockchain consensus algorithm in the context of BFL systems, there is a lack of analysis for already existing consensus algorithms. This is also mentioned by survey papers such as [59, 86]. Therefore, we intend to fill in the gap by providing an impact analysis of different consensus algorithms on already existing blockchain platforms.

3.2 Model Parameter Storage

Another important component of BFL systems is the location, where the model parameters are stored in order to be shared with the servers. According to the literature, the model parameters may either be stored on-chain, i.e., in the blockchain itself, or off-chain, i.e., in a separate storage provider [92].

With the *on-chain storage* [9, 29, 35–37, 71, 85, 89], the smart contract stores the model parameters itself, which means that the parameters themselves will be stored in the blockchain. However, most blockchain platforms have a limit on how large a block can be and, consequently, the amount of data that can be stored per contract is limited [37]. In these cases, smart contracts are chunked, i.e., a single contract is split into many different contracts that hold smaller chunks of the parameters [71]. In addition, this allows for the new model parameters to be directly calculated through the smart contract as the values are directly accessible [37].

With the *off-chain storage* [4, 8, 50, 53, 55, 61, 64, 91], the smart contract holds a reference to the model parameters in some external (decentralized) storage systems. In this case, the new model parameters cannot be calculated directly on the smart contract, as the smart contracts have limited functionality and are not able to download external information during execution. Instead, a set of devices perform the aggregation in parallel and submit their aggregation. Through the smart contract, the majority of the devices must agree on what is the next global aggregation. Whether these devices are the servers or the clients, it all depends on the architecture of the system.

Even though most implementations prefer an on-chain storage, these implementations also use custom blockchain implementations [9, 29, 35, 36, 85], which means that they can implement a platform that has different restrictions on how much data a smart contract can handle. When it comes to using already existing blockchain platforms such as Ethereum, most implementations prefer off-chain storage using a system such as the InterPlanetary File System[4, 8, 50, 55, 64, 91].

3.3 Participants Selection Algorithms

Usually, only some clients are asked to submit a model update in each round. The process of choosing the clients that participate in each round can vary and have different costs. In most works, such as [42, 64, 89], the number of clients and which clients specifically participate are chosen *randomly*. In other systems such as [24, 29], clients are allowed to take initiative, operating in a *first come, first served* basis. The survey [59] mentions that there is a lack of analysis on how the selection of the participants impacts the accuracy of the BFL system, as well as the communication and computation costs.

3.4 Scoring and Aggregation Algorithms

Different solutions can be found in the literature regarding verification techniques. The authors of [16] created a committee-based verification mechanism. To implement it, they deploy a verification smart contract on the blockchain, which periodically elects different clients as committee members. Then, the committee is responsible for voting if the submitted model updates are valid or not.

The authors of [50] implement a verification algorithm based on the trend of the validation error accuracy. To implement it, the model updates of each client are validated using a public validation data set known to both servers and clients. The result of this validation also influences the reward distribution.

Scores-based systems [8, 64, 74, 91], also known as reputation-based systems, are, by far, the most common among the literature and also the only ones that can be applied to already existing blockchain platforms. These work by giving clients with consistently high quality data and updates higher amounts of points. Then, the updates with less points are either rejected, or they have a smaller influence on the aggregation.

In most of these works, as also noted by [59, 83], the costs of scoring algorithms have not been considered. It is important to understand the trade-off between communication and computation costs, and the usage of the different scoring algorithms. The authors of [83] also mention that there is a lack of comparison of the different security models of the system when using different scoring algorithms.

3.5 Privacy Mechanisms

The majority of the reviewed works did not mention which privacy mechanism they used. However, we found Differential Privacy [55, 64, 91] to be the most common mechanism, followed by the Homomorphic Encryption [50, 85]. In addition, the authors of [59] also point out the lack of consideration for how privacy mechanisms may impact the system, both in terms of accuracy, as well as resource consumption. From our review, we can confirm that this is still the case.

3.6 Other Remarks

In addition, there are several works designing BFL frameworks [17, 50, 61, 83], but very few of them provide the source code or build a framework that is intended to be used by others. Providing the source code is extremely important when it comes to reproducibility and verification, so that others can analyze it and do further research using it.

Finally, as it can be seen in Table 3.2, only [56] discusses that it is theoretically possible to implement Vertical Federated Learning in a BFS setting. However, it provides no implementation or design details.

3.7 Conclusions

From the literature review, we can draw the following conclusions. Firstly, there is a clear lack on how different components of a BFL system, such as consensus algorithms, scoring algorithms, number of clients, impact the accuracy, communication and computation costs of the system. Consequently, this work intends to fill in this gap by providing a detailed analysis on how some of these algorithms impact execution time, transaction costs, transaction latency, model accuracy and convergence, communication costs, and computation costs of the system.

Secondly, even though there are many works on designing BFL frameworks, very few are released to the public, or modular. We, therefore, will work on designing and implementing a modular BFL framework that can be easily changed to support new algorithms and will be available to the public to empower future research.

Lastly, to the best of our knowledge, there is only one work [56] that discusses that it is theoretically possible to implement Vertical Federated Learning in a BFS setting, but provides no practical solution.

Paper	Platform					Consensus							
	Ethereum	Hyperledger	EOS	MultiChain	Custom	PoW	PoA	(p)BFT	PoS	PoFL	PoQ	Committee	Other
[55]	✓					✓							
[90]	✓					✓							
[37]	✓						✓						
[71]	✓						✓						
[16]	✓								✓				
[67]	✓			✓								✓	
[78]	✓												
[64]	✓												
[17]	✓	✓											
[4]	✓												
[61]	✓												
[8]	✓												
[89]		✓					✓						
[53]		✓											
[50]			✓				✓						
[35]					✓	✓							
[12]					✓				✓				
[45]					✓				✓				
[44]					✓						✓		
[85]					✓							✓	
[9]					✓								✓
[5]					✓								✓
[24]					✓								✓

Table 3.1: Blockchain Platforms and Consensus Algorithms

Paper	Platform					Consensus							
	Ethereum	Hyperledger	EOS	MultiChain	Custom	PoW	PoA	(p)BFT	PoS	PoFL	PoQ	Committee	Other
[29]					✓								
[36]					✓								
[48]					✓								
[49]					✓								
[86]						✓							
[69]						✓							
[56]						✓			✓				
[66]						✓		✓					
[23]						✓							
[25]						✓		✓	✓				
[91]								✓	✓				
[34]								✓					
[46]									✓				
[47]									✓				
[68]										✓			
[11]										✓			
[74]										✓			
[63]												✓	
[75]												✓	
[93]												✓	
[32]												✓	
[42]												✓	

Table 3.1: Blockchain Platforms and Consensus Algorithms (Continued)

I: IID, N: Non-IID, H: Horizontal, V: Vertical

	Data Distribution	Data Partition	Dataset
[4]	I	H	Breast Cancer Dataset
[5]	N	H	
[9]	N	?	?
[11]	I	H	MNIST, CIFAR10
[12]	?	H	MNIST
[16]	I	H	MovieLens
[17]	I, N	?	?
[23]	I	H	MNIST
[24]	?	H	CIFAR10
[25]	I	H	MNIST
[29]	?	H	?
[33]	I	H	MNIST
[34]	I	H	MNIST
[35]	I	H	?
[36]	N	H	MNIST
[37]	N	H	MNIST
[42]	I	H	FEMNIST
[44]	I	H	Reuters, 20News
[45]	I	H	Uber Pickups, MNIST
[46]	I	H	MNIST
[47]	I	H	CIFAR10
[48]	?	H	?
[49]	I, N	H	?
[50]	?	H	MNIST
[53]	N	H	MNIST, CIFAR10
[55]	?	H	?
[56]	I, N	H,V	?
[61]	I, N	H	?
[63]	I	H	MNIST
[64]	N	H	?
[66]	I	H	?
[67]	I	H	CICIDS 2017
[68]	I	H	CIFAR10
[69]	I, N	H	MNIST, CIFAR10
[71]	I	H	NYC 2018 Taxi
[75]	I	H	LFW, MNIST, CelebA, CASIA
[85]	?	H	MNIST
[86]	?	H	?
[89]	I	H	MNIST
[90]	I	H	Air-Conditioning
[91]	I	H	MNIST
[93]	I	H	?

Table 3.2: Data Distribution, Data Partition and Datasets

Chapter 4

Framework Design and Implementation

In this chapter, we provide detailed information regarding the design of our modular framework, as well as its implementation.

4.1 BlockLearning Framework's Design

The modular framework, to which we called BlockLearning, is designed in such a way that modules can be added, as well as removed or changed, easily. In this framework, the devices, identified by the address of their account in the blockchain, can be classified into three categories: *trainers*, *aggregators* and *scorers*. Additionally, the entity that deploys the contract and is responsible for starting and terminating the rounds is called *model owner*.

A device, i.e., a client or a server, can be categorized as one or more categories. For example, BlockFlow's scoring algorithm is executed by the clients, which are then categorized as trainers and scorers, while the servers are categorized as aggregators. In contrast, Multi-KRUM is executed by the servers, which are then categorized as aggregators and scorers, while the clients are categorized as trainers. By allowing each device to play more than one role, the framework provides flexibility to support different architectures and algorithms.

Figure 4.1: BlockLearning's Execution Flow

The framework supports a modular sequential flow represented in Figure 4.1. This flow is based on the current literature and the steps required to perform the Federated Learning process via the blockchain. The steps are explained below:

1. The model owner initializes the round. During the round initialization, depending on the participant selection algorithm, the trainers that will participate may have been selected already, or not.

2. The trainers retrieve the information such as the global weights from the last round and train the model using their local data. Then, the trainers submit their model updates.

3. If a scoring algorithm is enabled, the scorers retrieve the updates and calculate the scores. Then, they submit their scores.

4. The aggregators retrieve the model updates and execute the aggregation algorithm and submit the aggregation results to the blockchain.

5. Finally, the model owner sends a transaction to the blockchain in order to terminate the round. At this point, the smart contract checks if the majority of the aggregators agreed on the aggregation. If so, the round is marked as terminated. Otherwise, the round fails, indicating that the aggregators did not reach an agreement, which may indicate that some of the aggregators are compromised.

In the last step of the execution flow, the smart contract checks if the majority of the aggregators agree on the aggregation. The majority is defined by *at least 50%*. Therefore, the framework offers a 50% threat model. However, the threat threshold can be changed, changing the threat model.

4.1.1 Structure and Modules

The framework is divided into three main software components: the smart contracts, the library, and the testbed. The structure of the framework, as well as its components and their corresponding modules, is depicted on Figure 4.2. Each of the components plays a different role in the overall system in order to support the logical flow shown in Figure 4.1. In the following subsections, each of the components will be explained in more detail.

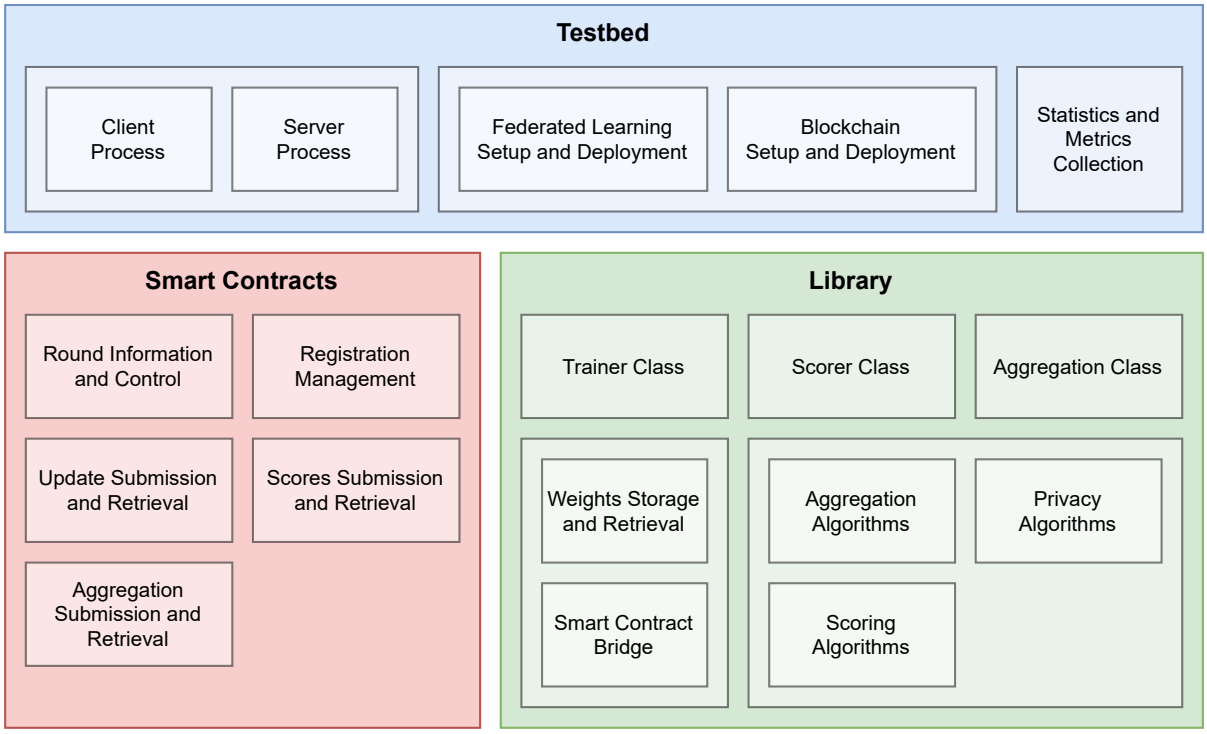


Figure 4.2: BlockLearning's Structure and Modules

4.1.1.1 Smart Contracts

The first component of the framework is the smart contracts. The smart contracts live on the blockchain and are the main means of communication between FL clients and servers. In addition, the smart contracts hold information regarding the current status of the round, as well as the updates, scores, aggregations, among others. The smart contracts provide the following functionality:

- *Round Information and Control*: the smart contract must provide information on whether the round is ongoing and which phase, i.e., scoring, aggregation, or termination phase, it is in. It must allow for flexibility such that new phases can be added in the future, such as the backpropagation confirmation phase we need for our vertical model. In addition, it must allow for rounds to be started and marked as terminated. Round phase advancements are defined through pre-defined conditions that, once met, automatically move the round to the next phase. For example, after all updates are received, the smart contract should move to the next phase.

- *Registration Management*: the smart contract must allow devices to register themselves as trainers, aggregators, or scorers in the system. Finally, the smart contract should provide information about which devices participate in each round.

- *Update Submission and Retrieval*: the smart contract must allow trainers to submit their updates, which must include a pointer to the model weights and the amount of data points that were used to train the model. In addition, it can include the training accuracy and testing accuracy for each individual trainer. The submissions must be accessible.

- *Scores Submission and Retrieval*: the smart contract must allow scorers to submit their scores. It must be possible to know which scorer scored which update and they must be accessible.

- *Aggregation Submission and Retrieval*: the smart contract must allow aggregators to submit the aggregations, which contain a pointer to the weights. The aggregations must be accessible.

4.1.1.2 Library

The second component of the framework is the library. The library encodes the algorithms, utilities, and building blocks necessary to implement the scripts that run on the clients and the servers. It must include:

- *Aggregation, Scoring* and *Privacy Algorithms*: implementation of the different algorithms. For each algorithm type, a common interface must be implemented, such that adding new algorithms is easy and simple and they are interchangeable.

- *Weights Storage and Retrieval*: utilities to load and store weights on the decentralized storage provider. These must also provide an interface in order to make it easy to change the storage provider by providing a different implementation.

- *Smart Contract Bridge*: a contract class that provides an interface to the smart contract that lives on the blockchain. With this class, it should be possible to call

the smart contract functions as if they were local functions.

- *Trainer, Scorer* and *Aggregator Classes*: a class per each device category. This class must register the devices as their category upon initialization. It must also provide methods to execute the training, scoring and aggregation tasks, respectively.

4.1.1.3 Testbed

The third component of the framework is the testbed. The testbed provides the platform to conduct the experiments in a reproducible way, for instance by setting static seeds for randomness. The testbed must include:

- *Client, Server* and *Owner Scripts*: scripts that will be run at the clients, the servers, and at the model owner, respectively. These scripts will use the library in order to perform the right tasks according to which algorithm is being used.
- *Federated Learning Setup and Deployment*: scripts and tools to easily deploy the client and server machines in a test environment, such as containers.
- *Blockchain Setup and Deployment*: scripts and tools to easily deploy the blockchain network in a test environment using the different consensus algorithms, and to deploy the contract to such network.

In addition, the testbed must also include tools to collect the required statistics and logs that can be later processed to retrieve the metrics necessary for the impact analysis.

4.2 BlockLearning Framework's Implementation

In this section, we go over the implementation details of the BlockLearning framework, following the guidelines defined in Section 4.1. The complete implementation is publicly available on GitHub[1].

4.2.1 Smart Contracts

As mentioned previously, we use the Ethereum [87] blockchain platform as it is the most popular and compatible with all the techniques we use for our experiments and comparison with the related work. Therefore, the smart contracts must be implemented in a programming language that supports Ethereum. We chose the Solidity [14] programming language as it is the most well-known with the widest support.

Since our framework supports different algorithms, we need four different smart contracts. These smart contracts inherit most of their functionality from an abstract smart contract that provides the common data structures and functionality, named `Base`. Then, we implement the following classes, that derive from `Base`:

- `RandomSelectionNoScoring`, which is used when we do not need a scoring algorithm. It only adds a new function to the `Base` class in order to allow the model owner to start a round with random participant selection.

[1] `https://github.com/hacdias/blocklearning`

- `RandomSelectionScoring`, which is used when we need a scoring algorithm. This smart contract implements the required methods to support the scoring phase, such as scoring submissions and the scoring round. In addition, it adds a function to allow the model owner to start a round with random participant selection.
- `FirstComeFirstServedNoScoring`, which is used with the first-come first-served participant selection algorithm with no scoring mechanism. It adds a function to allow the model owner to start a round with first-come first-served participant selection.

A class diagram with the public interfaces of the contracts, as well as the data types, is depicted in Figure 4.3. From the diagram, we can see that the smart contract provides round information and control, registration management, updates submission and retrieval, scores submission and retrieval, as well as aggregation submission and retrieval. One may note that the scores submission and retrieval are only implemented in `RandomSelectionScoring` as the remaining smart contracts are not used with scoring mechanisms.

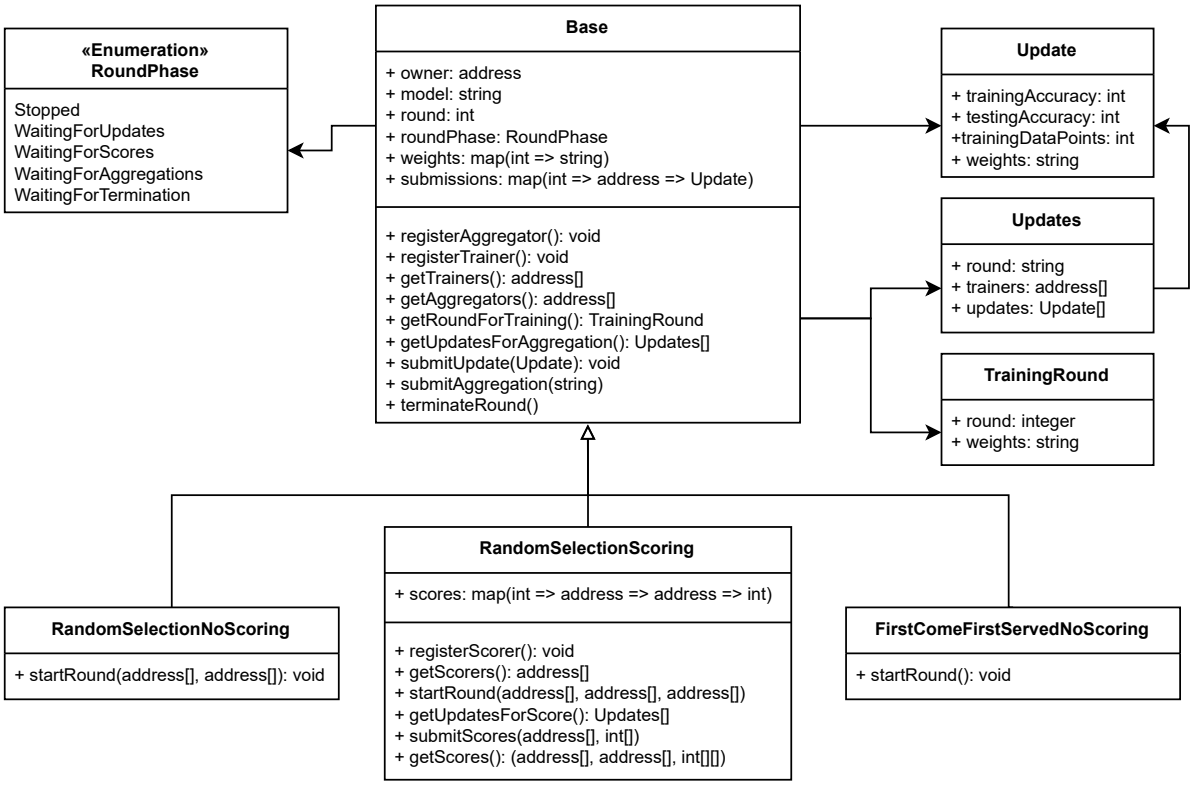


Figure 4.3: Smart Contracts Class Diagram

An interesting implementation detail to note is that score and accuracy values are stored as integers. Currently, Solidity does not support floating point numbers. To preserve fidelity, the original values are multiplied by a large integer, 10^{18}. Then, when the values are retrieved from the smart contract, they are divided by the same value in order to get the original value.

4.2.2 Library

The library is implemented in the Python [81] programming language. The main motivation for using Python is that many well-known Machine Learning libraries, such as TensorFlow [1] and PyTorch [62] are implemented in Python, as well as many data processing tools.

4.2.2.1 Aggregation, Scoring and Privacy Algorithms

The first component of the library is the aggregation, scoring and privacy algorithms. Each of these categories of algorithms has a specific interface to which each algorithm must conform to. By having a common interface, we can implement new algorithms, or change existing ones, easily. The interfaces are as follows:

- `aggregate(trainers, updates, scorers, scores)` → `weights`
  The aggregators must provide a function `aggregate` that receives an array with the trainer addresses, an array with the updates sorted by the same order as the trainers, an array with the scorers and an array with the scores sorted by the same order as the scorers. It is important to note that the scorers and the scores are optional arguments since a scoring algorithm is not always required. The function returns an array with the aggregated weights.

- `score(round, trainers, updates)` → `trainers, scores`
  The scorers must provide a function `score` that receives an integer with the round number, an array with the trainer addresses, as well as an array of updates that are sorted by the same order as the trainer addresses. The function returns an array with the trainers and their submission scores.

- `privatize(x)` → `y`
  The privacy mechanisms must provide a function `score` that receives an array of the weights `x` and returns the privatized weights `y`.

Each of the aggregation and scoring algorithms is implemented based on the algorithms and details provided by the original authors. The local differential privacy is implemented using IBM's `diffprivlib` [28] library.

4.2.2.2 Weights Storage and Retrieval

The second component of the library is the utilities to store and retrieve the weights. The weights storage class also provides a common interface such that it is possible to change which storage provider we use. For our implementation, we use the InterPlanetary File System (IPFS) [6] as our decentralized storage provider since it was used by many of the works reviewed in Chapter 3.

IPFS is a distributed content addressed file system, which implies that, every file is addressed by its content. It works by attributing a hash, based on the file's content. Using this hash, also known as Content Identifier (CID), the file can be retrieved from the network and guaranteed to be immutable. Instead of storing the entire file in the blockchain, the CID can be stored. Pairing IPFS with the blockchain keeps the system decentralized and distributed, while offloading the storage to a different system.

4.2.2.3 Smart Contract Bridge

The third component is the smart contract bridge class. The smart contract bridge is implemented using the `Web3.py` [22] library, which provides utilities to call the functions of the smart contracts. The contract bridge class provides 1:1 functions for each functions of the smart contract.

4.2.2.4 Trainer, Scorer and Aggregator Classes

The fourth and the final component of the library is the `Trainer`, `Scorer` and `Aggregator` classes. These classes implement the main flow of each of these procedures using the modules aforementioned described. For example, the trainer class is initialized with the contract bridge, the weights storage, the model, the data and an optional privacy mechanism. Then, it provides a method `train()` that executes the training procedure. Similarly, the scorer class provides `score()` and the aggregator class provides `aggregate()`.

4.2.3 Testbed

The testbed, that is, the platform to conduct the experiments. It was mostly implemented using the aforementioned library and Docker [52]. Docker is a platform that allows to easily deploy applications in an isolated setting through what is called a container, allowing us to simulate multiple devices in the same network. Each container runs an image, which is the name given to the piece of software than runs on the container.

In the testbed, we have two major components: the client, server and model owner scripts, the federated learning environment deployment and the blockchain deployment. These are discussed on the following subsections.

4.2.3.1 Client, Server and Owner Scripts

The client, server and model owner scripts are the processes that will run at the client, server and model owner, respectively. These are implemented using the BlockLearning library. In each of these scripts, we first load the required data, such as the data set in the clients, and initialize the required algorithms, namely the scoring, aggregation and privacy algorithms.

Then, depending on the scoring algorithm, we initialize the relevant classes at the correct machines. For example, for the BlockFlow scoring algorithm, the client initializes a `Trainer` and a `Scorer`, while the server initializes an `Aggregator`. In contrast, for Multi-KRUM, the client only initializes a `Trainer`, while the server initializes an `Aggregator` and a `Scorer`. On Algorithm 1 you can visualize part of the main loop of the client script.

4.2.3.2 Blockchain Setup and Deployment

The Blockchain setup and deployment is done using already existing tools and our library. As previously mentioned, we use Docker containers in order to run the experiments. Moreover, we use Docker Compose in order to deploy multiple containers at once and orchestrate the deployment process.

Algorithm 1 Client Script Main Loop

Require: $s \in \{\emptyset, \text{BlockFlow}, \text{MarginalGain}\}$
  $T \leftarrow$ Initialize Trainer
  if s is not $\emptyset$ **then**
    $S \leftarrow$ Initialize Scorer
  end if
  while True **do**
    $P \leftarrow$ Get Phase From Smart Contract
    if P is Waiting For Updates **then**
      Execute Training Procedure $T.train()$
    else if P is Waiting For Scores **then**
      Execute Scoring Procedure $S.score()$
    end if
  end while

We use different Ethereum implementations, depending on the consensus algorithm since they are not all available within the sample implementation. Ethereum's main implementation, `go-ethereum` [21], provides PoA and PoW. For QBFT, we use a fork called `quorum` [13], which is mostly identical to `go-ethereum` but supports QBFT.

Moreover, the Blockchain setup and deployment follows the following steps:

1. *Generate Accounts.* In first place, the Ethereum accounts for the clients and servers are generated using the provided `go-ethereum` toolkit. Each account is pre-loaded with 100 ETH, the Ethereum currency, so that clients or servers will not run out of currency to submit their transactions.

2. *Build Images.* In second place, we build the Docker images that will be used to deploy the Blockchain network. This images are based on the images provided by each of the Ethereum's implementations that we use. In addition, they pre-load the account information, as well as some additional configuration to ensure that all nodes are connected when the network is bootstrapped.

3. *Deploy Network.* In third place, the network is deployed using Docker Compose and the configured amount of nodes.

4. *Deploy Contract.* Finally, the contract is deployed to the network using Truffle, which is a tool designed to help developers developing and deploying smart contracts.

Finally, we would like to mention that originally we were planning on testing the PoS consensus algorithm. However, the only fork providing PoS support does not work in private network settings [19]. Therefore, it was not possible to run an experiment with PoS.

4.2.3.3 Federated Learning Setup and Deployment

Similarly to the Blockchain setup and deployment, we also use Docker Compose for the Federated Learning system. The process is identical as in the previous section, except

that we only build the images and deploy the Federated Learning network.

4.2.3.4 Statistics and Metrics Collection

The different components of the library, such as the `Trainer`, `Scorer` and `Aggregator` classes, produce logs. These logs contain information related to timestamps and round number, and events that happen at certain points of the execution, such as: *aggregation started*, *aggregation ended*, *scoring started*, *scoring ended*, among others. These logs are retrieved from the containers using command-line tools implemented into a script called `toolkit.py`.

In addition, resource-related statistics, such as RAM usage, CPU usage, and network traffic, are collected directly from the Docker, through the `docker stats` command.

Chapter 5

Experimental Setup and Evaluation

In this chapter, we provide information regarding the experimental setup and performance evaluation. To be more precise, this chapter explains which data set we use, how the data is partitioned, which models are used in the experiments, the software and hardware specifications of the experimental environment, as well as description of experiments we perform.

5.1 Data Set

The data set used for the experiments is the MNIST [39] data set. The MNIST data set includes 70,000 images of handwritten digits from 0 to 9, where each image is 28 by 28 pixels. Some samples are illustrated in Figure 5.1.

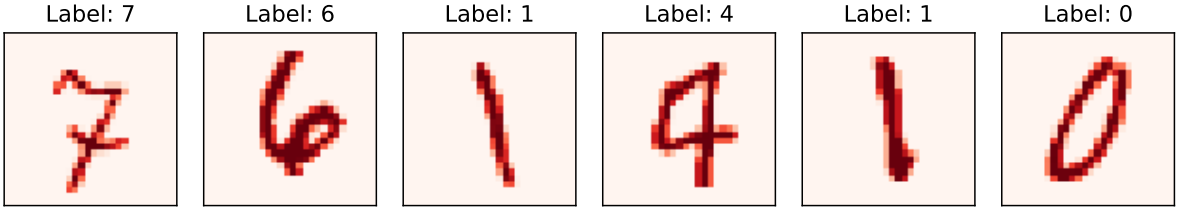


Figure 5.1: MNIST Example Samples

The MINST data set is not only a well-known data set but also widely used by the majority of the reviewed works, as seen in Table 3.2. Therefore, to be able to compare our experiment results with the original works, we use the same data set.

5.2 Client Sampling

The client sampling, that is, the process of dividing the samples among the clients, depends on how data is partitioned across federated learning clients. Data may be partitioned horizontally or vertically in federated learning systems. In the following subsections, we explain how we sample the data for each of the data partitions.

5.2.1 Horizontal

In horizontally partitioned data, as explained in Section 2.1.2, different clients have different samples that share the same feature space. Additionally, in a distributed system, it is expected that the clients are heterogeneous in terms of their computational characteristics and data. Therefore, it is safe to assume that the data distribution in a distributed setting is *non-iid*.

To simulate a *non-iid* distribution, both in terms of number of samples and number of classes at each client, we use the Dirichlet distribution [18, 43]. The Dirichlet distribution, $Dir(\alpha)$, is a probability distribution characterized by its parameter α, which controls the degree of *non-iid*-ness of the distribution. The higher the α, the more identically distributed the data will be. The lower the α, the more likely it is for each client to only hold samples of a single class.

For our experiments, we set $\alpha = 0.1$ in the Dirichlet distribution as it yields a realistic *non-iid* distribution [43], where some clients hold many samples of a few classes, while other clients have few samples of many classes. Moreover, the clients have, on average, 2500 samples each. Some clients have more samples, some have less, simulating a *non-iid* distribution.

In order to perform the horizontal client sampling, we used a publicly available tool [79] that supports sampling from the MNIST data set directly using the Dirichlet distribution. We did so for 5, 10, 25 and 50 clients. Figure 5.2 illustrates the sample distribution for 10 clients. From Figure 5.2, it is possible to see how *non-iid* the distribution is, both in terms of number of data samples and class distribution. For example, client 7 has many samples from classes 2 and 4, while having none of the remaining classes. At the same time, client 10 has a few samples from classes 0, 1, 2, and 9 and many from class 7.

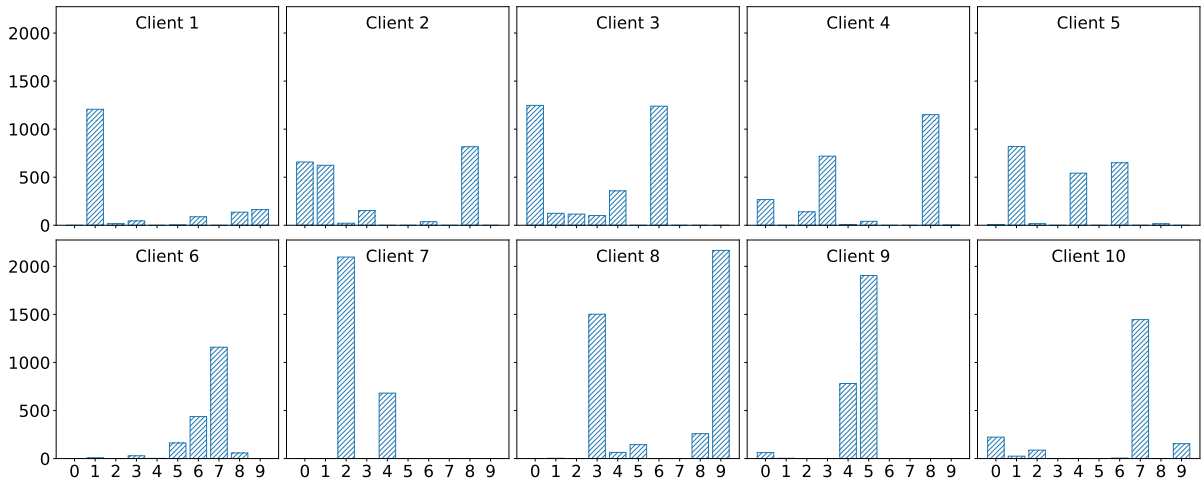


Figure 5.2: Horizontal Data Distribution For 10 Clients

5.2.2 Vertical

Vertically partitioned data is significantly different from horizontally partition data, in the sense that the clients share intersecting sample spaces, but different feature spaces. Therefore, it is not possible to simply divide the samples among the clients.

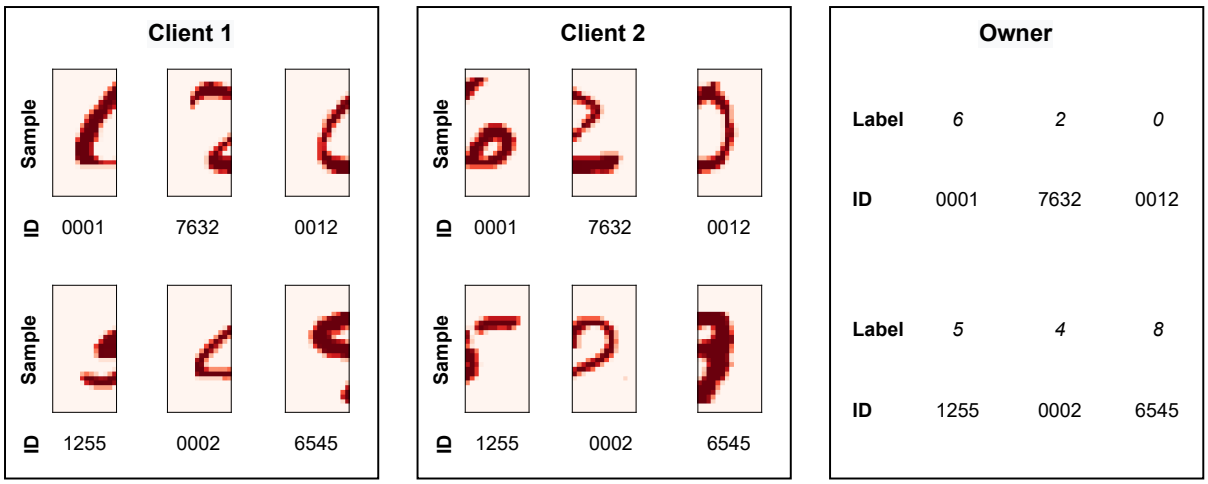


Figure 5.3: Vertical Data Distribution for 2 Clients

For the vertical data partition, we use the work reported in [72]. Firstly, we choose how many samples to assign to each client. We chose 20,000 samples in order to match the original work [72] we will be comparing to. Then, the samples are randomly chosen from the original data set. Subsequently, each sample is assigned a unique identifier (ID) that will be used as label when giving the data samples to each client. Only the servers have access to the ground truth labels. After assigning the IDs, the feature space F will be divided into C parts F_c, where C is the number of clients. Finally, the features F_c, with $c \in C$ will be assigned to each of the clients.

For vertical data partitioning, we divide the data set as in [72] to use it with a Split-CNN [31] model, which will be introduced in the next section. To use this model, the model owner is expected to have the labels, while the clients are expected to have some features of each sample. For the MNIST data set, we can think of the features as vertical fragments of the image. To divide a 28 by 28 image sample between 2 clients, for example, we split the image into two 14 by 28 segments, as depicted in Figure 5.3.

5.3 Machine Learning Models

The models used on this work are simple models found in related work. The goal of this work is not to provide the most efficient or accurate FL model. Therefore, we do not dive into the details of the models. The models used for horizontal and vertical training are succinctly explained below.

5.3.1 Horizontal Model

For the horizontal FL, we use a simple Convolutional Neural Network (CNN) [15] with three levels of convolution intercalated with max pooling to reduce overfitting. These layers are followed by a flattening layer and two dense layers that culminate in the output. The architecture is depicted in Figure 5.4 and more details about its parameters can be found in Table 5.1. To train this model, both servers and clients have the same model. Then, the clients train the the model with their own data set. Subsequently, the clients send the weights to the servers by submitting their local model update to the blockchain

via the smart contract. Finally, the servers aggregate the weights and publish the global model weights through the smart contract.

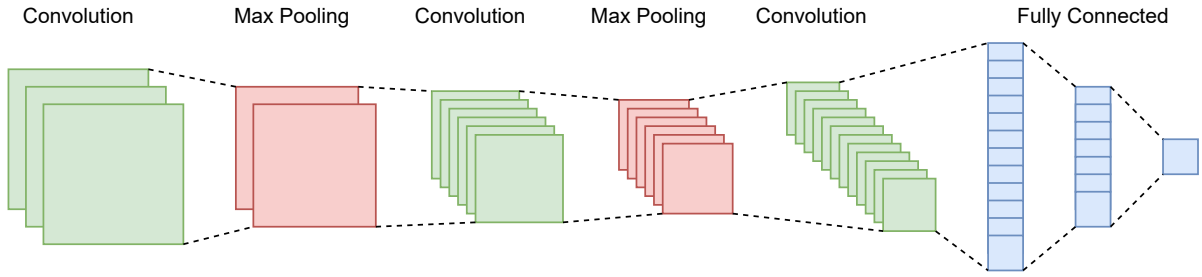


Figure 5.4: CNN Model Architecture

Layer Type	Output Shape
Convolutional 2D	(26, 26, 32)
Max Pooling 2D	(13, 13, 32)
Convolutional 2D	(11, 11, 64)
Max Pooling 2D	(5, 5, 64)
Convolutional 2D	(3, 3, 64)
Flatten	(576)
Dense	(64)
Dense	(10)

Table 5.1: CNN Model Parameters of the Horizontal FL

5.3.2 Vertical Model

For the vertical FL, we use a dual-headed, or four-headed, Split-CNN [31, 72], depending on whether we have two or four clients. The model at the clients is the head model, while the model at the servers is the tail model. To train this model, each client gives its input data to the models and collects the output of the last layer. Then, this intermediate output is sent to the servers, which are then given to the tail model. The servers calculate the gradients, which are then backpropagated to the clients. For more details, please consult the original works where the workings of this model are given in more detail. The architecture is depicted in Figure 5.5 and more details about its parameters can be found in Table 5.2.

Layer Type	Output Shape
Convolutional 2D	(26, 12, 32)
Max Pooling 2D	(13, 6, 32)
Convolutional 2D	(11, 11, 64)
Max Pooling 2D	(5, 2, 64)

(a) Head

Layer Type	Output Shape
2 Input Layers	(5, 2, 64)
Concatenation	(5, 2, 128)
Flatten	(1280)
Dense	(512)
Dense	(256)
Dense	(10)

(b) Tail

Table 5.2: Split-CNN Dual-Headed Model

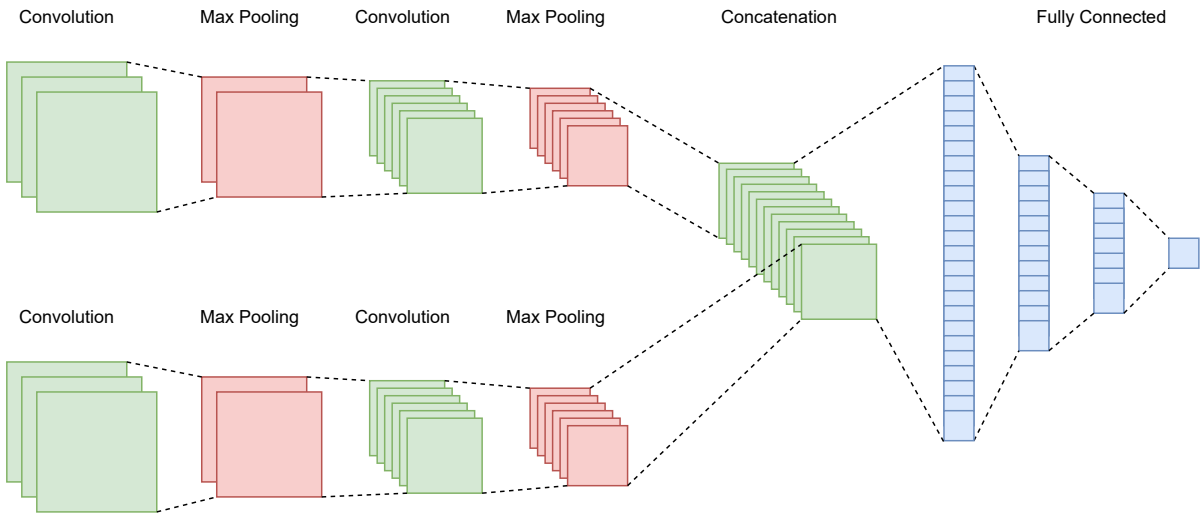


Figure 5.5: Split-CNN Model Architecture

5.4 Hardware and Software Specifications

The experiments were executed in a remote machine, whose hardware and software specifications can be found in Table 5.3. Due to resource limitations, it was not possible to have a machine with GPU. Furthermore, if we consider that FL systems are being run in IoT clients, it is unlikely that such resource-constrained devices would have a GPU available. In addition, the MNIST data set and the models we used are relatively simple, which means that they can be easily trained using CPUs. Nonetheless, it is worth mentioning that the training process would likely be faster in machines with GPUs.

Hardware	Model
CPU	AMD Ryzen 5 3600 6-Core 4.2 GHz
RAM	64 GB
Disk	500 GB NVMe

(a) Hardware

Software	Version
Docker	20.10.15
Docker Compose	2.5.0
Python	3.8.13
Node.js	16.15.0
Truffle	5.5.13
Ganache	7.1.0
Solidity	0.5.16

(b) Software

Table 5.3: Hardware and Software Specifications of Experiments

5.5 Performance Evaluation Metrics

We select the following metrics for our performance evaluation: execution time, transaction costs, transaction latency, model accuracy and convergence, communication costs, and computation costs.

5.5.1 Execution Time

To compare execution time, we define two metrics: the *End-to-end (E2E) Execution Time* and the *Mean Round Execution Time*. The former is defined by the time it takes for an experiment to be executed from start to the end. The latter is defined by the mean time it takes to complete an experiment round, which can be calculated by dividing the E2E Execution Time by the number of rounds of the experiment.

5.5.2 Transaction Costs and Transaction Latency

To compare the blockchain costs, namely the impact of waiting for transactions, we define two metrics: the *Transaction Latency* and the *Transaction Cost*. The former is defined as the mean time it takes between submitting a transaction and it being accepted by the network. The latter is defined by the mean computation effort to execute a transaction, which, in the case of Ethereum, is measured in *Gas*.

Both the transaction latency and transaction cost values are retrieved directly from the Blockchain. Ethereum provides information on how much time transactions take to be accepted, as well as how much each transaction costs. Then, we only calculate the mean.

5.5.3 Model Accuracy

To compare the model accuracy, we use a global *Accuracy* metric for the FL model, where the model owner, that is, the one that initiates the process, has some data set with which it can test the model. The logs produced by the model owner contain the accuracy and are used to extract the accuracy of each round.

5.5.4 Communication and Computation Costs

To compare the communication costs, we define the *Network Traffic Per Round* metric, which is defined by how much network traffic flows to and from each process. It is collected for the client, server, and blockchain processes individually. By knowing the network traffic required for each process, we can draw conclusions regarding how the network traffic impacts different types of devices. For example, if there is a high volume of traffic per round at the client process, and the clients are resource-constrained IoT devices with low network bandwidth, then it is expected that each round takes longer since less traffic can go through the device at a single point in time.

To compare the computation costs, we collect the *RAM Usage* and *CPU Usage*.

The communication and computation costs metrics are collected at the client, server, and blockchain processes in order to be able to differentiate the effects of the different algorithms on the different parts of the system. However, it is important to note that, in practical settings, the server process and the blockchain process run on the same device.

To collect these metrics, we use `docker stats`, which is a command provided by Docker, the platform used for BlockLearning's Testbed. The statistics command provides a live stream of the container's resource consumption, namely the CPU percentage, the RAM memory usage, and the network traffic in and out.

5.6 Experiment Groups

The conducted experiments can be divided into three groups, of which two analyze how using different types of algorithms impact the system's performance in terms of model accuracy, convergence, communication, and computation costs. These two groups relate to impact analysis of:

1. *Consensus Algorithms*: PoA, PoW, and QBFT. Consensus algorithms are a component of the blockchain and they are expected to impact Horizontal Federated Learning and Vertical Federated Learning equally.

2. *Horizontal Federated Learning*

    (a) *Participant Selection Mechanisms*: random selection versus first-come first-served.

    (b) *Scoring Algorithms*: BlockFlow, Multi-KRUM, Marginal Gain, as well as without any scoring algorithm. For each scoring algoritm, we also analyse the impact of:

        i. *Number of Clients*: 5, 10, 25, 50, selected based on the current literature and available resources we have at our disposal to execute the experiments.

        ii. *Privacy Degree*: 1 and 5, as well as without any privacy mechanism.

In the third and last experiment group, we investigate if it is possible to implement and run a Blockchain-based Federated Learning with vertically partitioned data. To do so, we analyze how to extend the BlockLearning framework in order to support the Split-CNN model. Then, we implement the required extensions to the framework. Finally, we execute the experiments and compare the results with the original work where the Split-CNN model was used with the MNIST dataset in order to validate our experiment.

As explained in Section 2.1.2, Vertical Federated Learning systems have an additional step, in which the Private Set Intersection (PSI) of the client's data sets is calculated. In this work, we assume that the PSI is calculated beforehand and that it is already known to all devices. The PSI calculation can be done in different ways and it is its own area of research of Computer Science. The related works we analyzed either did not provide information on how this was calculated, or also assumed that is has been calculated beforehand. In future works, it would be interesting to integrate a PSI mechanism into the framework.

All experiments were performed for 50 rounds so that we can compare the model accuracy results to other papers. This way, we can validate if our experiments are within the expected values. Secondly, all experiments were performed with 10 servers that run both the server process and the blockchain process. Thirdly, all experiments, except for those where the number of clients are compared, are run with 25 clients.

Chapter 6

Impact Analysis of Consensus Algorithms

In this chapter, we analyze the first experiment group, that is, the impact of using different consensus algorithms on the Blockchain-based Federated Learning system. The consensus algorithms are part of the blockchain itself. Therefore, they do not impact horizontal and vertical FLs differently. In this set of experiments, all properties of the system are fixed, except for the consensus algorithm, which varies between PoA, PoW and QBFT.

6.1 Execution Time, Transaction Cost, and Transaction Latency

The first performance evaluation metrics we look into are the mean time it takes for a round to complete, the mean transaction latency, and the mean transaction cost. These values are presented in Table 6.1.

Metric	PoA	PoW	QBFT
E2E Time (m)	18.93	30.62	18.97
Mean Round Time (s)	22.70	36.72	22.74
Mean Transaction Latency (s)	1.549	1.821	1.558
Mean Transaction Cost (Gas)	183124	227052	182880

Table 6.1: Execution Time, Transaction Cost, and Latency of Consensus Algorithms

Regarding time, we can observe that different consensus algorithms can lead to very different execution times. On the one hand, PoA and QBFT are the fastest consensus algorithms, providing the lowest execution times. Additionally, the difference between both is minimal, i.e., only 0.04 seconds per round. On the other hand, PoW takes the longest, being 1.6 times slower than both PoA and QBFT.

Transaction latency and costs follow a similar trend as the execution times. Both PoA and QBFT have similar transaction latency and cost, differing with a small amount, while

PoW has a higher transaction latency and cost. PoW costs are 1.2 times higher than PoA and QBFT.

As explained in subsection 2.2.3, PoW works by solving increasingly complex mathematical equations that consumes high amounts of resources. This intense process can lead to slower response rates, which translates to higher transaction latency and costs. This, in turn, increases the time it takes for each round to complete.

6.2 Model Accuracy and Convergence

The model accuracy, as well as the convergence, as it can be seen in Figure 6.1, does not change significantly per consensus algorithms. Consensus algorithms determine the order, at which the transactions are processed and ensure consistency between the multiple blockchain nodes. This only affects the blockchain internal processes, and not the ML process. Therefore, it was not expected that the consensus algorithms would have an impact on the model accuracy.

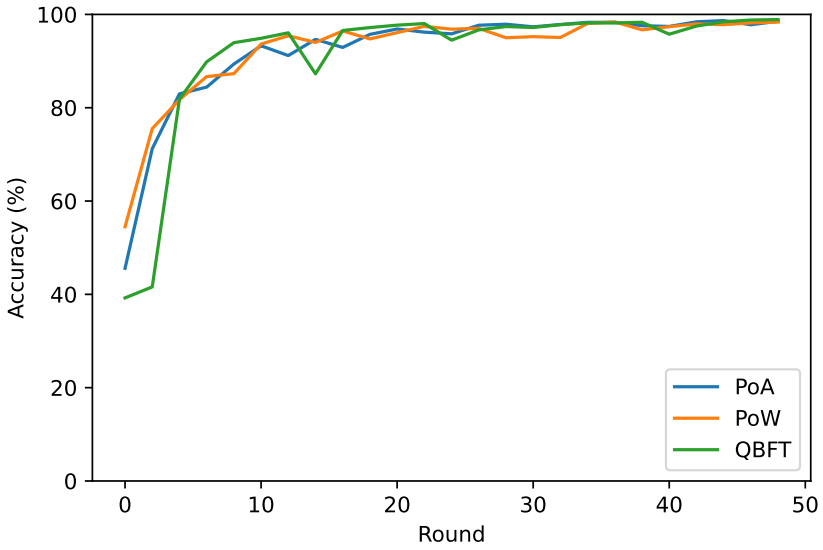


Figure 6.1: Accuracy Per Consensus Algorithm

6.3 Communication Costs

For the communication costs, we analyze the inbound and outbound network traffic per round at the client, server, and blockchain processes. These values can be observed in Figure 6.2.

On one hand, the inbound and outbound traffic for the clients and server have negligible differences when using different consensus algorithms. As mentioned in the previous section, the consensus algorithms have no expected impact on the ML process. Since the clients and servers only concern the ML process, it was expected that the clients and servers would not be affected. The small differences, in the order of < 2 MB, we can observe on the servers are likely related to fluctuations in the random participant selection algorithm. If more participants are being selected, more data needs to be transmitted,

and vice-versa. In this case, 17.9, 18.3, and 17.7 clients participated in each round, on average, for PoA, PoW and QBFT, respectively.

On the other hand, the traffic at the blockchain process varies considerably depending on the consensus algorithm used. On average, PoW requires more bandwidth per round than PoA, but the difference is minimal. However, QBFT requires 2 times more network traffic than PoW and 4 times more traffic than PoA. QBFT is a three-phase consensus algorithm, which requires a higher number of network messages to be transmitted before reaching a consensus. In addition, the size of the messages also differs. When combining both of these aspects, we can conclude that the expected network traffic per round using the QBFT algorithm would be higher, as verified.

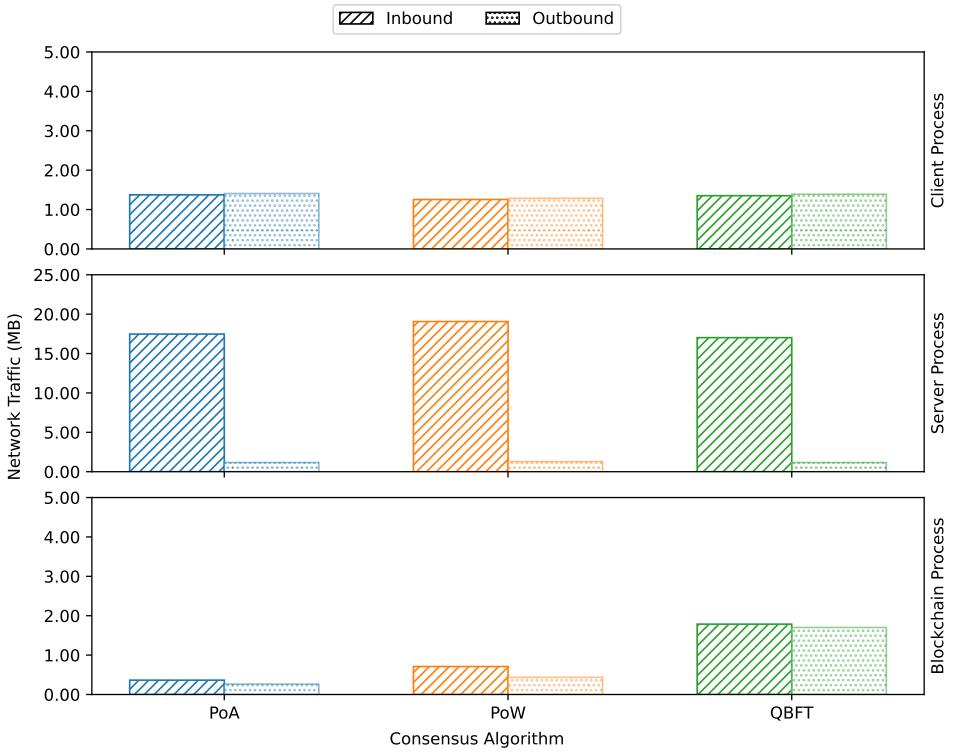


Figure 6.2: Network Traffic Per Round Per Consensus Algorithm

6.4 Computation Costs

Regarding computation costs, we look at both RAM and CPU usage on the client, server, and blockchain processes. Figure 6.3 and Figure 6.4 show the mean RAM usage and mean CPU usage, respectively, per consensus algorithm during the execution of the experiments. As mentioned previously, the execution times for PoA and QBFT are lower than for PoW, which can be seen in the figures by not showing more data past minute 19. Additionally, as explained before, the consensus algorithms are not expected to have a direct impact on the clients or the servers.

Regarding the clients, the RAM usage and CPU usage do not differ significantly regardless of which consensus algorithm is used. From the RAM usage, we observe that the clients reach the same peak. However, the rate at which that peak is reached is different. For PoW, since there are higher transaction latencies, it takes longer to reach the next round,

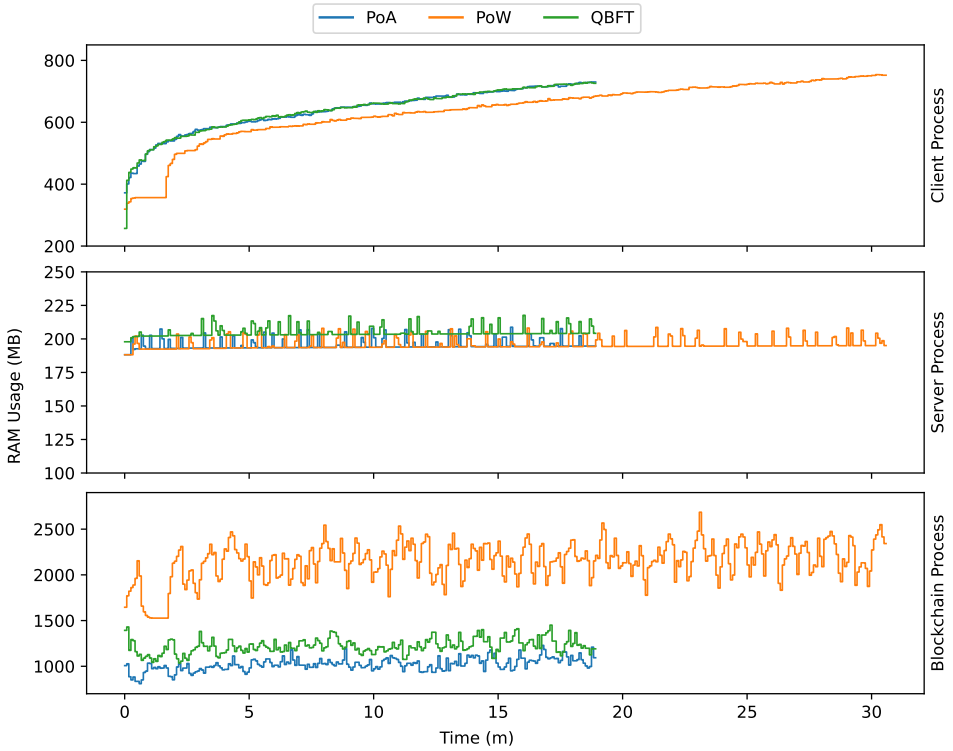


Figure 6.3: RAM Usage Per Consensus Algorithm

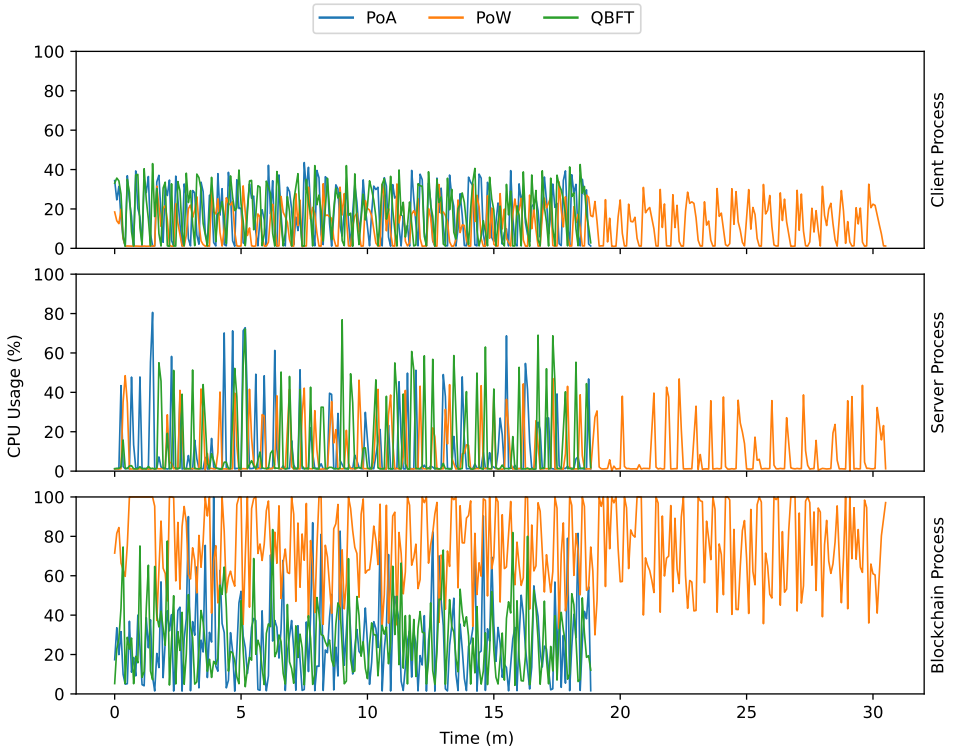


Figure 6.4: CPU Usage Per Consensus Algorithm

leading to a slower growing RAM usage during the model training phase. On the other hand, from the CPU usage, we observe that there are more idle moments, that is, moments at which the CPU usage is lower. This can also be explained by the higher transaction latencies, during which the clients cannot do anything other than wait for the transaction to be approved.

Regarding the servers, the same reasoning as for the clients can be applied. While the RAM usage is consistent across different consensus algorithms, we notice that when using QBFT, the RAM usage at the servers is slightly higher. However, this is a likely negligible difference, as the difference is minimal (< 10 MB) considering the total RAM Usage (≈ 200 MB). The CPU usage is similar to what we observed for the clients, with a higher amount of idle moments.

Regarding the blockchain process, we observe a larger difference, both in terms of RAM and CPU usages. For both, PoW consumes a much higher level of resources. On average, PoW consumes 2 times more then RAM and has a 2 times higher CPU usage. This can also be explained by the way PoW works by solving complex mathematical equations, which require intense computation resources.

6.5 Conclusions and Improvements

In conclusion, we can observe that different consensus algorithms have no direct impact on the model accuracy and computation and communication costs at the clients and servers. However, they have an impact on the time and computation and communication costs at the blockchain process. PoA and QBFT are much faster than PoW. In addition, they require less computation power, both in RAM and CPU. However, QBFT consumes incurs more communication costs than both PoA and PoW. Therefore, there is a clear correlation between the computation costs and the time it takes. The higher the communication costs, the higher the transaction latencies, which translates to slower round times.

Assuming that the blockchain network is only being used for FL, PoA is the most cost-effective of the consensus algorithms we analyzed.

It is also worth pointing out that, as discussed in Chapter 3, PoA is criticized for not being as decentralized as the other algorithms due to the way the validator nodes are chosen. If a higher degree of decentralization is required, as in a public network for example, PoW and QBFT may be better options. There is therefore a trade-off between the degree of decentralization and the communication and computation consumption.

For future work, it would be interesting to study if the Ethereum blockchain could be adapted to easily incorporate the custom consensus algorithms that were seen in Section 3.1. These algorithms work by producing proofs directly from the Machine Learning process. This can lead to better usage of resources if the blockchain network is solely used for a FL system.

Chapter 7

Impact Analysis of Participant Selection and Scoring Algorithms in Horizontal Blockchain-based Federated Learning

In this chapter, we analyze the impact of using different participant selection and scoring algorithms on a Blockchain-based Federated Learning (BFS) system with horizontal data partition. In addition, for each scoring algorithm, we analyze how they behave and how the system is impacted by different number of clients, as well as privacy degrees. Due to the high number of plots, the communication and computation costs plots are placed at the end of the chapter.

7.1 Participant Selection Algorithms

In this set of experiments, all properties of the system are fixed, except for the participant selection algorithm, which can be either random selection or first-come first-served.

Both algorithms choose the number of clients in the same way, via a uniform random distribution. However, the clients themselves are chosen differently. Consequently, it is possible that the distribution of chosen clients is slightly different, which may affect the system performance. Figure 7.1 illustrates client participation (represented by the bars), as well as the average number of participation per client (represented by the lines). It is clear from the plot that the distributions are different.

On one hand, random selection presents a more uniform distribution, where each client was selected a similar number of times. On the other hand, first-come first-served presents a more skewed distribution, where some clients, such as client 1, participate many times and others, such as client 12, participate very few times. To support this observation, we calculated the standard deviation. For the random selection, the standard deviation is approximately 2.49, while for the first-come first-served it is 11.84.

From this observations, we can conclude that, by letting clients take initiative to join a round, similarly to what happens in first-come first-served, it is possible that some will end up participating more than others. By participating more often, the clients will have

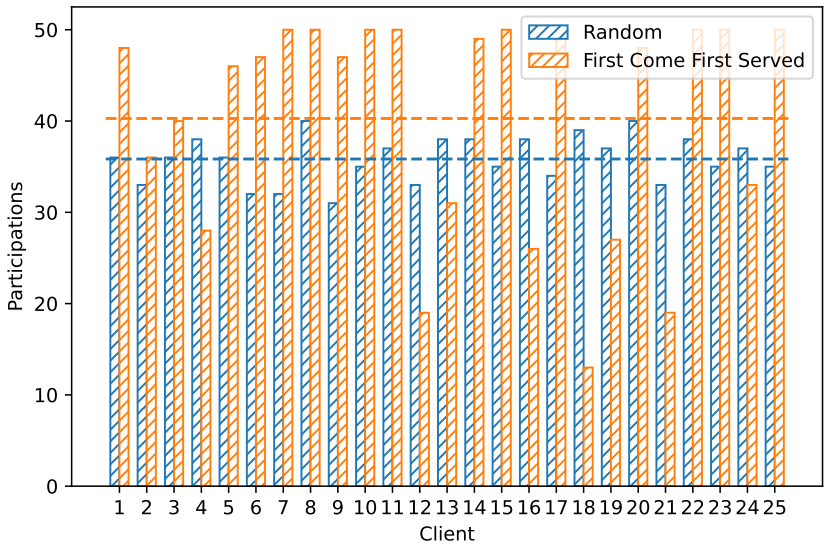


Figure 7.1: Participation of Each Client Per Selection Algorithm

more influence on the global model, which can lead to skewed results.

7.1.1 Execution Time, Transaction Cost, and Transaction Latency

Regarding execution time, it can be seen from Table 7.1 that both algorithms only differ in approximately 1.2 minutes, which translates to a difference of 1.1 seconds per round. In addition, the random participation was slightly faster than the first-come first-served. This negligible difference is likely caused by the fact that, in random selection, less clients participated on average per round, as shown in Figure 7.1. With slightly less clients, we expect that a round takes slightly less time.

	First Come First Served	Random
E2E Time (m)	19.70	18.93
Mean Round Time (s)	23.62	22.70
Mean Transaction Latency (s)	1.560	1.549
Mean Transaction Cost (Gas)	189179	183124

Table 7.1: Execution Time, Transaction Cost, and Transaction Latency Per Participant Selection Algorithm

7.1.2 Model Accuracy and Convergence

As it can be seen from Figure 7.2, even though both algorithms reached identical model accuracy values at the last round, the random selection was more stable during the initial 20 rounds. This can be explained by the fact that the distribution of clients participating in each round with the random selection was closer to a uniform selection. Since the data is *non-iid*, by having the same clients participate repeatedly, the model can become skewed towards their data. However, after 20 rounds, the majority of the clients had the opportunity to participate in the model, which explains why it became more and more stable.

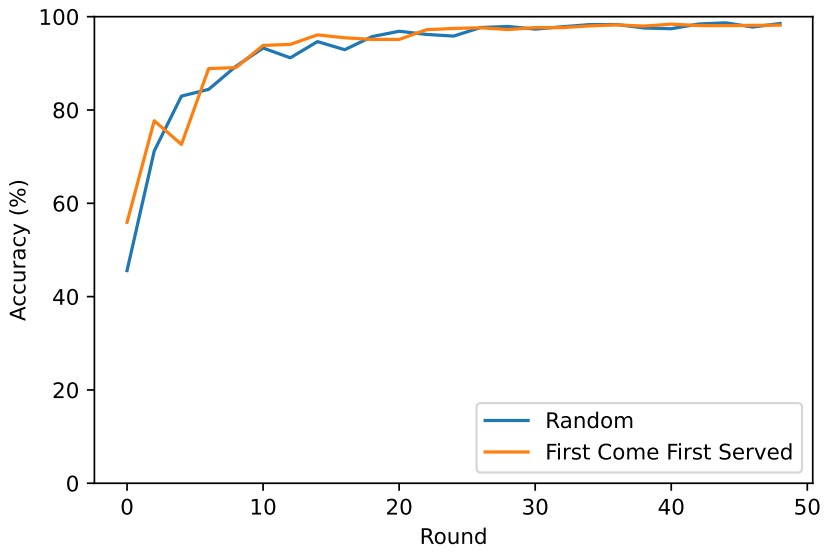


Figure 7.2: Model Accuracy Per Participant Selection Algorithm

7.1.3 Communication Costs

Figure 7.3 illustrates the network traffic per round for client, servers, and the blockchain processes. This algorithms solely influences which and how many clients are selected per round. Therefore, the individual communication traffic of each process is not expected to change significantly as long as the number of participants per round does not change significantly. In these experiments, we observe form Figure 7.1 that the difference of clients participating per round is smaller than 5. This small difference explains why the inbound traffic at the server process is slightly higher, since the server has to download weights from a higher number of clients in order to perform the aggregation.

7.1.4 Computation Costs

Figure 7.4 and Figure 7.5 show the RAM usage and CPU usage per each process, respectively. Similarly to what was observed regarding the communication costs, the RAM usage at the server is slightly higher, which is explained by the additional weights to perform the aggregation. One may note that this difference happens due to the randomness of the process. In other runs, it is possible that the number of selected devices would be lower and therefore the difference would be smaller.

7.1.5 Conclusions

From this set of experiments, we conclude that random selection performs better in terms of fairness of selection, that is, every client is given an equal chance of participating during the training process. In systems with *non-iid* data, it is important to give all clients a chance to participate such that the model is trained with the most diverse data in order to produce the best results. Use of the random selection can ensure that the most amount of data is seen from most clients.

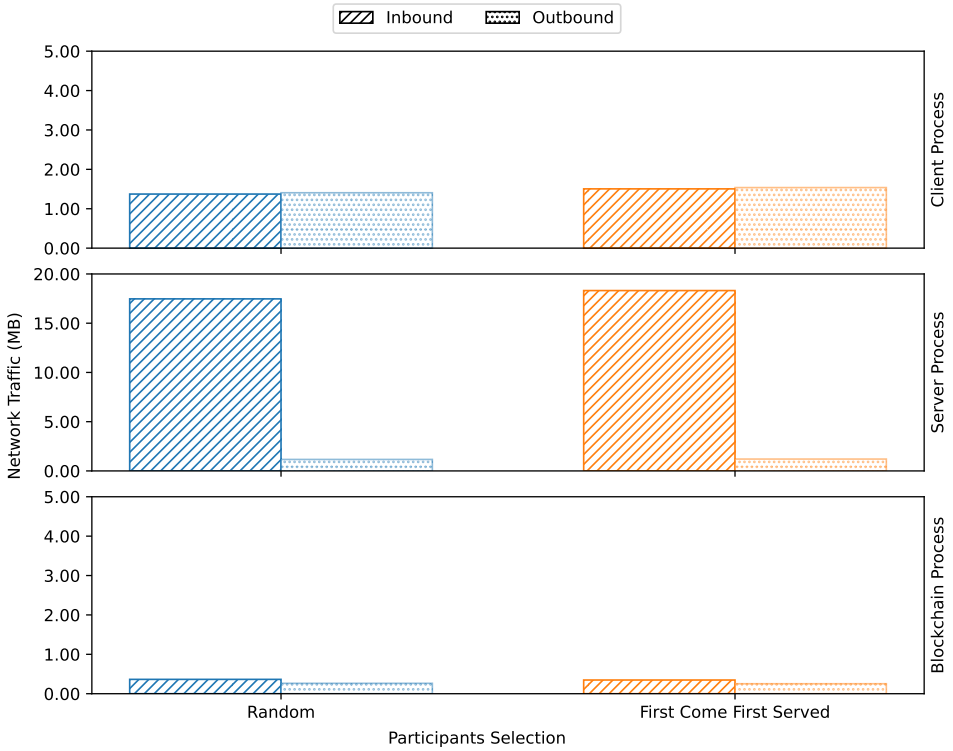


Figure 7.3: Network Traffic Per Round Per Participant Selection Algorithm

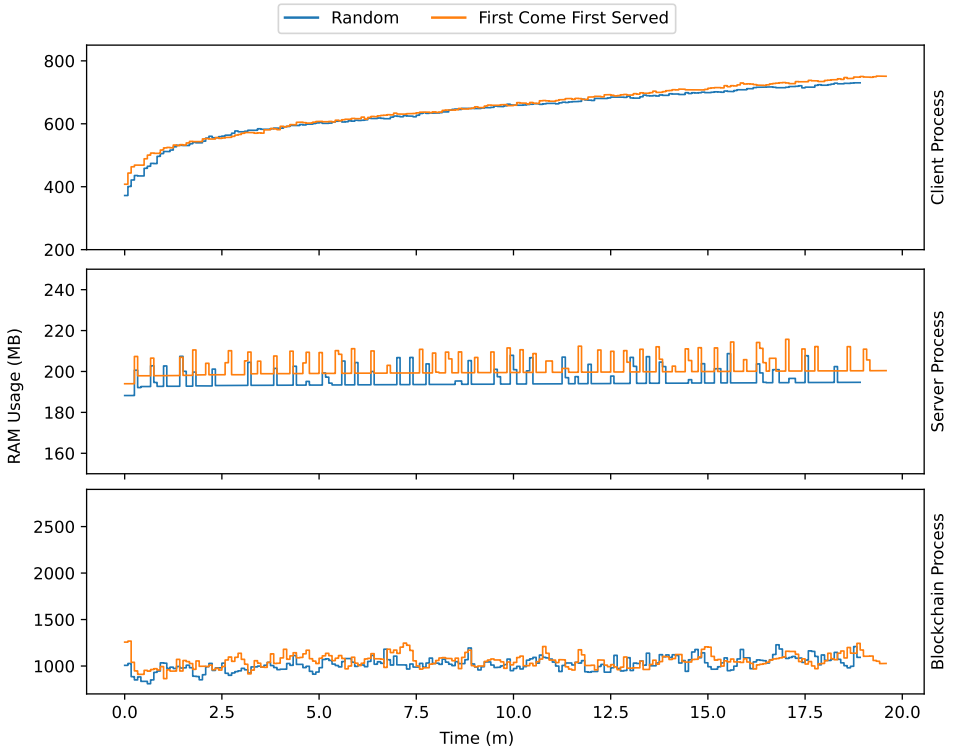


Figure 7.4: RAM Usage Per Participant Selection Algorithm

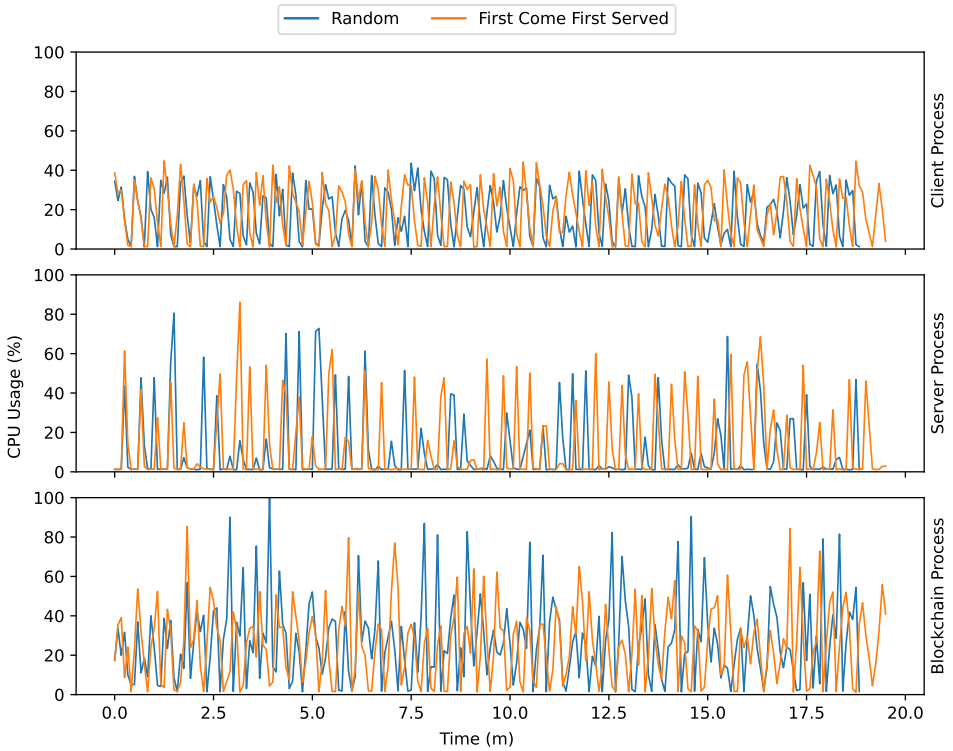


Figure 7.5: CPU Usage Per Participant Selection Algorithm

7.2 Scoring Algorithms

In this set of experiments, all properties of the system are fixed, except for the scoring algorithm, which varies between BlockFlow, Marginal Gain, Multi-KRUM, or no scoring algorithm. Then, we analyze the impact of using different numbers of clients, as well as different privacy degrees.

7.2.1 Overall Comparison

We first compare the impact of the different scoring algorithms on the overall system without varying the number of clients or the privacy degree. This will let us draw some initial conclusions about the different scoring algorithms that may help explain differences observed in the remaining experiments.

7.2.1.1 Execution Time, Transaction Cost, and Transaction Latency

As it can be seen from Table 7.2, every scoring algorithm has different execution time and transaction cost. Firstly, one may notice that not using a scoring algorithm provides the fastest execution time as well as the lowest transaction cost. Both of these observations are explained by the fact that scoring algorithms require more transactions to submit the scores. Overall, the fastest scoring algorithm is Multi-KRUM, taking around 31 seconds per round, while both BlockFlow and Marginal Gain are the slowest, taking both around 49 seconds per round.

Secondly, we observe that BlockFlow and Marginal Gain not only take the longest, but also have similar execution times. As explained in Section 2.3.2, BlockFlow and Marginal

	None	BlockFlow	Marginal Gain	Multi-KRUM
E2E Time (m)	18.93	40.95	41.38	26.25
Mean Round Time (s)	22.70	49.11	49.64	31.48
Mean Transaction Latency (s)	1.549	1.564	1.577	1.573
Mean Transaction Cost (Gas)	183124	339645	257686	280733

Table 7.2: Execution Time, Transaction Cost, and Transaction Latency Per Scoring Algorithm

Gain scores are computed by the clients, whereas Multi-KRUM scores are computed by the servers. Since the number of clients is higher than the servers, which are fixed, there are more devices performing scoring computations with BlockFlow and Marginal Gain. With more devices submitting scores, there are more transactions being submitted to the blockchain, leading to higher execution times. Therefore, it is expected that algorithms that run on the clients, such as BlockFlow and Marginal Gain, take longer than algorithms that run on the servers, such as Multi-KRUM.

Thirdly, we observe that the transaction latency is not influenced by the scoring algorithms. As it can be seen form Chapter 6, the transaction latency is mostly affected by the blockchain consensus algorithms, which, in these experiments, is fixed. In contrary, the transaction costs vary per scoring algorithm. Scoring algorithms that have more devices involved, such as scoring algorihtms executed by the servers, namely BlockFlow and Marginal Gain, have higher transaction costs. Since transaction costs work on a "supply and demand" basis, it is expected that the more transactions are required, the higher the cost will be.

7.2.1.2 Model Accuracy and Convergence

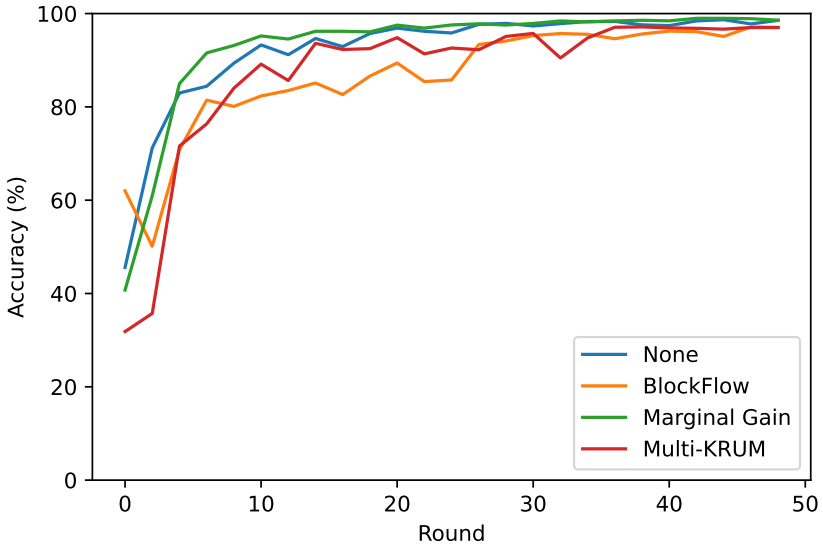


Figure 7.6: Model Accuracy Per Scoring Algorithm

As it can be seen from Figure 7.6, all scoring algorithms reached a high model accuracy of at least 97%. However, some algorithms reached higher accuracy values faster than others, that is, some converge faster. Overall, Marginal Gain converges the fastest, followed by no scoring, then by Multi-KRUM and lastly by BlockFlow.

The BlockFlow, is not only the slowest converging scoring algorithm, but also the only algorithm that does not reject submissions when aggregating, as explained in Section 2.3.2. By not rejecting submissions, but still giving them a score to be used during the weighted aggregation, worse submissions are always included in the global model, which can lead to lower convergence rates.

The Marginal Gain and Multi-KRUM algorithms both reject the worst submissions in each round and only consider the best. While the Marginal Gain uses its own score for the aggregation, the Multi-KRUM uses the number of samples of each submission, similar to the case when no scoring algorithm is used. For this reason, the Multi-KRUM algorithm convergence resembles the one of no scoring algorithm, while the Marginal Gain has a smoothest convergence curve.

7.2.1.3 Communication Costs

Communication costs also vary massively depending on which scoring algorithm is used. Some place more strain on the clients, where others place more strain on the servers. Figure 7.7 presents the network traffic per round per scoring algorithm on the clients, servers, and blockchain processes.

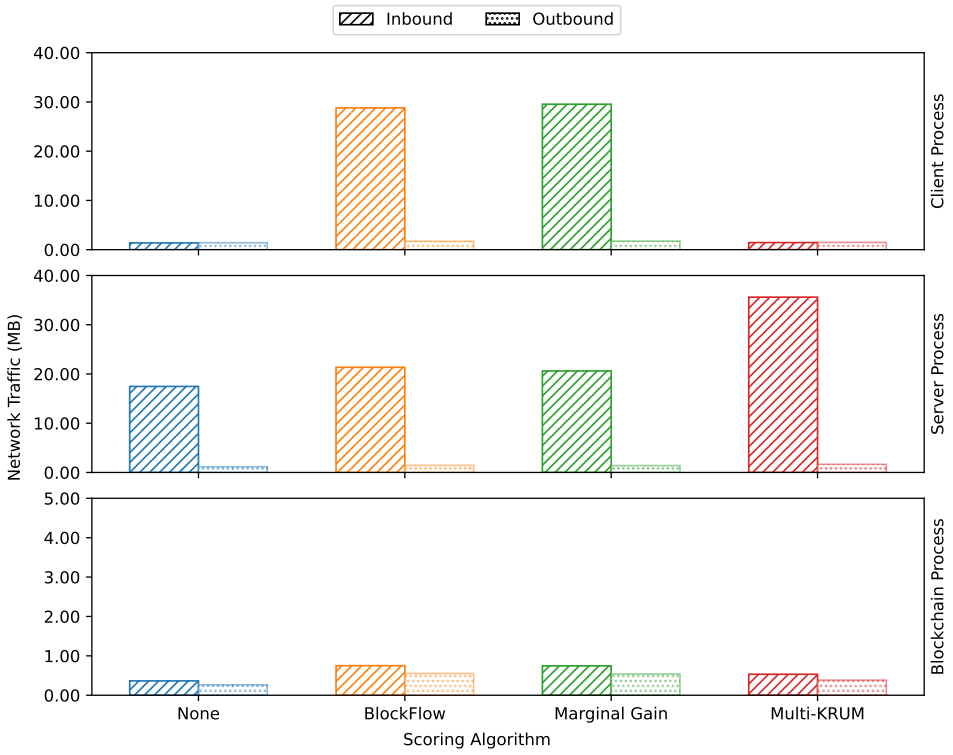


Figure 7.7: Network Traffic Per Round Per Scoring Algorithm

Regarding the clients, there is a large difference in terms of the network traffic when comparing no scoring algorithm and the Multi-KRUM with the BlockFlow and Marginal Gain. On the one hand, the Multi-KRUM has similar traffic requirements to using no scoring algorithm on the clients because the scoring algorithm is executed on the servers. On the other hand, the BlockFlow and Marginal Gain have higher inbound traffic at the clients, while the outbound traffic remains similar. This can be explained by the fact that both BlockFlow and Marginal Gain algorithms are executed by the clients. Consequently,

each client has to download the weights from all other clients in each round in order to calculate the score, leading to a higher inbound traffic.

Regarding the servers, there is not much difference between algorithms, except for the Multi-KRUM, which calculates the scores on the server. Therefore, the server downloads the weights of each client and requires additional time to calculate the scores, compared to the remaining algorithms that only download the weights once for the aggregation.

Finally, regarding the blockchain, we observe that when using the BlockFlow and Marginal Gain algorithms that there is a higher network traffic. The Multi-KRUM also requires more traffic than no scoring algorithm, but not as much as the BlockFlow or Marginal Gain. Since the number of clients is higher than the number of servers, there are more transactions when the scoring algorithm runs on the clients. When there are more transactions per round, there is more activity in the blockchain, leading to more network traffic per round.

7.2.1.4 Computation Costs

The computation costs across the servers and clients follow a similar trend to what we have seen with the communication costs. Figure 7.8 and Figure 7.9 show the RAM and CPU usages on the client, server and blockchain processes, respectively.

Regarding the clients, all algorithms require similar amounts of RAM. The algorithms that run on the client, i.e., the Marginal Gain and BlockFlow, consume slightly more RAM, due to having more weights stored in memory, but the difference is negligible when compared to the total amount of RAM they consume. This can be explained by the fact that the weights are relatively small ($\approx$ 2 MB) compared to the total RAM necessary to train a model. With respect to the CPU usage, it can be seen that the algorithms that run on the client, i.e., BlockFlow and Marginal Gain, have the lowest CPU idle time on the clients, while taking longer to be executed. This shows that calculating the scores on the client implies consistently higher CPU usage on the clients for longer periods of time.

Regarding the servers, it is clear that Multi-KRUM, being the only scoring algorithm that runs on the server, requires higher amount of RAM. However, the difference ($\approx$ 15 MB) is not significant when considering that the servers have large amount of resources at their disposal. In addition, the Multi-KRUM also shows higher level of CPU usage, with frequent spikes to 100%.

Finally, regarding the blockchain, the difference of CPU and RAM usages among the different scoring algorithms is negligible. Even though the blockchain receives more transactions in total, it does not reflect on the RAM and CPU usage. The blockchain, by itself, already produces blocks at a constant rate. Therefore, the number of transactions required for the BFL system does not change significantly the CPU or RAM usage.

7.2.1.5 Conclusions and Improvements

In conclusion, scoring algorithms that are executed on the clients, i.e.,the Marginal Gain and BlockFlow, have a higher impact on the overall system, leading to longer experiment execution times and higher resource usage for the clients. In contrary, algorithms that are executed on the servers, i.e., the Multi-KRUM, have a higher impact on the servers.

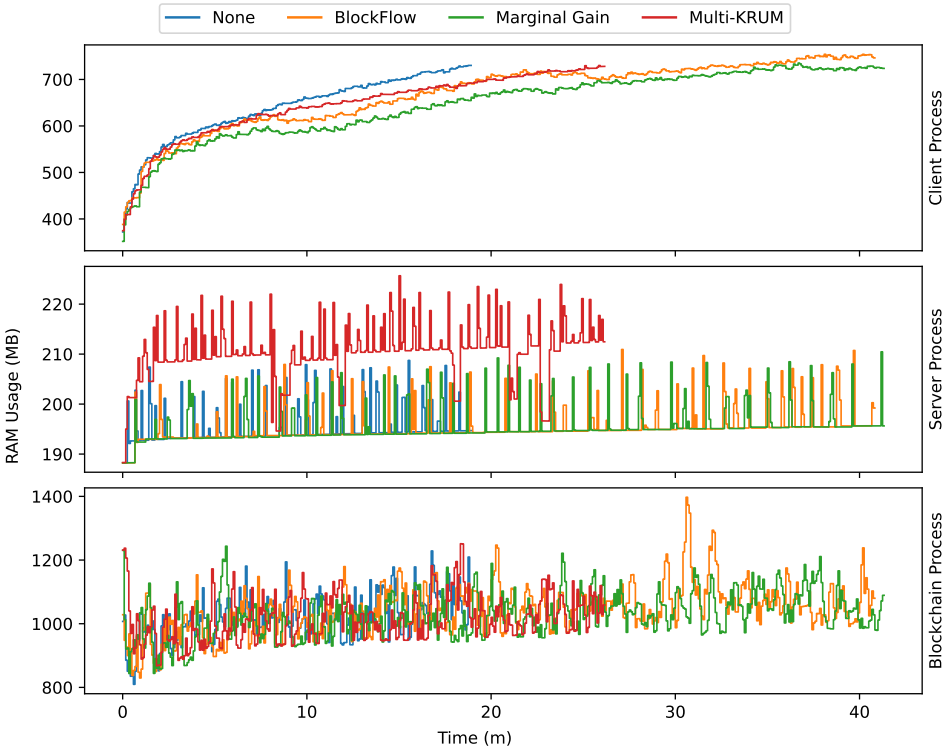


Figure 7.8: RAM Usage Per Scoring Algorithm

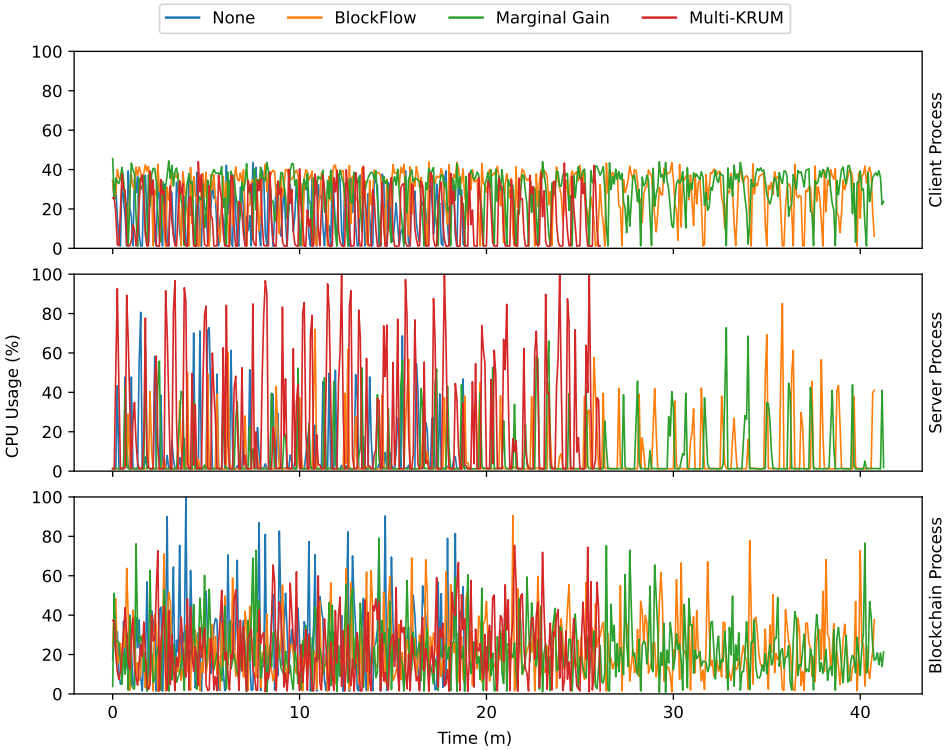


Figure 7.9: CPU Usage Per Scoring Algorithm

Since there is usually a much higher number of clients than servers, the scoring algorithms executed by the servers have less impact on the overall system than the former. Additionally, the Marginal Gain was the most well performing algorithm in terms of model accuracy and convergence speed, followed by the Multi-KRUM and the BlockFlow.

If we had to choose an algorithm, the choices come down to the priorities of the system. If we are working with a system with resource-constrained devices, such as IoT systems, it is important that the impact on the clients is low. Therefore, scoring algorithms that run on the server, such as the Multi-KRUM, are more valuable. If the opposite is true, or if the resource consumption at the client is not relevant, the Marginal Gain could be chosen as it provides the best accuracy of the three.

It is also important to mention that algorithms that require more network traffic per round may be slower on clients with low bandwidth, which is the case of many IoT networks. With lower bandwidths, less traffic can go through at any point in time. Therefore, in case of high network traffic required during a round, devices with low bandwidth can make the process slower.

As a future improvement, servers can cache the client's update weights. Specifically, in case of the Multi-KRUM algorithm, the servers can download each of the client's submission twice: one time for scoring, one time for aggregating. However, the weights downloaded both times are the same as they are part of the same round. Therefore, caching can work well in favor of reducing the network traffic at the server for scoring algorithms that are executed by servers.

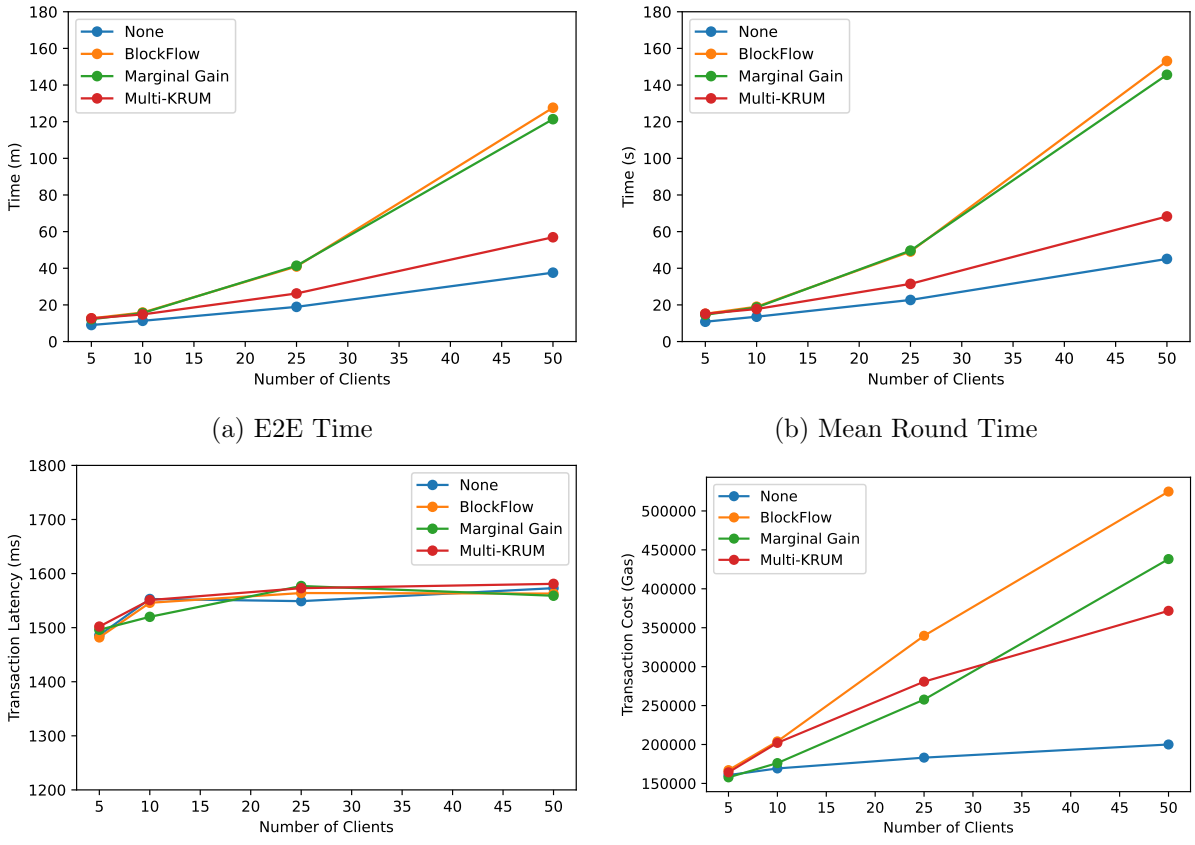
7.2.2 Number of Clients

In this section, we analyze the impact of different numbers of clients on the scoring algorithms. For this comparison, all properties of the system are fixed, except for the amount of clients, which varies between 5, 10, 25 and 50, per each scoring algorithm.

7.2.2.1 Execution Time, Transaction Cost, and Transaction Latency

Figure 7.10 illustrates the execution times, as well as the transaction latency and costs. The execution times of all scoring algorithms increase with the number of clients. However, they do not increase the same way. The Multi-KRUM algorithm, as well as not using any scoring algorithm, have a smaller execution time increase with the number of clients, when compared to BlockFlow and Marginal Gain. This can be explained by the fact that, in BlockFlow and Marginal Gain, the scorers are the clients. Since, as previously discussed, there are more clients than servers, the smart contract has to wait for more scorers to submit their scores, than it would have to wait when using Multi-KRUM. Consequently, the execution time increase with the number of devices is higher with scoring algorithms executed by the clients.

Regarding the transaction latency, there is no significant change with the variation of number of clients. As discussed before, transaction latency is mostly determined by the capacity of the network to handle the transactions. The Ethereum has a relatively fixed capacity of transactions per second of around 15 transactions per second. Our system alone does not reach that rate of transactions, not influencing the transaction latency. It is worth pointing out that, if the number of clients was in the order of hundreds, leading



(a) E2E Time

(b) Mean Round Time

(c) Mean Transaction Latency

(d) Mean Transaction Cost

Figure 7.10: Execution Time, Transaction Cost, and Transaction Latency Per Number of Clients

to elevated number of transactions, the latency would likely increase.

Regarding the transaction costs, we observe that they increase with the increased number of clients. In addition, the transaction costs growth is similar per algorithm. Firstly, we can calculate how many transactions we incur per algorithm per round by $T + A + S$, where T is the number of trainers, usually clients, A is the number of aggregators, usually servers, and S the number of scorers, which can be either the clients or servers. When no scoring algorithm is used, the system only requires $T + A$ transactions, where T is the number of clients and the only growing variable. With the Multi-KRUM, the system requires $T+2A$ transactions, since $S = A$, leading to a faster growth than when no scoring algorithm is used. Finally, both BlockFlow and Marginal Gain algorithms require $2T + A$ transactions and since $T > A$ and T is the number of growing clients, it is also expected that the transaction cost would increase more than for the Multi-KRUM.

7.2.2.2 Model Accuracy and Convergence

As it can be seen from Figure 7.11, the model accuracy and how it converges varies differently with different scoring algorithms. Overall, we observe that lower number of clients leads to a less stable convergence, represented by the spiking in the model accuracy plots. With a low number of clients, such as 5, the selected number of clients per round is also low due to the random way the participant selection algorithms chooses how many clients participate per round. Therefore, the model is trained with less diverse data per

round. Consequently, the model can skew at some points during training, leading to a less stable convergence. In contrary, using more clients, namely 25 and 50, has an overall positive effect on both convergence stability and convergence speed. This is also likely related to the fact that the model is trained with more diverse samples from more clients in each round, leading to a better results.

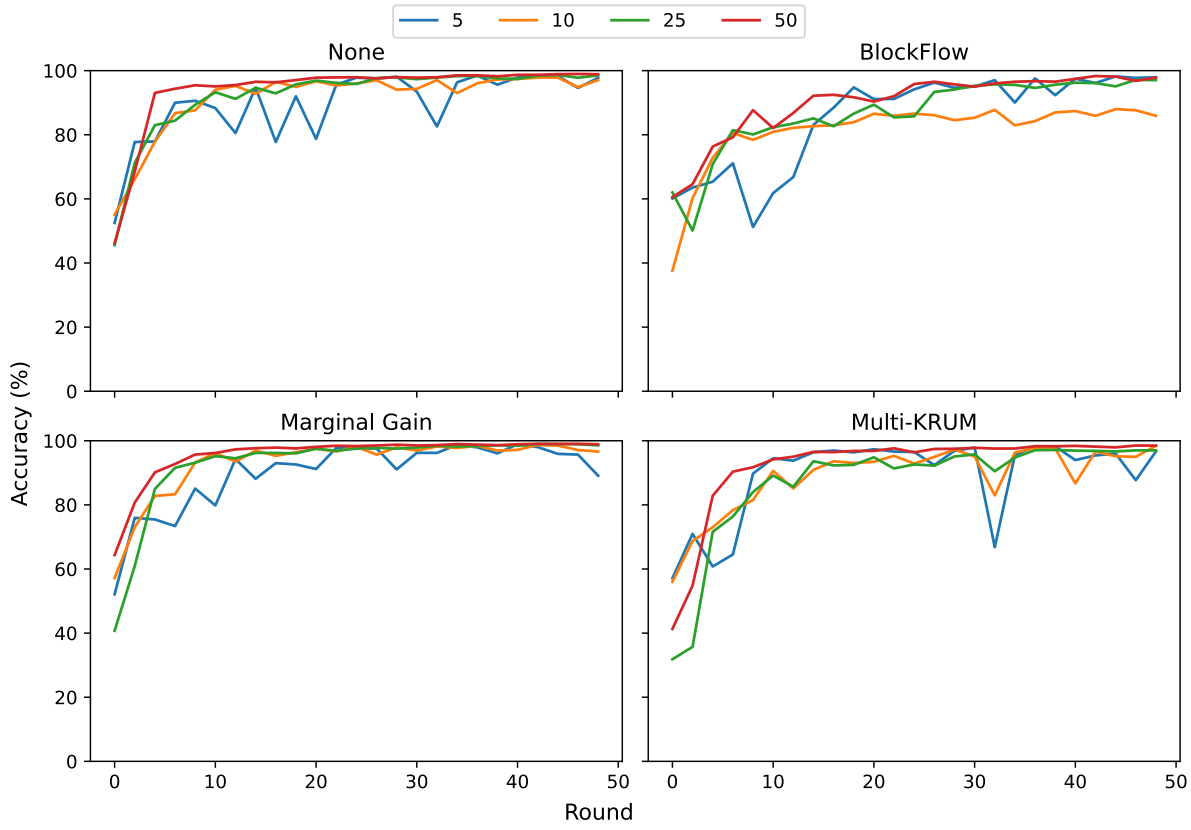


Figure 7.11: Model Accuracy Per Number of Clients

When it comes to the algorithm that performs best, the conclusions drawn from Section 7.2.1 remain true: the Marginal Gain performs the best, followed by the Multi-KRUM and finally by the BlockFlow.

7.2.2.3 Communication Costs

As it can be seen from Figure 7.12, the communication costs vary with the number of clients. Firstly, we observe that in terms of incoming traffic at the client process, there are significant differences depending on the scoring algorithm used.

One may notice that incoming traffic at the clients only increases when using scoring algorithms that are executed by the clients, such as the BlockFlow and the Marginal Gain. This can be explained by the fact that the more clients means the more weights from these clients need to be downloaded and scored. In contrary, when the scoring algorithm is executed by the server, there are virtually no changes in the incoming traffic when number of clients is increased. This is because the client only has to download the global weights after each round, which is not influenced by the number of clients.

Secondly, we observe that the incoming traffic always grows linearly at the servers when

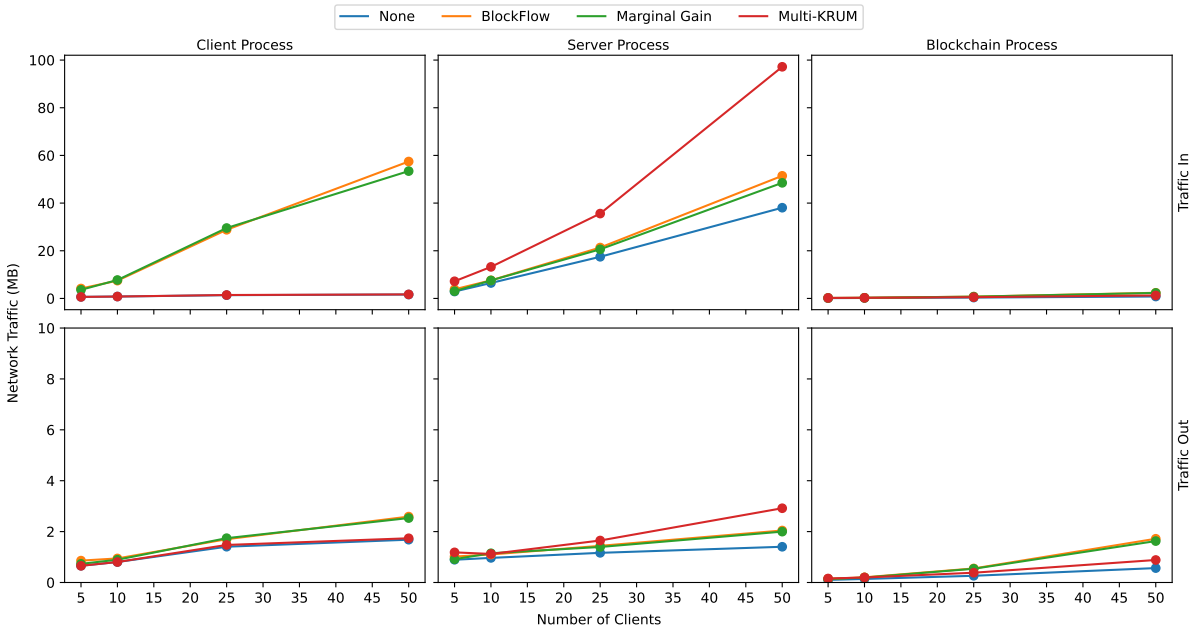


Figure 7.12: Network Traffic Per Number of Clients

using a scoring algorithm that runs at the clients. However, when using a algorithm that runs on the server, such as the Multi-KRUM, the growth is higher than for the remaining algorithms. In all cases, the servers always have to download the weights for aggregation. However, if the scoring is executed at the servers, they will download the weights twice as explained in Section 7.2.1, leading to a super-linear traffic growth. As previously suggested, this can be improved by caching the weights between these two phases.

Thirdly, we observe that the outgoing traffic differences are not significant for any of the processes. Regarding the clients and servers, we observe a very small increase on the client or server processes, if the scoring algorithm is executed by the clients or servers, respectively. However, the differences of outgoing traffic per number of clients for each scoring algorithm are negligible.

Finally, the differences observed in the blockchain process, both for incoming and outgoing traffic are negligible. With the increase of the number of clients, there are more transactions going through the blockchain. However, the transactions on the blockchain are very small when compared to the size of weights that need to be uploaded and downloaded by the clients and servers, respectively.

7.2.2.4 Computation Costs

Figure 7.16, Figure 7.17, Figure 7.18 show the RAM usage at the clients, servers, and blockchain processes, while Figure 7.19, Figure 7.20, Figure 7.21 show the CPU usage at the clients, servers, and blockchain processes, respectively. Overall, it can be seen that the number of clients has no major impact on RAM and CPU usage of either clients or servers.

Even though the usage of RAM and CPU at a certain points in time does not vary significantly, some scoring algorithms have more impact on the clients, or on the servers,

as discussed in Section 7.2.1. In addition, we observe that the increase of resource usage is proportional to the number of clients, depending on whether the the scoring algorithm is executed by the clients or the servers.

The only major change is observed in the RAM and CPU usage of the blockchain process. The higher the number of clients, the higher the RAM and CPU usage. This is expected as more clients are connected to the blockchain, meaning that more devices continuously interact and send transactions to the blockchain.

7.2.2.5 Conclusions and Improvements

In conclusion, scoring algorithms executed by the clients, i.e., the BlockFlow and Marginal Gain, show higher execution time increases with respect to the number of clients. In addition, the resource usage at the clients increases linearly with the number of devices. In contrary, algorithms executed by the servers, i.e., the Multi-KRUM, do not have significant effects on the clients' resources usage. Therefore, in systems with resource-constrained clients, algorithms executed by the server may be the ideal solution.

Finally, the resource usage of the blockchain process, mainly in terms of RAM, increases with the number of clients. The blockchain resource usage is an important aspect to consider in BFS systems. There is a clear trade-off between the number of clients with the blockchain resource usage.

7.2.3 Privacy Degrees

In this section, we analyze the impact of different privacy degrees on each scoring algorithm. In this set of experiments, all properties of the system are fixed, except for the degree of privacy, which varies between 0, 1 and 5, per each scoring algorithm.

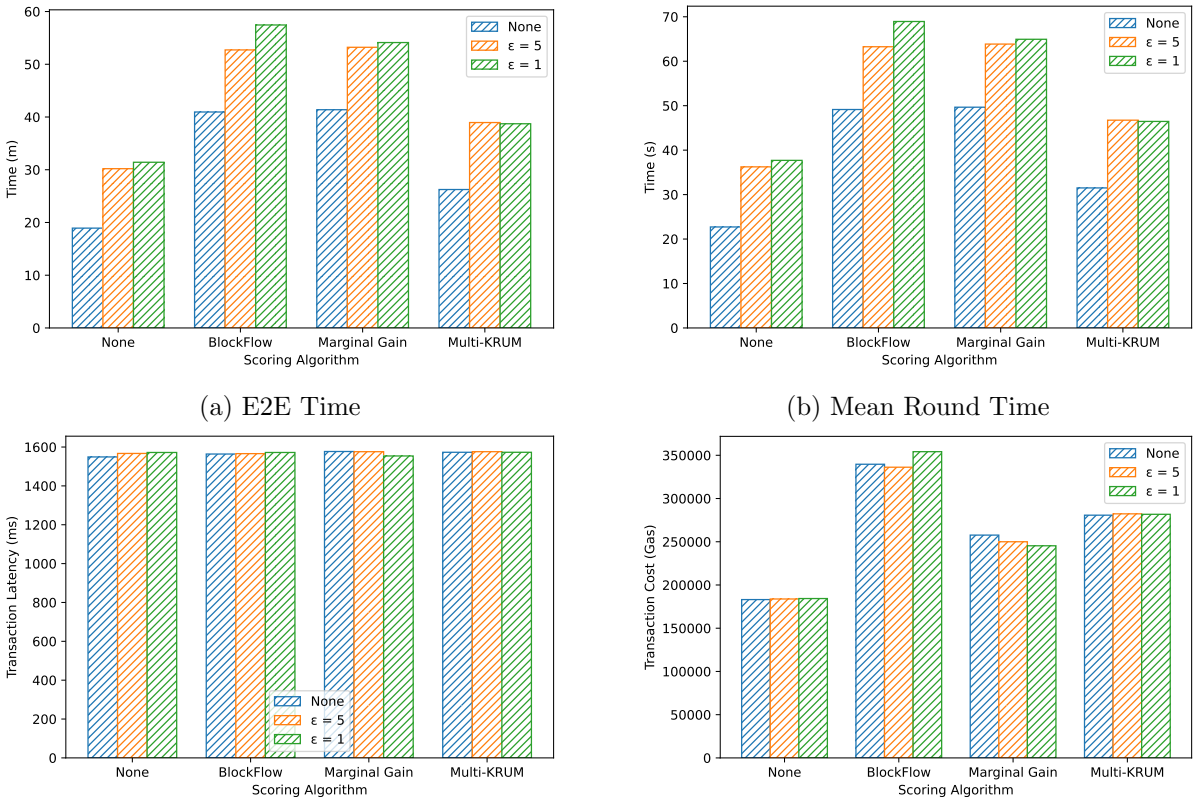
7.2.3.1 Execution Time, Transaction Cost, and Transaction Latency

As it can be seen from Figure 7.13, having a privacy mechanism increases the execution time of each round by approximately 16.6 seconds. In addition, the privacy degree itself does not seem to influence the execution time significantly.

Regarding the transaction latency and costs, there are no significant differences. The privacy mechanism is executed by the clients before they submit their model update, and does not change the number of transactions. Consequently, it has no impact on the blockchain process itself.

7.2.3.2 Model Accuracy and Convergence

As it can be seen from Figure 7.14, different scoring algorithms have different accuracy drops using different privacy degrees. Overall, we observe that not using a scoring algorithm performs the worse with the highest degree of privacy, i.e., $\epsilon = 1$. In addition, a lower degree of privacy, i.e., $\epsilon = 5$, yields similar, yet lower, model accuracy than not using privacy degree. The higher the degree of privacy, the higher the noise values that are added to the original weights. Since the weights include added noise, it is expected that the accuracy will be lower with the higher degrees of privacy.



(a) E2E Time

(b) Mean Round Time

(c) Transaction Latency

(d) Transaction Cost

Figure 7.13: Execution Time, Transaction Cost, and Transaction Latency Per Privacy Degree

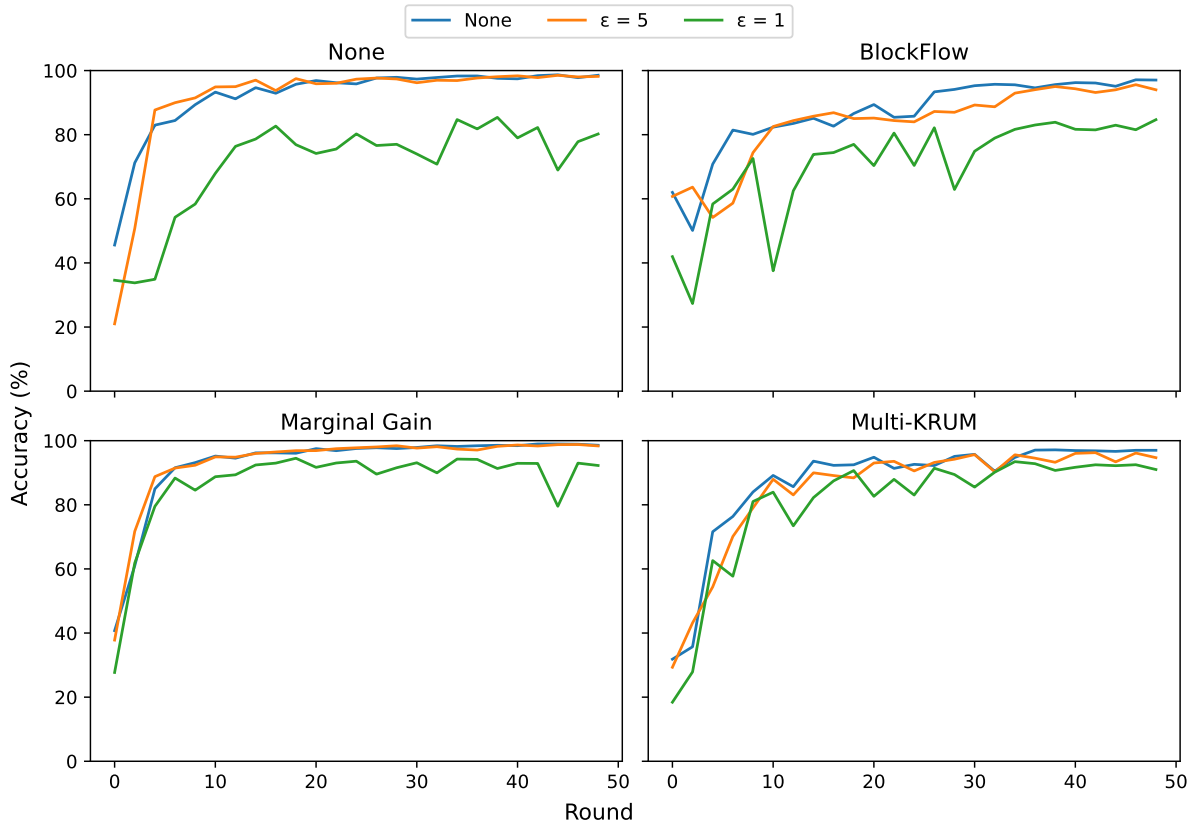


Figure 7.14: Model Accuracy Per Privacy Degree

Out of the three scoring algorithms, the Marginal Gain and Multi-KRUM perform the best in presence of the higher degrees of privacy. As explained in Section 2.3.2, both of these algorithms reject the worst updates, while BlockFlow does not. By rejecting the worst update, these algorithms always keep the best updates that provide the higher accuracy, even after noise is being added to the weights. Therefore, the Multi-KRUM and Marginal Gain have a lower accuracy drop with the higher privacy degrees, allowing them to maintain high model accuracy while preserving privacy.

7.2.3.3 Communication Costs

In terms of communication costs, as illustrated in Figure 7.15, there are no significant differences depending on the privacy degree used. Adding noise to the weights does not necessarily increase their sizes and, for that reason, the network traffic costs are not expected to change significantly.

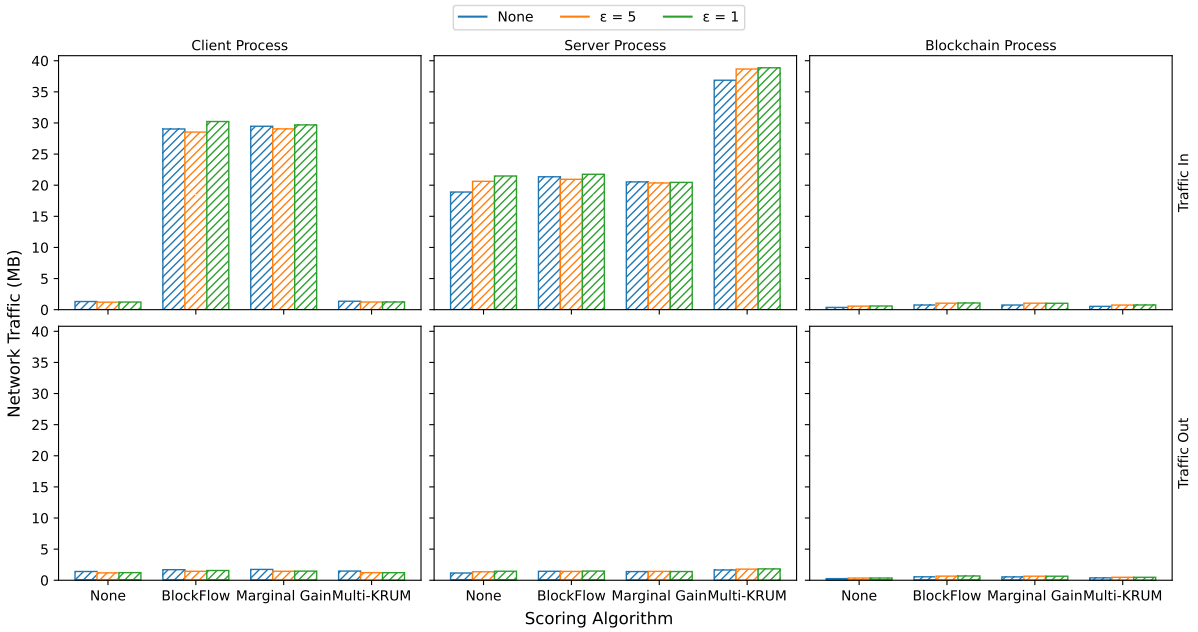


Figure 7.15: Network Traffic Per Privacy Degree

7.2.3.4 Computation Costs

Figure 7.22, Figure 7.23, Figure 7.24 show the RAM usage at the clients, servers, and blockchain processes, while Figure 7.25, Figure 7.26, Figure 7.27 show the CPU usage at the clients, servers, and blockchain processes, respectively. Since the privacy algorithm is only executed by the client process, we only expect the computation costs, namely the CPU usage, to increase at the client process. Overall, we can observe that this is true as there are no significant changes in the RAM or CPU usage for the servers and blockchain processes.

As it can be seen from Figure 7.22, the privacy mechanisms have no significant impact on RAM usage. The privacy algorithm has little RAM usage when compared to the total required for model training. In contrary, the CPU usage is higher, not necessarily in terms of usage percentage, but in terms of longer usage times. This can be seen from Figure 7.25

and is explained by the time required for the execution of the privacy algorithm at the clients.

7.2.3.5 Conclusions

In conclusion, we see that using a privacy algorithm increases the computation costs, and consequently, the resource usage, at the clients. This leads to longer execution times as clients that need to process their weights and add noise to them. Therefore, there is a clear trade-off between execution time and privacy. However, increasing the privacy degree does not increase the execution time.

In addition, one may notice that increasing the privacy degree leads to the lower model accuracy. This is expected since the privacy algorithm used introduce noise to the weights. However, some of the scoring algorithms are still able to achieve higher model accuracy values even with higher privacy degrees. This is the case for the Marginal Gain and Multi-KRUM, which are the most well-performing algorithms.

Finally, we can argue that adding a privacy preserving algorithm to a BFS system is crucial, specially if the model is trained with sensitive data. One of the main arguments to apply blockchain to a Federated Learning system is the traceability and auditability. To do so, the weights, or their representation in our case, are recorded in the blockchain, which means they are visible and retrievable by anyone in the network. This implies that there is a trade-off between traceability and auditability and the requirement for privacy mechanisms, which in turn leads to higher resource usage.

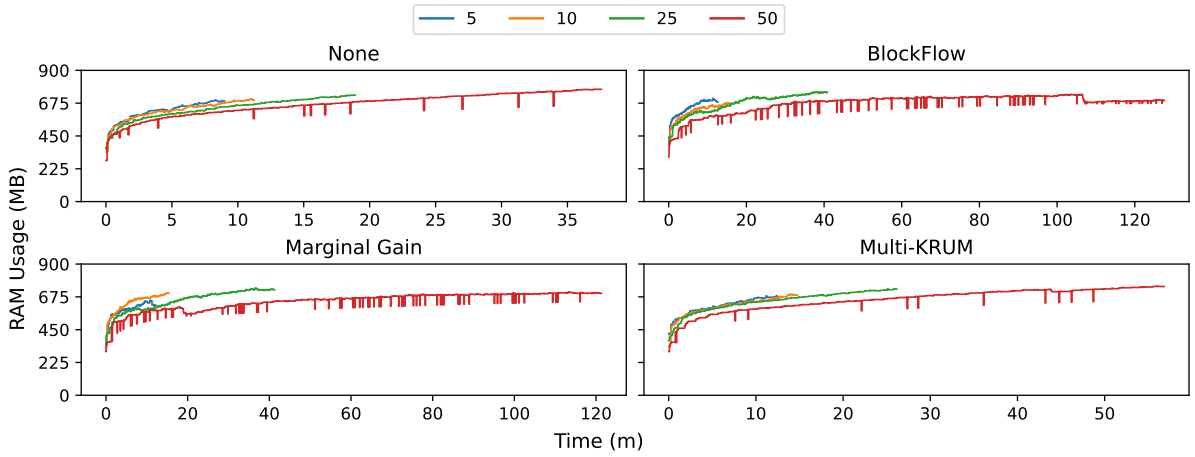


Figure 7.16: Client Process RAM Usage Per Number of Clients

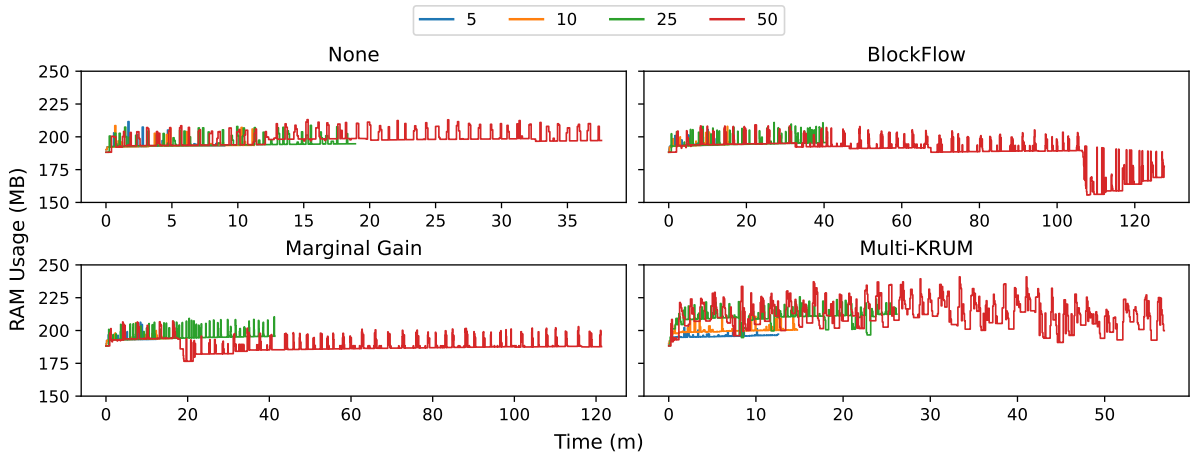


Figure 7.17: Server Process RAM Usage Per Number of Clients

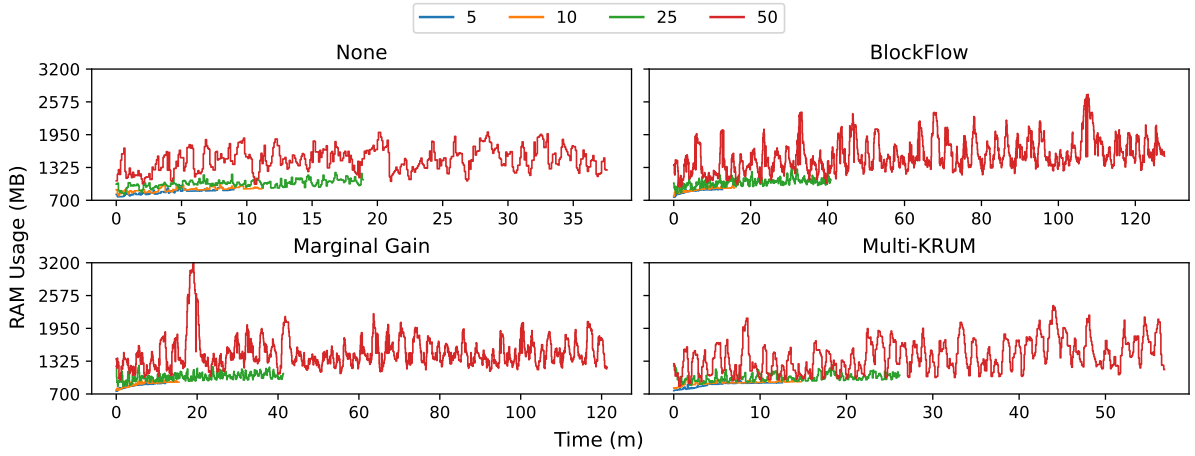


Figure 7.18: Blockchain Process RAM Usage Per Number of Clients

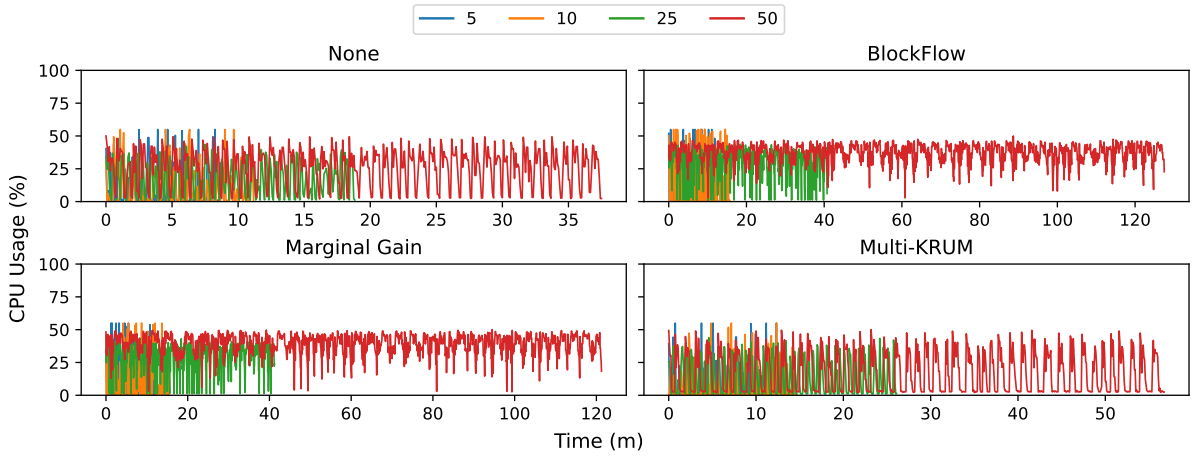


Figure 7.19: Client Process CPU Usage Per Number of Clients

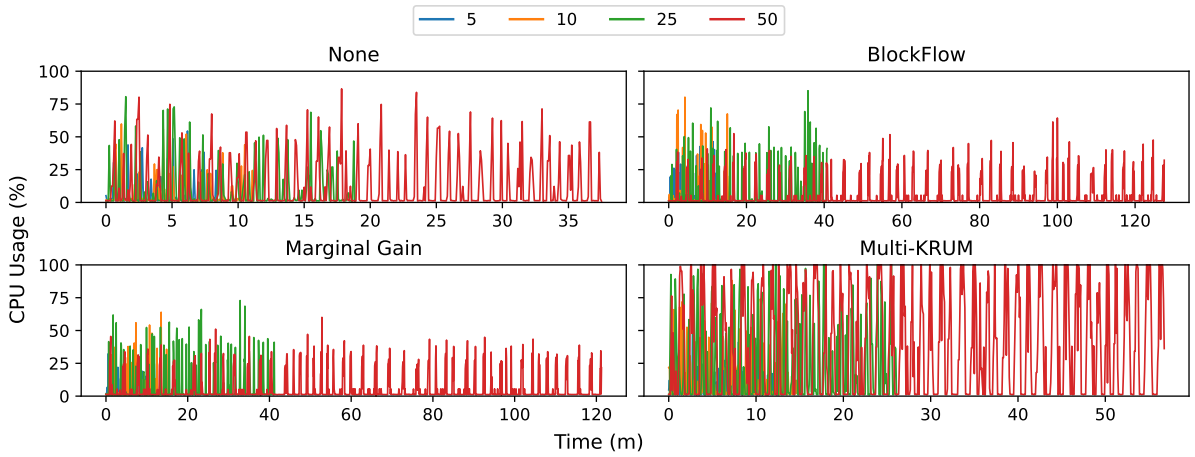


Figure 7.20: Server Process CPU Usage Per Number of Clients

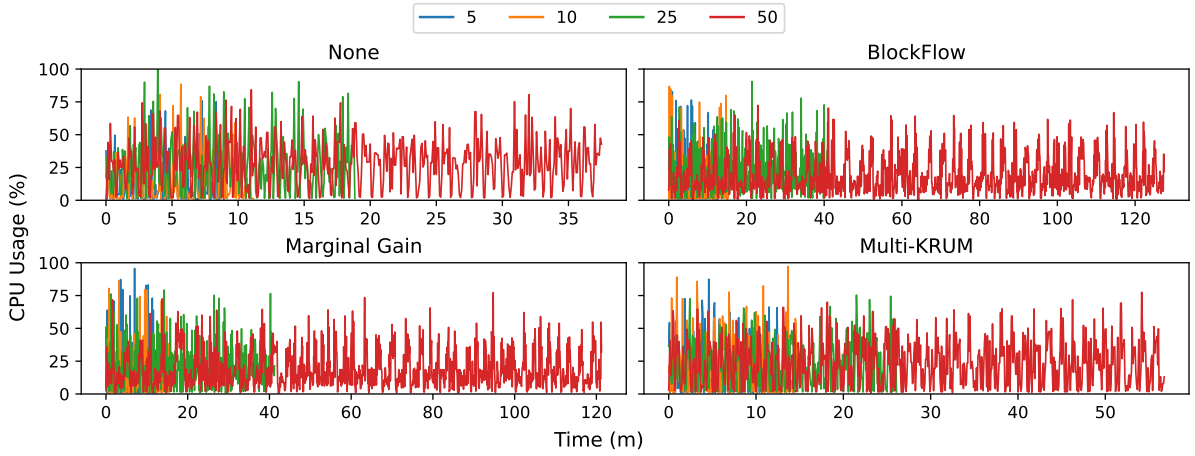


Figure 7.21: Blockchain Process CPU Usage Per Number of Clients

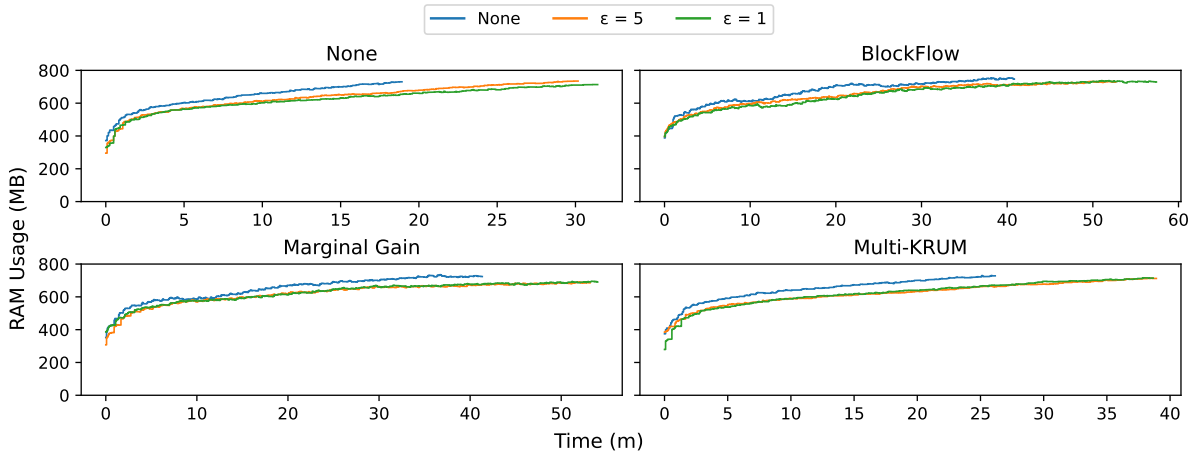


Figure 7.22: Client Process RAM Usage Per Privacy Degree

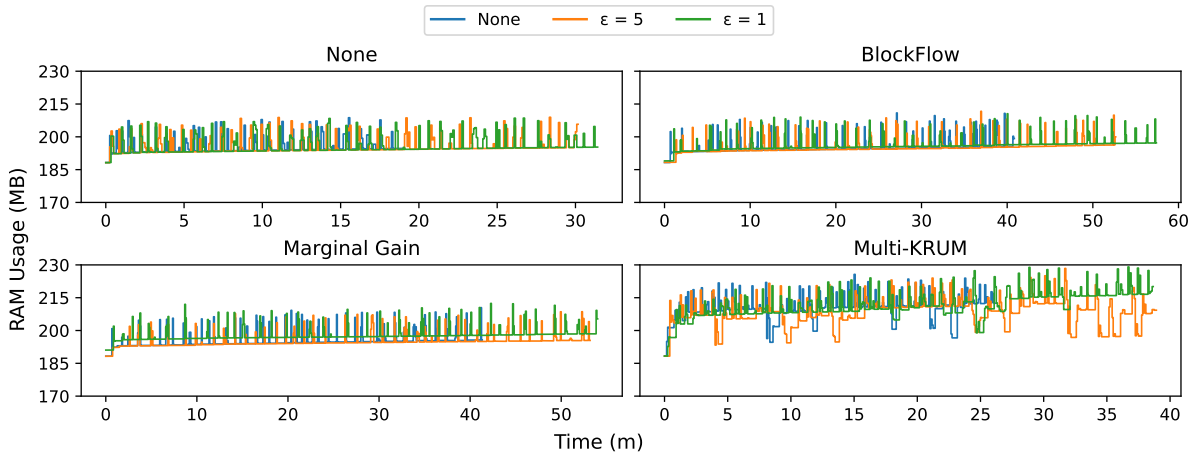


Figure 7.23: Server Process RAM Usage Per Privacy Degree

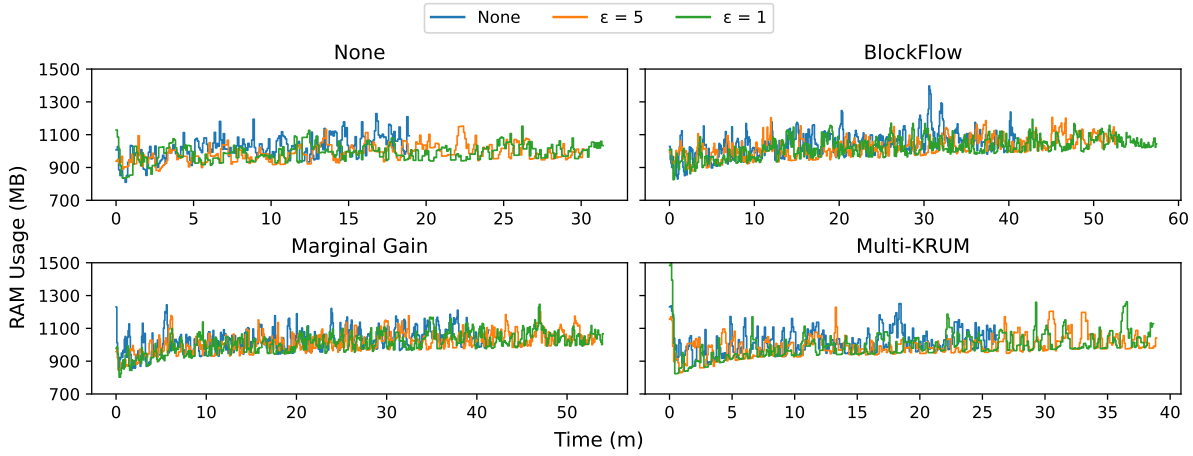


Figure 7.24: Blockchain Process RAM Usage Per Privacy Degree

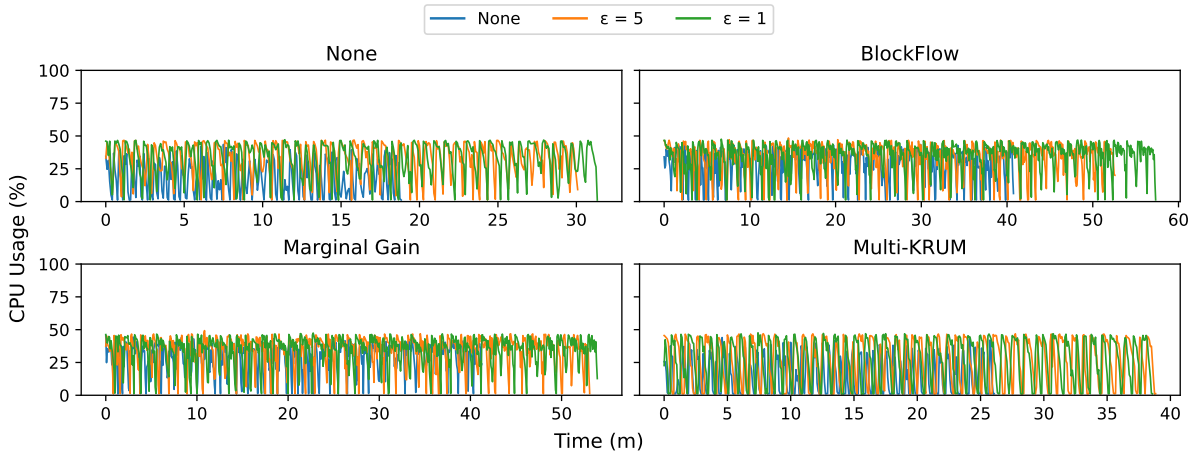


Figure 7.25: Client Process CPU Usage Per Privacy Degree

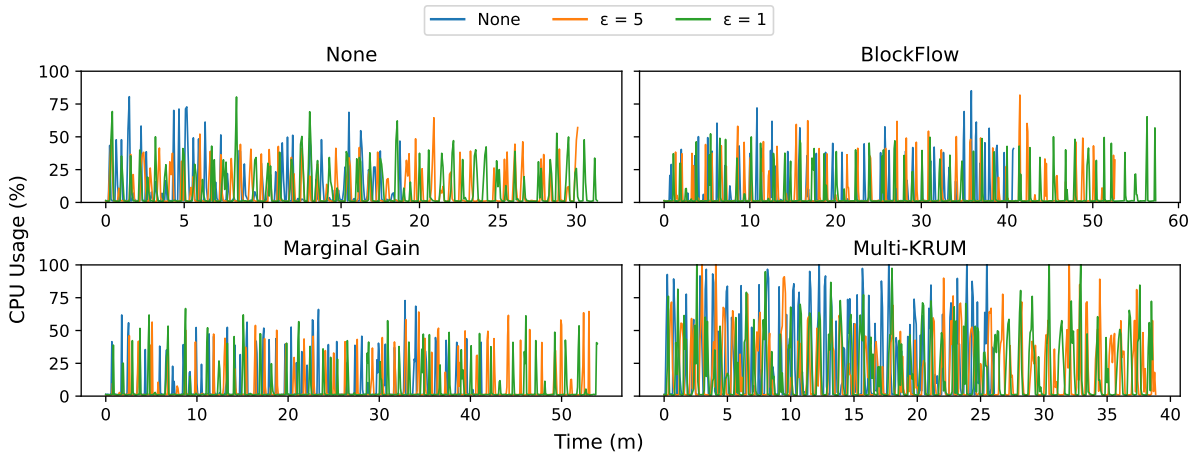


Figure 7.26: Server Process CPU Usage Per Privacy Degree

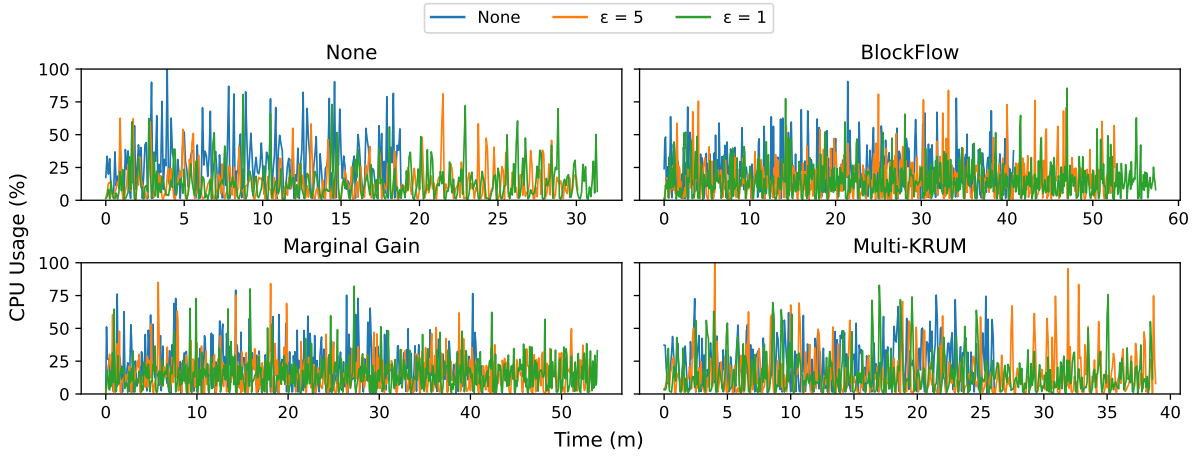


Figure 7.27: Blockchain Process CPU Usage Per Privacy Degree

Chapter 8

Proof of Concept of Vertical Blockchain-based Federated Learning

In this chapter, we provide a proof of concept of Blockchain-based Federated Learning (BFS) applied to a system with vertically partitioned data. Specifically, we aim to:

1. Present design and implementation of a Vertical Blockchain-based Federated Learning system. As discussed in Section 3.6, only one other work [56] discusses possibilities of integration of blockchain with Vertical Federated Learning with blockchain, but it does not provide an actual design or implementation.

2. Demonstrate that our framework is flexible to support additional execution flow steps and algorithms, as well as both Horizontal and Vertical Federated Learning. As explained in Section 5.3, the model we use for Vertical Federated Learning requires an additional step in each round. By showing that it is trivial to add new steps to the rounds, we also show that BlockLearning is flexible.

8.1 Requirements Analysis

As discussed in Section 5.3, our Vertical Federated Learning implementation uses the Split-CNN model. This model poses different requirements on our framework when compared to a regular CNN. These requirements are as follows:

1. *Support different models for the clients and the servers.* As explained in Section 5.3, the clients have the head model, while the servers have the tail model. This is different than the model we used for Horizontal Federated Learning, which is the same on the devices, regardless of their type.

2. *Support additional backpropagation confirmation phase.* After submitting the aggregations, and before terminating the round, the clients have to confirm that they backpropagated the gradient updates from the tail model to their own head models. This is related to the fact that the clients and the servers have different models.

The second requirement can be further divided into the following functional requirements:

1. Execution of backpropagation confirmation phase after aggregations submission phase.

2. Supporting backpropagation gradient submission by the servers to the smart contract.

3. Supporting backpropagation retrieval by the clients from the smart contract.

4. Supporting backpropagation confirmation by the clients to the smart contract.

The execution flow of each round when a Split-CNN model is used is illustrated in Figure 8.1. Compared to the original execution flow, depicted in Figure 4.1, it becomes clear that (i) the scoring phase is not used, as it is not relevant in the context of Vertical Federated Learning, and (ii) there is an additional backpropagation confirmation phase before the round is terminated.

Figure 8.1: Round Execution Flow With Split-CNN Model

8.2 BlockLearning's Extension

Most of the aforementioned requirements pertain the smart contract. Therefore, we first start by making the required changes to the smart contracts:

1. Add a new phase, called `WaitingForBackpropagation`, to the `RoundPhase` enumeration in the `Base` contract.

2. Create a new smart contract, called `VerticalSplitCNN`, which inherits the main functionality from the `Base` smart contract, and provides the additional functionalities required for the backpropagation phase.

Figure 8.2 illustrates the smart contract additions to the original design of our BlockLearning framework presented in Chapter 4. It is important to note that all these changes are additions, and they do not change how any existing feature behaves.

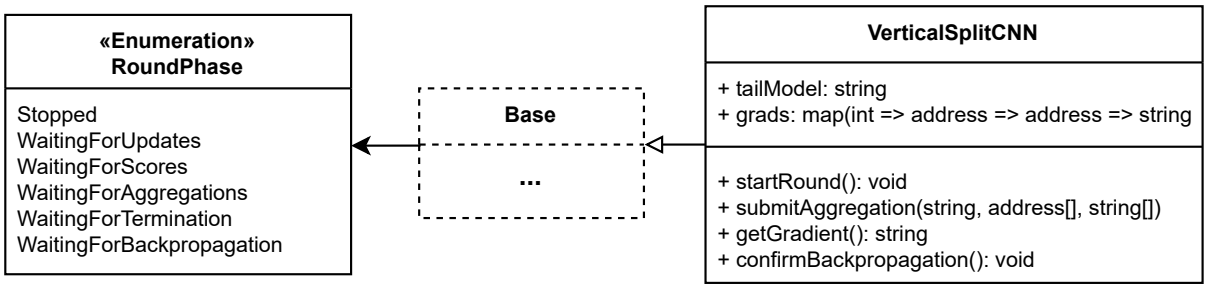


Figure 8.2: Split-CNN Smart Contracts Extension Class Diagram

After creating the new smart contract, the smart contract bridge has to be extended in

order to include the new smart contract functions. This extension is trivial as explained in Section 4.2.2.3.

The model training procedure for the Split-CNN model is slightly different from the CNN used for Horizontal Vertical Learning. For the Split-CNN, the clients submit an update with the intermediate outputs of the last layer of the head model, instead of the weights. During the aggregation phase, the servers calculate the gradients of the intermediate outputs, which are then backpropagated to the clients. To support this, we implement two new classes: `TrainerSplitCNN` and `AggregatorSplitCNN`, which implement the `trainer()` and `aggregate()` interfaces, respectively, as specified in Chapter 4. In addition, the `TrainerSplitCNN` class supports a new method, `backward()`, that will be called during the new backpropagation phase.

Finally, we write server and client scripts for the Vertical Federated Learning using the building blocks from the BlockLearning framework. Algorithm 2 illustrates the client main loop when used with a Split-CNN model. The main differences from the original BlockLearning framework, as presented in Chapter 4, are that the framework does not have scoring algorithm, and that we take into account the backpropagation phase. The server main loop when used with a Split-CNN model is the same as the original server main loop except that it initializes an instance of the `AggregatorSplitCNN` class instead of the regular aggregator.

Algorithm 2 Client Script Main Loop for Split-CNN

```
T ← Initialize Split-CNN Trainer
while True do
    P ← Get Phase From Smart Contract
    if P is Waiting For Updates then
        Execute Training Procedure T.train()
    else if P is Waiting For Backpropagation then
        Execute Backpropagation Procedure T.backward()
    end if
end while
```

8.3 Experiments and Results

After extending our framework in order to support the Split-CNN model, we executed the experiments for Vertical Federated Learning in order to validate whether our implementation was successful and the Vertical BFL can be supported. The experiments were executed in the same way as the experiments for the Horizontal BFL, using BlockLearning's Testbed. The only difference is that, this time, we used the client and server scripts that we developed for the Split-CNN model.

We ran two experiments with two different number of clients: 2 and 4. The decision to use 2 clients was motivated by [31], where the Split-CNN was introduced for Vertical FL without blockchain for the first time. In addition, we also performed experiments with 4 clients.

8.3.1 Execution Time, Transaction Cost, and Transaction Latency

As it can be seen from Table 8.1, this experiment was faster than most experiments, which is easily explained by the low number of clients. In addition, it is observed that the experiments take longer with 4 clients than with 2.

Regarding the transaction latency, it can be seen that it is similar to what we have seen in previous chapters. Similarly, the transaction costs do not present significant changes as the number of clients is relatively low. However, as expected, the transaction costs are slightly higher in case of having 4 clients.

	2	4
E2E Time (m)	18.08	24.30
Mean Round Time (s)	21.68	29.15
Mean Transaction Latency (s)	1.482	1.418
Mean Transaction Cost (Gas)	138659	141013

Table 8.1: Execution Time, Transaction Cost, and Transaction Latency Per Number of Clients

8.3.2 Model Accuracy and Convergence

Figure 8.3 illustrates the model accuracy of our experiments as well as those of Romanini et al. [72], where a Split-CNN model without blockchain was used with the MNIST data set. It can be seen that model accuracy of [72] is higher, which may be related to implementation differences such as the Machine Learning library used, which is not known.

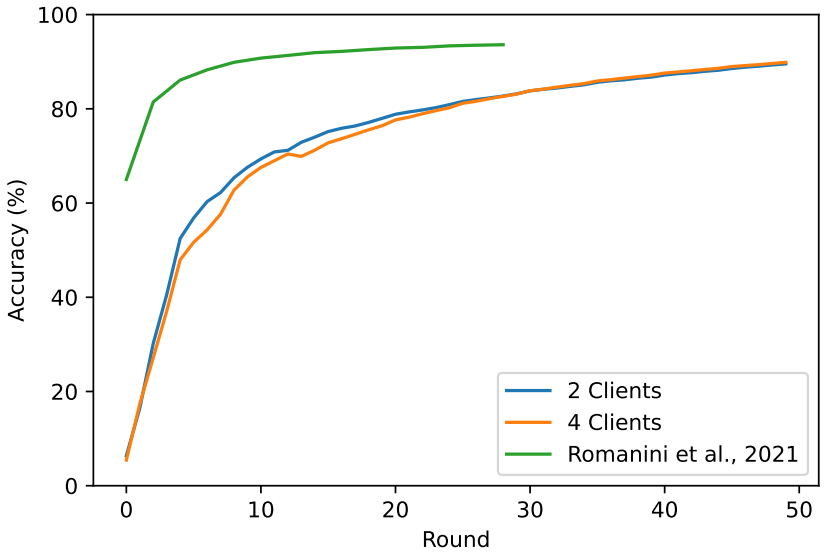


Figure 8.3: Model Accuracy Per Number of Clients

8.3.3 Communication Costs

The communication costs, illustrated in Figure 8.4, are also within the expected values.

We can observe at the client sides that there is no major difference of traffic when the number of clients increases. This can be explained by the fact that, by using a Split-CNN, each client is only required to upload its own intermediate results and downloads the gradient updates, which are similar in size.

At the servers, the costs are higher as the number of clients increases. Since the higher number of clients lead to higher number of heads in the Split-CNN model, the servers are required to download more intermediate results and to upload more gradient updates. Therefore, the network traffic at the servers increases with the number of clients.

On the blockchain, the difference of number of clients is not significant to make a significant difference on traffic, since these experiments ran with a very low number of clients.

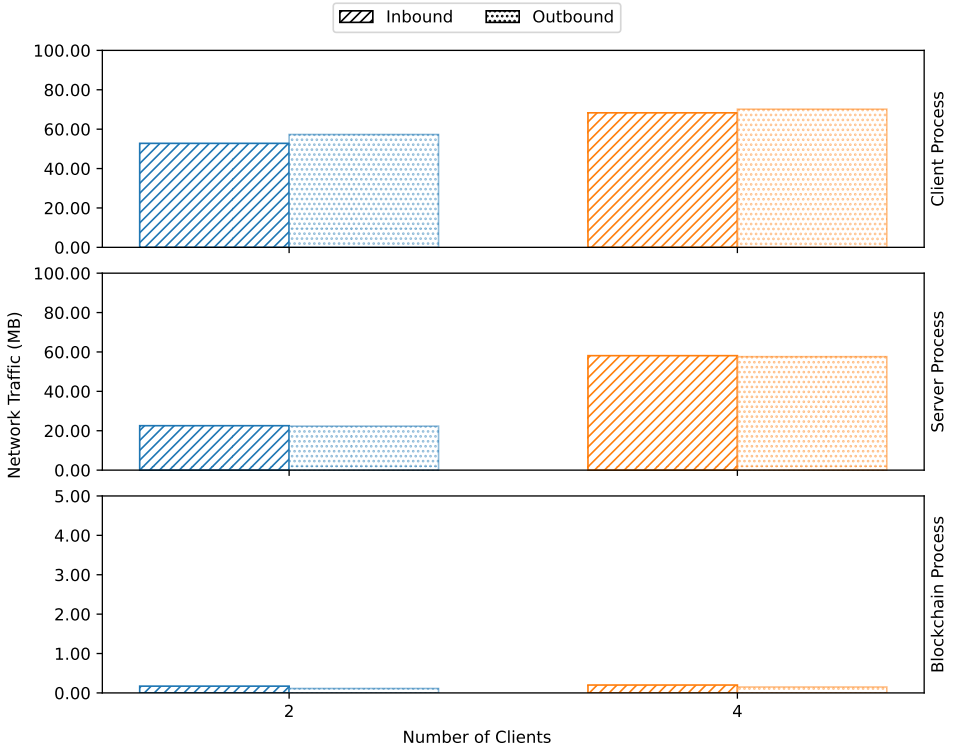


Figure 8.4: Network Traffic Per Round Per Number of Clients

8.3.4 Computation Costs

Computation costs, namely RAM usage and CPU usage, are depicted in Figure 8.5 and Figure 8.6, respectively.

Regarding the RAM usage, we observe that with a higher number of clients, there is a higher RAM usage on the serves and the blockchain processes. This is caused by the fact that more data is being stored in-memory due to the higher amount of intermediate results that the servers store in-memory, as well as the number of blockchain transactions in the blockchain. At the clients, however, the opposite happens. This can be explained by the fact that when there are more clients, each client has less features as per the data partitioning explained in Section 5.2.

Regarding the CPU usage, we see similar results as to the RAM usage, which are explained

by the same reasons.

8.4 Conclusions and Improvements

From our experiments, we can conclude that it is possible to apply a Blockchain-based Federated Learning system to vertically partitioned data. In addition, we also showed how flexible our BlockLearning framework is and how we can add new features to it without changing the rest of the framework.

In future, it would be interesting to investigate how to make BlockLearning more generic in order to support other Vertical Federated Learning models than only Split-CNN. In addition, it would be interesting to incorporate the Private Set Intersection phase into our framework. This would allow the framework to be directly applied to cases where the clients have intersecting, but not equal, sample spaces.

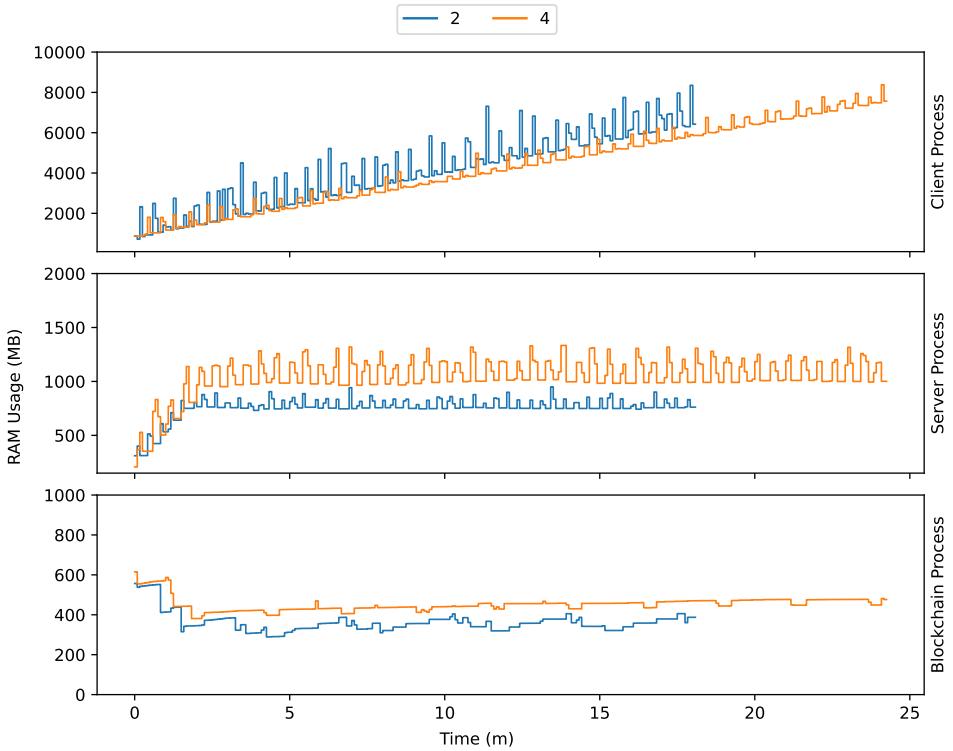


Figure 8.5: RAM Usage Per Number of Clients

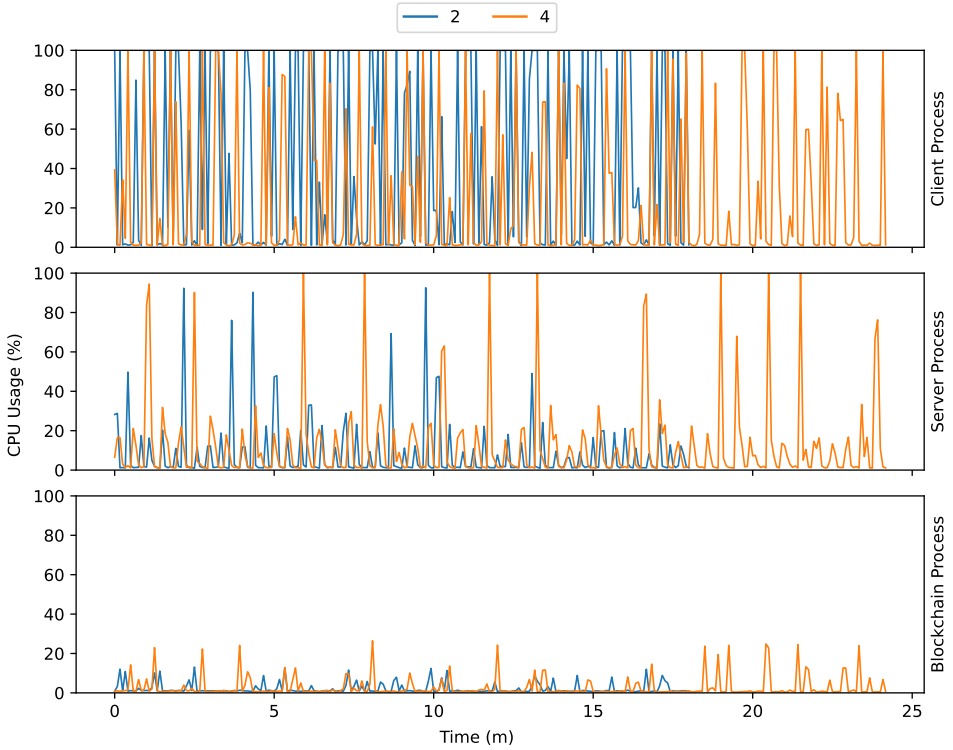


Figure 8.6: CPU Usage Per Number of Clients

Chapter 9

Conclusions and Future Directions

Blockchain-based Federated Learning (BFL) was initially introduced to bring some desirable properties of the blockchain, such as immutability, persistency, authentication, and decentralization, to the Federated Learning. In this thesis, we explored how different types of algorithms, namely consensus, participant selection, and scoring algorithms, impact the execution time, transaction costs, transaction latency, model accuracy and convergence, communication costs, and computation costs of the BFL system.

Our literature review revealed that there is a lack of publicly accessible and modular BFL framework. To fill this gap, we designed and implemented the first open-source modular BFL framework, which allows others to customize the system, in terms of architecture, algorithms and execution flow. By making it available to the public, it has the potential to empower future research on new BFL-related algorithms and architectures, without requiring the researchers to write the whole system from scratch.

9.1 Looking Back at the Main Research Question

We aimed to answer the question on *what is the impact of different consensus, participant selection and scoring algorithms in a BFL system on execution time, convergence and accuracy, as well as communication and computation costs.* For doing so, we executed all the experiments summarized in Table 9.1. The main findings of our experiments regarding each type of algorithms are:

- *Consensus Algorithms*: On the one hand, PoW presented the highest computation costs and is the slowest. On the other hand, QBFT and PoA presented the lowest computation costs and are the fastest. Moreover, QBFT has a three times higher communication costs compared to PoW and PoA. Consequently, we concluded that PoA is the most cost-efficient consensus algorithm analyzed.

- *Participant Selection Algorithms*: Both random selection and first-come first-served presented similar computation and communication costs. Random selection revealed to be more fair, providing more stable accuracy convergence, as it gives each client an equal chance of participating in a round.

- *Scoring Algorithms*: Adding a scoring algorithm to the BFS system increased the

execution time up to twice as much, depending on whether the algorithm is executed by the servers of the clients. Out of the three scoring algorithms analyzed, Marginal Gain provided the highest accuracy with increasing number of clients and increasing privacy degree. At the same time, it also had the highest computation costs for the clients. Moreover, Multi-KRUM revealed to be a good alternative in order to minimize the computation costs for the clients, while retaining high accuracy. Finally, BlockFlow performed the worst in all of the mentioned aspects.

Overall, our experiments showed that adding a blockchain, namely the Ethereum, to a Federated Learning system, increases the execution time, communication costs, and computation usage in general. After all, by adding a blockchain to a FL system, we are replacing a single centralized server by multiple distributed servers, which have to coordinate between themselves in order to reach a consensus in terms of storage and execution, consuming more time and resources.

Finally, we provided a proof of concept on how to extend our BlockLearning framework in order to support Vertical Federated Learning. By doing so, we presented the first implementation of Vertical Federated Learning in the context of BFL, showing that it is possible to have a Blockchain-based Vertical Federated Learning system. In addition, we demonstrated that our framework is flexible and extensible, such that it can be used to develop new algorithms and architectures.

9.2 Future Work

Below is a summary of some future directions that would be interesting pursue in the context of Blockchain-based Federated Learning systems:

- *Consensus Algorithms*: to investigate if it is feasible to extend the Ethereum blockchain with the custom resource-efficient algorithms presented in Section 3.1.
- *Scoring Algorithms*: to investigate and develop new scoring algorithms that do not require model evaluation at the clients side in order to reduce the resource usage. An example of this type of algorithm is Multi-KRUM, which is executed by the servers. All analyzed scoring algorithms executed by clients involve model evaluation, which leads to longer execution times and higher resource usage.
- *Blockchain-based Vertical Federated Learning*: to extend the BlockLearning framework in order to support the Private Set Intersection phase, such that it becomes more flexible to develop new Vertical Federated Learning algorithms.
- *BlockLearning GUI*: to develop a graphical interface for BlockFlow in order to allow users to submit training requests through an easy to use interface, as well as visualize the training process and download the weights directly without the need for command line tools.

⋆: This Work, I: IID, N: Non-IID, H: Horizontal, V: Vertical, ?: Unknown

Group	ID	Work	Consensus Algorithms	Clients	Participants Selection	Scoring	Data		Privacy Degree	Accuracy
							Partition	Distribution		
Consensus Algorithms	1	⋆	PoA	25	Random	None	H	N	None	98.54
	2		PoW							98.35
	3		IBFT							98.90
Participant Selection Algorithms	1	⋆	PoA	25	Random	None	H	N	None	98.54
	4				FCFS					98.18
Scoring Algorithms	1	⋆	PoA	25	Random	None	H	N	None	98.54
	10					BlockFlow				97.04
	14					Marginal Gain				98.58
	18					Multi-KRUM				97.00
Number of Clients	5	⋆	PoA	5	Random	None	H	N	None	97.76
	6			10						97.06
	1			25						98.54
	7			50						98.88
	8	⋆	PoA	5	Random	BlockFlow	H	N	None	97.94
	9			10						85.92
	10			25						97.04
	11			50						97.84
		[55]	PoW	25	?		H	?	$\epsilon = ?$	≥ 85.00
				50						
				100						
	12	⋆	PoA	5	Random	Marginal Gain	H	N	None	89.12
	13			10						96.62
	14			25						98.58
	15			50						98.90
		[8]	?	?	?		H	?	None	≥ 90.00

Table 9.1: Experiment Configurations and Accuracy

⋆: This Work, I: IID, N: Non-IID, H: Horizontal, V: Vertical, ?: Unknown

Group	ID	Work	Consensus Algorithms	Clients	Participants Selection	Scoring	Data		Privacy Degree	Accuracy
							Partition	Distribution		
Number of Clients	16	⋆	PoA	5	Random	Multi-KRUM	H	N	None	96.68
	17			10						98.44
	18			25						97.00
	19			50						98.48
		[91]	PoS, pBFT		?	?	H	I	$\epsilon = 10$	98.00
Privacy Degrees	1	⋆	PoA	25	Random	None	H	N	None	98.54
	19								$\epsilon = 5$	98.18
	20								$\epsilon = 1$	80.22
	10	⋆	PoA	25	Random	BlockFlow	H	N	None	97.04
	21								$\epsilon = 5$	94.00
	22								$\epsilon = 1$	84.68
		[55]	PoW	25	?		H	?	$\epsilon = ?$	≥ 85.00
	14	⋆	PoA	25	Random	Marginal Gain	H	N	None	98.58
	23								$\epsilon = 5$	98.36
	24								$\epsilon = 1$	92.26
		[8]	?	?	?		H	?	None	≥ 90.00
	18	⋆	PoA	25	Random	Multi-KRUM	H	N	None	97.00
	25								$\epsilon = 5$	94.70
	26								$\epsilon = 1$	91.00
		[64]	?	?	Random		H	N	$\epsilon = ?$	94.39
		[91]	PoS, pBFT	?	?		H	I	$\epsilon = 10$	98.00
									$\epsilon = 5$	96.50
									$\epsilon = 1$	86.00
Vertical Federated Learning	27	⋆	PoA	2	N/A	N/A	V	N	None	85.38
	28			4						88.31

Table 9.1: Experiment Configurations and Accuracy (Continued)

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Academy, B. Proof of authority explained, Dec 2020.

[3] Alqahtani, S., and Demirbas, M. Bottlenecks in blockchain consensus protocols. In *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)* (aug 2021), IEEE.

[4] Awan, S., Li, F., Luo, B., and Liu, M. Poster: A reliable and accountable privacy-preserving federated learning framework using the blockchain. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2019), CCS '19, Association for Computing Machinery, p. 2561–2563.

[5] Bao, X., Su, C., Xiong, Y., Huang, W., and Hu, Y. Flchain: A blockchain for auditable federated learning with trust and incentive. In *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)* (2019), pp. 151–159.

[6] Benet, J. Ipfs - content addressed, versioned, p2p file system, 2014.

[7] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, B., Van Overveldt, T., Petrou, D., Ramage, D., and Roselander, J. Towards federated learning at scale: System design. In *Proceedings of Machine Learning and Systems* (2019), A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, pp. 374–388.

[8] Cai, H., Rueckert, D., and Passerat-Palmbach, J. 2cp: Decentralized protocols to transparently evaluate contributivity in blockchain federated learning environments, 2020.

[9] Cao, M., Zhang, L., and Cao, B. Toward on-device federated learning: A direct

acyclic graph-based blockchain approach. *IEEE Transactions on Neural Networks and Learning Systems* (2021), 1–15.

[10] Castro, M., and Loskov, B. Practical byzantine fault tolerance, 1999.

[11] Chai, H., Leng, S., Chen, Y., and Zhang, K. A hierarchical blockchain-enabled federated learning algorithm for knowledge sharing in internet of vehicles. *IEEE Transactions on Intelligent Transportation Systems 22*, 7 (2021), 3975–3986.

[12] Chen, H., Asif, S. A., Park, J., Shen, C.-C., and Bennis, M. Robust blockchained federated learning with model validation and proof-of-stake inspired consensus, 2021.

[13] ConsenSys. Consensys/quorum: A permissioned implementation of ethereum supporting data privacy.

[14] Contributors, S. Solidity 0.8.15 documentation, 2021.

[15] Contributors, T. Convolutional neural network (cnn) : Tensorflow core.

[16] Cui, L., Su, X., Ming, Z., Chen, Z., Yang, S., Zhou, Y., and Xiao, W. Creat: Blockchain-assisted compression algorithm of federated learning for content caching in edge computing. *IEEE Internet of Things Journal* (2020), 1–1.

[17] Desai, H. B., Ozdayi, M. S., and Kantarcioglu, M. Blockfla: Accountable federated learning via hybrid blockchain architecture. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2021), CODASPY '21, Association for Computing Machinery, p. 101–112.

[18] D'Hondt, T. Federated learning over local learning: an opportunity for collaboration. Master's thesis, Eindhoven University of Technology, 2020.

[19] Dias, H. How to use pos in a local network, Apr 2022.

[20] Edwood, F. Proof-of-work vs. proof-of-stake for scaling blockchains, Jun 2020.

[21] Ethereum. ethereum/go-ethereum: Official go implementation of the ethereum protocol.

[22] Ethereum. Ethereum/web3.py: A python interface for interacting with the ethereum blockchain and ecosystem.

[23] Fan, S., Zhang, H., Zeng, Y., and Cai, W. Hybrid blockchain-based resource trading system for federated learning in edge computing. *IEEE Internet of Things Journal 8*, 4 (2021), 2252–2264.

[24] Fang, C., Guo, Y., Ma, J., Xie, H., and Wang, Y. A privacy-preserving and verifiable federated learning method based on blockchain. *Computer Communications 186* (2022), 1–11.

[25] Feng, L., Zhao, Y., Guo, S., Qiu, X., Li, W., and Yu, P. Bafl: A blockchain-based asynchronous federated learning framework. *IEEE Transactions on Computers 71*, 5 (2022), 1092–1103.

[26] Finney, H. Reusable proofs of work.

[27] GERON, A. *Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools and techniques to build Intelligent Systems.* O'Reilly, 2019.

[28] Holohan, N., Braghin, S., Mac Aonghusa, P., and Levacher, K. Diffprivlib: the IBM differential privacy library. *ArXiv e-prints 1907.02444 [cs.CR]* (July 2019).

[29] Hua, G., Zhu, L., Wu, J., Shen, C., Zhou, L., and Lin, Q. Blockchain-based federated learning for intelligent control in heavy haul railway. *IEEE Access 8* (2020), 176830–176839.

[30] Jain, A., Patel, H., Nagalapatti, L., Gupta, N., Mehta, S., Guttula, S., Mujumdar, S., Afzal, S., Sharma Mittal, R., and Munigala, V. Overview and importance of data quality for machine learning tasks. In *26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (New York, NY, USA, 2020), KDD '20, Association for Computing Machinery, p. 3561–3562.

[31] Jin, T., and Hong, S. Split-cnn: Splitting window-based operations in convolutional neural networks for memory system optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2019), ASPLOS '19, Association for Computing Machinery, p. 835–847.

[32] Kang, J., Xiong, Z., Niyato, D., Xie, S., and Zhang, J. Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. *IEEE Internet of Things Journal 6*, 6 (2019), 10700–10714.

[33] Kang, J., Xiong, Z., Niyato, D., Yu, H., Liang, Y.-C., and Kim, D. I. Incentive design for efficient federated learning in mobile networks: A contract theory approach. In *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)* (2019), pp. 1–5.

[34] Kang, J., Xiong, Z., Niyato, D., Zou, Y., Zhang, Y., and Guizani, M. Reliable federated learning for mobile networks. *IEEE Wireless Communications 27*, 2 (2020), 72–80.

[35] Kim, H., Park, J., Bennis, M., and Kim, S.-L. Blockchained on-device federated learning. *IEEE Communications Letters 24*, 6 (2020), 1279–1283.

[36] Kim, Y. J., and Hong, C. S. Blockchain-based node-aware dynamic weighting methods for improving federated learning performance. In *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (2019), pp. 1–4.

[37] Korkmaz, C., Kocas, H. E., Uysal, A., Masry, A., Ozkasap, O., and Akgun, B. Chain fl: Decentralized federated machine learning via blockchain. In *2020 Second International Conference on Blockchain Computing and Applications (BCCA)* (2020), pp. 140–146.

[38] Lan, Y., Liu, Y., Li, B., and Miao, C. Proof of learning (pole): Empowering machine learning with consensus building on blockchains (demo). *Proceedings of the*

AAAI Conference on Artificial Intelligence 35, 18 (May 2021), 16063–16066.

[39] LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist 2* (2010).

[40] Li, D., Han, D., Weng, T.-H., Zheng, Z., Li, H., Liu, H., Castiglione, A., and Li, K.-C. Blockchain for federated learning toward secure distributed machine learning systems: a systemic survey. *Soft Computing* (Nov. 2021).

[41] Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine 37*, 3 (2020), 50–60.

[42] Li, Y., Chen, C., Liu, N., Huang, H., Zheng, Z., and Yan, Q. A blockchain-based decentralized federated learning framework with committee consensus. *IEEE Network 35*, 1 (2021), 234–241.

[43] Lin, T., Kong, L., Stich, S. U., and Jaggi, M. Ensemble distillation for robust model fusion in federated learning. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 2351–2363.

[44] Lu, Y., Huang, X., Dai, Y., Maharjan, S., and Zhang, Y. Blockchain and federated learning for privacy-preserved data sharing in industrial iot. *IEEE Transactions on Industrial Informatics 16*, 6 (2020), 4177–4186.

[45] Lu, Y., Huang, X., Zhang, K., Maharjan, S., and Zhang, Y. Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles. *IEEE Transactions on Vehicular Technology 69*, 4 (2020), 4298–4311.

[46] Lu, Y., Huang, X., Zhang, K., Maharjan, S., and Zhang, Y. Blockchain and federated learning for 5g beyond. *IEEE Network 35*, 1 (2021), 219–225.

[47] Lu, Y., Huang, X., Zhang, K., Maharjan, S., and Zhang, Y. Low-latency federated learning and blockchain for edge association in digital twin empowered 6g networks. *IEEE Transactions on Industrial Informatics 17*, 7 (2021), 5098–5107.

[48] Ma, C., Li, J., Shi, L., Ding, M., Wang, T., Han, Z., and Poor, H. V. When federated learning meets blockchain: A new distributed learning paradigm. *IEEE Computational Intelligence Magazine 17*, 3 (2022), 26–33.

[49] Majeed, U., and Hong, C. S. Flchain: Federated learning via mec-enabled blockchain network. In *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (2019), pp. 1–4.

[50] Martinez, I., Francis, S., and Hafid, A. S. Record and reward federated learning contributions with blockchain. In *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)* (2019), pp. 50–57.

[51] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and Arcas, B. A. y. Communication-efficient learning of deep networks from decentralized data.

In *Proc. of the 20th International Conference on Artificial Intelligence and Statistics* (2017), vol. 54, pp. 1273–1282.

[52] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal 2014*, 239 (2014), 2.

[53] Mondal, A., Virk, H., and Gupta, D. Beas: Blockchain enabled asynchronous & secure federated machine learning, 2022.

[54] Moniz, H. The istanbul bft consensus algorithm, 2020.

[55] Mugunthan, V., Rahman, R., and Kagal, L. Blockflow: An accountable and privacy-preserving solution for federated learning. *ArXiv* (2020).

[56] Nagar, A. Privacy-preserving blockchain based federated learning with differential data sharing, 2019.

[57] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, 2009.

[58] Nguyen, D. C., Ding, M., Pathirana, P. N., Seneviratne, A., Li, J., and Vincent Poor, H. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials 23*, 3 (2021), 1622–1658.

[59] Nguyen, D. C., Ding, M., Pham, Q.-V., Pathirana, P. N., Le, L. B., Seneviratne, A., Li, J., Niyato, D., and Poor, H. V. Federated learning meets blockchain in edge computing: Opportunities and challenges. *IEEE Internet of Things Journal 8*, 16 (2021), 12806–12825.

[60] of Cambridge, U. Cambridge bitcoin electricity consumption index.

[61] Passerat-Palmbach, J., Farnan, T., Miller, R., Gross, M. S., Flannery, H. L., and Gleim, B. A blockchain-orchestrated federated learning architecture for healthcare consortia, 2019.

[62] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[63] Peng, Z., Xu, J., Chu, X., Gao, S., Yao, Y., Gu, R., and Tang, Y. Vfchain: Enabling verifiable and auditable federated learning via blockchain systems. *IEEE Transactions on Network Science and Engineering 9*, 1 (2022), 173–186.

[64] Peyvandi, A., Majidi, B., Peyvandi, S., and Patra, J. C. Privacy-preserving federated learning for scalable and high data quality computational-intelligence-as-a-service in society 5.0. *Multimedia Tools and Applications* (Mar 2022).

[65] Pfitzner, B., Steckhan, N., and Arnrich, B. Federated learning in a medical context: A systematic literature review. *ACM Trans. Internet Technol. 21*, 2 (jun

2021).

[66] Pokhrel, S. R., and Choi, J. Federated learning with blockchain for autonomous vehicles: Analysis and design challenges. *IEEE Transactions on Communications 68*, 8 (2020), 4734–4746.

[67] Preuveneers, D., Rimmer, V., Tsingenopoulos, I., Spooren, J., Joosen, W., and Ilie-Zudor, E. Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences 8*, 12 (2018).

[68] Qu, X., Wang, S., Hu, Q., and Cheng, X. Proof of federated learning: A novel energy-recycling consensus algorithm. *IEEE Transactions on Parallel and Distributed Systems 32*, 8 (2021), 2074–2085.

[69] Qu, Y., Pokhrel, S. R., Garg, S., Gao, L., and Xiang, Y. A blockchained federated learning framework for cognitive computing in industry 4.0 networks. *IEEE Transactions on Industrial Informatics 17*, 4 (2021), 2964–2973.

[70] Qu, Y., Uddin, M. P., Gan, C., Xiang, Y., Gao, L., and Yearwood, J. Blockchain-Enabled Federated Learning: A Survey. *ACM Computing Surveys* (Mar. 2022), 3524104.

[71] Ramanan, P., and Nakayama, K. Baffle : Blockchain based aggregator free federated learning. In *2020 IEEE International Conference on Blockchain (Blockchain)* (2020), pp. 72–81.

[72] Romanini, D., Hall, A. J., Papadopoulos, P., Titcombe, T., Ismail, A., Cebere, T., Sandmann, R., Roehm, R., and Hoeh, M. A. Pyvertical: A vertical federated learning framework for multi-headed splitnn, 2021.

[73] Sarker, I. H. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science 2*, 3 (Mar 2021), 160.

[74] Shayan, M., Fung, C., Yoon, C. J. M., and Beschastnikh, I. Biscotti: A blockchain system for private and secure federated learning. *IEEE Transactions on Parallel and Distributed Systems 32*, 7 (2021), 1513–1525.

[75] Shen, M., Wang, H., Zhang, B., Zhu, L., Xu, K., Li, Q., and Du, X. Exploiting unintended property leakage in blockchain-assisted federated learning for intelligent edge computing. *IEEE Internet of Things Journal 8*, 4 (2021), 2265–2275.

[76] Sherman, A. T., Javani, F., Zhang, H., and Golaszewski, E. On the origins and variations of blockchain technologies. *IEEE Security & Privacy 17*, 1 (2019), 72–77.

[77] Szilágyi, P. Eip-225: Clique proof-of-authority consensus protocol, Mar 2017.

[78] Toyoda, K., and Zhang, A. N. Mechanism design for an incentive-aware blockchain-enabled federated learning platform. In *2019 IEEE International Conference on Big Data (Big Data)* (2019), pp. 395–403.

[79] TsingZ0. Tsingz0/pfl-non-iid: Personalized federated learning simulation platform

with non-iid and unbalanced dataset.

[80] University, H. Differential privacy.

[81] Van Rossum, G., and Drake, F. L. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[82] Wang, S., Yuan, Y., Wang, X., Li, J., Qin, R., and Wang, F.-Y. An overview of smart contract: Architecture, applications, and future trends. In *2018 IEEE Intelligent Vehicles Symposium (IV)* (2018), pp. 108–113.

[83] Wang, Z., and Hu, Q. Blockchain-based federated learning: A comprehensive survey, 2021.

[84] Wei, K., Li, J., Ma, C., Ding, M., Wei, S., Wu, F., Chen, G., and Ranbaduge, T. Vertical federated learning: Challenges, methodologies and experiments, 2022.

[85] Weng, J., Weng, J., Zhang, J., Li, M., Zhang, Y., and Luo, W. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing 18*, 5 (2021), 2438–2455.

[86] Wilhelmi, F., Giupponi, L., and Dini, P. Blockchain-enabled server-less federated learning, 2021.

[87] Wood, G. Ethereum: A secure decentralised generalised transaction ledger.

[88] Yang, Q., Liu, Y., Chen, T., and Tong, Y. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol. 10*, 2 (jan 2019).

[89] Zhang, Q., Palacharla, P., Sekiya, M., Suga, J., and Katagiri, T. Demo: A blockchain based protocol for federated learning. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)* (2020), pp. 1–2.

[90] Zhang, W., Lu, Q., Yu, Q., Li, Z., Liu, Y., Lo, S. K., Chen, S., Xu, X., and Zhu, L. Blockchain-based federated learning for device failure detection in industrial iot. *IEEE Internet of Things Journal 8*, 7 (2021), 5926–5937.

[91] Zhao, Y., Zhao, J., Jiang, L., Tan, R., Niyato, D., Li, Z., Lyu, L., and Liu, Y. Privacy-preserving blockchain-based federated learning for iot devices. *IEEE Internet of Things Journal 8*, 3 (2021), 1817–1829.

[92] Zhou, J., Zhang, S., Lu, Q., Dai, W., Chen, M., Liu, X., Pirttikangas, S., Shi, Y., Zhang, W., and Herrera-Viedma, E. A survey on federated learning and its applications for accelerating industrial internet of things, 2021.

[93] Zhou, S., Huang, H., Chen, W., Zhou, P., Zheng, Z., and Guo, S. Pirate: A blockchain-based secure framework of distributed machine learning in 5g networks. *IEEE Network 34*, 6 (2020), 84–91.