ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica – Scienza e Ingegneria

Corso di Laurea in Ingegneria Informatica

INTEGRATING FEDERATED LEARNING WITH THE BLOCKCHAIN INFRASTRUCTURE

Relatore:
Prof. Paolo Bellavista

Presentata da:
Bombarda Alessandro

Sessione 3
Anno Accademico 2023/2024

Introduction

In recent years, advancements in distributed computing and decentralized technologies have reshaped the landscape of machine learning and data processing. This document explores the intersection of two innovative paradigms: Federated Learning and Blockchain. Each of these fields has gathered significant attention for its unique potential — federated learning for enabling collaborative model training without centralized data aggregation, and blockchain for providing decentralized governance and immutable ledgers with robust security guarantees.

Federated learning is a decentralized approach to training machine learning models collaboratively across multiple devices or nodes while keeping data local to each participant. Originally developed to address privacy concerns, Federated Learning has found applications in domains such as healthcare, mobile devices, and IoT, where sensitive data remains confined to the edge devices that generate it. However, Federated Learning is not without its challenges.

Blockchain, on the other hand, is a decentralized ledger technology that ensures secure and immutable decentralized data storage and trustless interactions among participants. First popularized by cryptocurrencies such as Bitcoin, blockchain has since evolved to support a wide range of use cases, including supply chain management, smart contracts, and secure data sharing.

The possibility of integrating blockchain with federated learning has sparked interest in the research community as a means to address some of the inherent limitations of the paradigm. Federated learning's reliance on a central server for model aggregation introduces risks of a single point of failure and trust issues among participants. Blockchain's decentralized nature and consensus mechanisms can solve this problem among others, such as ensuring model integrity and fairness in participation. However this comes with its own set of

challenges, mostly communication, storage and (depending on the underlying mechanism of the blockchain, also computation) overhead.

Integrating blockchain into federated learning systems presents unique challenges. These include the computational and communication overhead introduced by blockchain protocols, ensuring scalability in systems with millions of devices, and aligning the asynchronous nature of federated learning with blockchain's consensus requirements. Despite these hurdles, the potential benefits—such as enhanced security, transparency, and trust—make this integration a promising avenue of research.

This thesis aims to explore the foundational concepts of both federated learning and blockchain, analyze their potential integration, and address the challenges that arise from combining these technologies. Through theoretical insights and practical implementation, the study will evaluate the feasibility and benefits of using blockchain to enhance federated learning systems.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Blockchain

1.1 Preliminaries

1.1.1 Context of the Thesis

Advancements in technology have driven a data-centric shift across industries such as healthcare, finance, smart cities, and the Internet of Things (IoT). This growing reliance on data has introduced significant challenges in privacy, security, and scalability, necessitating innovative solutions. This thesis explores the intersection of two emerging paradigms—Federated Learning (FL) and Blockchain technology—as a potential means to address these challenges.

Federated Learning has emerged as a transformative approach to machine learning by enabling collaborative model training without the need to share raw data. This is achieved by keeping relevant computation local to devices and only sharing updates with the interested party. This preserves data privacy while still achieving high-performance learning. Despite its benefits, FL faces critical challenges, including reliance on a central server, susceptibility to malicious participants, and a lack of transparency in the training process.

Blockchain technology, initially developed as the backbone for cryptocurrencies like Bitcoin, provides a decentralized, secure, and transparent framework for managing data and operations among untrusted parties. With its

core features—such as consensus mechanisms, immutability, and smart contracts—Blockchain has demonstrated value in sectors ranging from finance to supply chain management. However, its scalability limitations and resource-intensive operations can hinder its applicability in dynamic environments.
Integrating FL and Blockchain presents a promising avenue for overcoming the limitations of both technologies. Blockchain's decentralization can eliminate the dependency on a central server in FL, while its transparency and consensus protocols enhance trust and accountability. Conversely, FL's distributed learning capabilities reduce the need for raw data storage on the Blockchain, mitigating its scalability concerns.
This thesis is motivated by real-world challenges such as:

- Ensuring privacy and security in data-sharing across regulated sectors like healthcare and finance.
- Establishing trust in collaborative environments where participants may not fully trust each other, such as in decentralized IoT systems.

While Federated Learning and Blockchain are an ongoing field of research and a fair amount of articles have been published about these topics, their integration has been underexplored. Combining the two technologies introduces novel challenges including communication privacy and efficiency, synchronism of updates, security and scalability.
Through the design and evaluation of a Blockchain-enhanced Federated Learning framework, this thesis aims to provide a foundational understanding of the two technologies and explore the challanges and opportunities in context of privacy, trust, and scalability. The following section outlines the specific objectives of this thesis, which are derived from this contextual foundation.

1.1.2 Objectives of the Thesis

This document aims to explore the feasibility and practicality of implementing a federated learning system on a blockchain network. The contributions of this work are threefold:

Explore the technologies The thesis seeks to provide to the reader a foundational understanding of both Federated Learning and Blockchain technologies. We will try to be as thorough as possible, within the limits of the purposes of this document, highlighting the individual strengths, weaknesses, and potential areas of application of both technologies.

Exploring the integration After providing an understanding of the technologies and their strengths and weaknesses we want to discuss the advantages and challenges that arise when joining them.
We will discuss them in depth in the coming chapters but as an anticipation, some of the advantages include:

- Addressing the central server reliance in FL through Blockchain's decentralized infrastructure.
- Enhancing trust, transparency, and security in FL systems using Blockchain's consensus mechanisms and immutability.
- Provide incentives for devices to partake in the federated learning process.

And some of the challenges include:

- Computational and communication overhead due to Blockchain protocols.
- Ensuring security and consistence of the gloabl model.

When discussing the challenges and advantages we will also take a look at existing research and how some of these problems are dealt with and how some advantages are successfully exploited.

Proposing our own implementation After listing implementative and theoretical challenges we will proceed to propose our own implementation of a federated learning system on a blockchain network.

We will try to compare it with the research papers discussed over the course of the document and provide our own unique opinions and implementation choices ragarding topics with:

- Outlining the network, consensus, and Blockchain layers to integrate the two technologies effectively.
- Exploring implementation choices, such as public versus private Blockchains, consensus mechanisms, and the role of smart contracts.
- Dealing with malicious updates in a robust manner.
- Making the system communication efficient and address storage and availability challenges.

We will then proceed to evaluate our implementation and compare it to a classic client-server implementation and show the feasibility of blockchain integration.

Contributing to Academic research Saying that one of the aims of this document is to contribute to academic research may sound a bit pretentious as this is by no means a groundbreaking work. However, by gathering existing research and providing our own implementation and opinions on some relevant topics we hope that this document will be useful to someone looking for a starting point in this field.

Conclusion By achieving these objectives, the thesis endeavors to shed light on the transformative potential of combining Blockchain and Federated Learning, laying the groundwork for innovative solutions to contemporary challenges in privacy-preserving and secure collaborative learning.

1.1.3 Structure of the Document

The remainder of this document is organized as follows:

- In this chaper we will delve into the fundamentals of blockchain technology, examining its mechanisms, benefits, and limitations, focusing on the Bitcoin network as a case study.

- In chapter 2 we introduce federated learning. As a preliminary, we propose a very basic introduction to machine learning in 2.2, which introduces some topics and terminology relevant to federated learning. Then we proceed to explore the workings of the original federated learning proposal in 2.5.

- In chapter 3 we explore the possibility of integrating the two technologies. First we explore advantages, challenges and existing solutions in 3.1. Then we proceed to present our implementation and our unique views on how to solve some challenges in 3.3. We then evaluate our implementation against vanilla federated learning in 3.4.

- Finally we sum up the key findings and insights of this document in 3.4.4.

1.2 Introduction to Blockchain

Nowadays blockchain has become a buzzword in the tech industry and it gets thrown around in many context just to generate hype. Many people view it as something mysterious and cryptic. However the core idea behind the technology is very simple: a blockchain is just a linked list of blocks where each new block can only be appended at the end of the list. The data can represent anything, from financial transactions to medical records: as a matter of fact, the first use case of blockchain was timestamping documents. The definition we just gave is very simple but intuitive. However when we talk about blockchain we usually refer to the ecosystem and philosophy surround it as well. A more precise definition is the following:

> *A blockchain is a decentralized and distributed digital ledger that records transactions across multiple computers in a way that ensures the security, transparency, and immutability of the data.*

We use to call transactions the events that are recorded in the chain even though they may not necessarily be associated with finance. Each transaction on a blockchain is grouped into a **block** which is linked to previous blocks, forming the append-only linked list.
The core idea of a blockchain is that once a block is validated and added to the chain, it is very hard to alter or delete it, making the blockchain highly resistant to tampering and fraud. This is made possible via a consensus mechanism 3.3.2, which is a process that multiple nodes (computers) follow to agree on the validity of blocks before adding them to the chain.
Blockchain technology enables secure, peer-to-peer transactions without the need for a central authority, such as a bank or government, because each participant in the network holds a copy of the entire blockchain. This decentralized structure is what ensures data integrity, as any changes to a transaction would require the consensus of the majority of the network. The transparency of the blockchain means that all participants can see the transaction history, making it a trusted source for verifying information.

Originally developed as the technology underpinning cryptocurrencies like Bitcoin, blockchain has since expanded to numerous applications beyond finance, including supply chain management, voting systems, and healthcare. As we will see in 1.3.3, smart contracts — programmable agreements that automatically execute when predefined conditions are met — are another innovative use of blockchain, as they enable automated and enforceable contracts without intermediaries.

Blockchain technology is still a field of ongoing research and, while the core principles have to remain consistent with what we stated above, there are many diverse implementation paths that can be taken when building a network based on a blockchain. In order to provide a foundational understanding of state of the technology and its limitations, we will explore the features and characteristics of the biggest and first implemented public blockchain: Bitcoin. This analysis will also be beneficial to the understanding of the project implementation as we will see in 3.3.

1.3 Blockchain Classification

1.3.1 Public vs Private Blockchains

Public blockchains are open to anyone, providing a high level of decentralization and transparency. However, this also implies higher costs associated with blockchain usage and also increases the risk of incurring into malicious actors, as anyone can participate without prior verification.
In contrast, private blockchains restrict access to authorized participants, allowing more control and accountability. Private blockchains are often preferable for enterprise applications, where security and control are paramount, and accountability can be enforced among known participants. Being specialized for a dedicated task, the costs associated with blockchain usage are also lower.

1.3.2 Consensus Mechanisms

Consensus mechanisms are algorithms which coordinate the participants of the chain towards reaching agreement on the global state of the network. Different consensus mechanisms have varying impacts on performance, energy consumption, and overall network security.
Proof of Work (PoW) was the first consensus mechanisms to be adopted. In PoW, miners solve cryptographic puzzles to validate transactions, which requires significant computational power and energy. While PoW ensures security, it is energy-intensive and can cause delays due to mining time, as well as increase the risk of blockchain forks. These factors limit its scalability and efficiency for real-time applications.
Proof of Stake (PoS) addresses some of PoW's shortcomings by selecting validators based on the amount of cryptocurrency they are willing to stake. PoS reduces energy consumption and lowers forking risks since validators are financially incentivized to act honestly. Additionally, PoS networks tend to be more scalable and faster.

Consortium-based mechanisms, often used in permissioned blockchains, involve a select group of trusted entities validating transactions. These systems are more energy-efficient than PoW and offer faster transaction processing due to fewer validators. This mechanism is the most energy efficient and scalable out of the three however it comes with the downside of being more centralized.

1.3.3 Smart Contract-Based Platforms

While custom blockchains allow for greater control and optimization specific to a use case, smart contracts provide a way to leverage existing, widely adopted blockchain platforms like Ethereum. Smart contracts are self-executing contracts with the terms directly written into code. This makes them ideal for automating tasks such as managing transactions and coordinating activities in decentralized applications. They allow federated learning frameworks to be easily deployed across different blockchains, as long as those platforms support similar contract functionality. This compatibility enables easier integration with established blockchain networks and reduces the need for building infrastructure from scratch.

1.4 Bitcoin Blockchain

Bitcoin was the first cryptocurrency, introduced by an anonymous entity known as Satoshi Nakamoto in 2008. It is a decentralized digital currency that operates on a peer-to-peer network. All bitcoin transactions happen on the bitcoin blockchain and after being successfully verified by its members, they are immutably stored on a block. [**?**]
The entities who take part in transactions in the bitcoin network are called wallets. Nodes may have multiple wallets as well as not have any at all, if they're not interested in participating in transactions.
Wallets consist of a pair of keys $(K_{\text{priv}}, K_{\text{pub}})$ where:

- K_{priv} uniquely identifies the wallet and is necessary to access the funds of the wallet.
- K_{pub} is a public key which can be generated from the private one.

Private keys are hard to memorize and easy to mistype thus, instead of storing the raw private key, users generate and store it in the form of a *seed phrase* or *mnemonic* which is a sequence of words that map uniquely to a private key. The pool of words that can be used is defined in the BIP39 standard.

1.4.1 Transactions

A Bitcoin transaction (TX) is a collection of **inputs**, **outputs** and **metadata** (See 1.1). Each input in a transaction references the funds of an output of a previous transaction.
As one would expect, the sum of the inputs must not be lower than the sum of the outputs. This doesn't mean, however, that the two amounts must be equal. The sum of the inputs must be **greater or equal** than that of the outputs and is, in practice, always greater. The difference between the two becomes the transaction fee, rewarded to miners (see the *Mining* section).

An interesting feature of bitcoin transactions is that outputs can't be partially spent. For example, if a TX output is worth `10 BTC`, an input of a future transaction referencing it, will necessarily use the full `10 BTC`.

It comes naturally to ask how to partially spend funds then. The answer is simple: if a wallet only wants to send `2 BTC` but has a unspent output of `10 BTC`, then the transaction will have:

- An input of `10 BTC` referencing the unspent output.
- An output of `2 BTC` sent to the recipient.
- A "change" output of `8 BTC` sent to the wallet itself.

As it appears, wallets **don't** have a balance in the strict sense. Instead there is the concept of **Unspent Transaction Output (UTxO)**. The *balance* of a wallet be retrieved by inspecting all the transactions within the blockchain and summing up the UTXOs of the wallet.

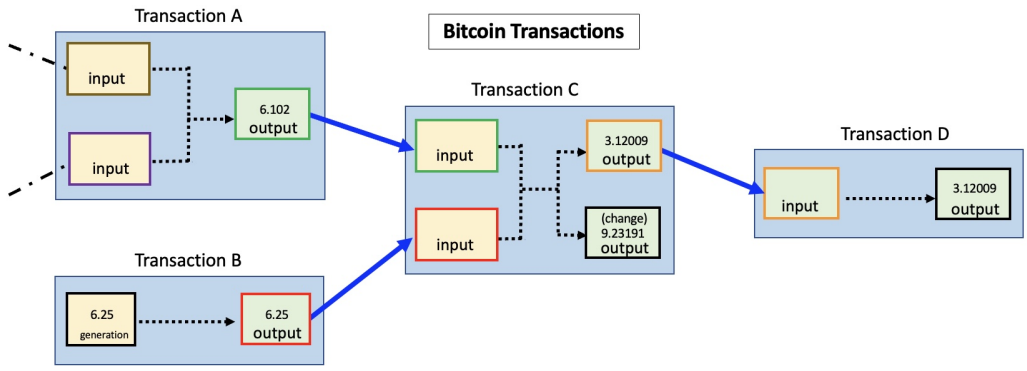

Figure 1.1: Bitcoin Unspent Transaction Outputs (UTxOs): each transaction output is an unspent output that can be used as an input in a future transaction.

Field	Description	Size
Version no	Currently 1. Set to 2 if you use `OP_CHECKSEQUENCEVERIFY` to enable timelocks	4 bytes
Flag	If present, always 0001, and indicates the presence of witness data	optional 2-byte array
In-counter	positive integer VI = VarInt	1 - 9 bytes
List of inputs	the first input of the first transaction is also called "coinbase" (its content was ignored in earlier versions)	<in-counter>-many inputs
Out-counter	positive integer VI = VarInt	1 - 9 bytes
List of outputs	the outputs of the first transaction spend the mined bitcoins for the block	<out-counter>-many outputs
Witnesses	A list of witnesses, 1 for each input, omitted if the flag above is missing	variable, see Segregated Witness
Lock Time	if non-zero and sequence numbers are <0xFFFFFFFF: block height or timestamp when transaction is final	4 bytes

Table 1.1: Record format of a Bitcoin transaction (inside a block)
Data from [**?**] page `https://en.bitcoin.it/wiki/Transaction`

Network Communication

When a wallet intends to transfer funds, the node owning the wallet broadcast the transaction to the Bitcoin network. The broadcasting happens via *gossip* peer to peer communication. The application level protocol used for this communication is JSON RPC. When nodes receive a transaction they do a

series of operations:

1. They check the transaction's validity: the transaction format is correct, the referenced outputs are not already spent and that the wallet has access to them.

2. They store the transaction by adding it to their *mempool* (a pool of unconfirmed transactions).

3. They broadcast the transaction to their peers.

On the network, transactions are identified by their **Transaction ID (TxID)** which is generated by simply hashing the transaction. One question which may naturally arise is: when nodes verify transactions, how do they know if a wallet "owns" the outputs it's trying to spend? Also how do we identify the recipient of the transaction if the network is anonymous? To address these questions we must introduce the scripting functionality of the network.

1.4.2 Scripting

Bitcoin has its own scripting language, called Bitcoin Script, which executes on a stack machine. Transaction outputs have multiple fields, one of which is the `script` field, also known as the **locking script**. This locking script specifies the conditions under which the output can be spent.

Field	**Description**	**Size**
Value	Transaction value in satoshis (1 sat = 10^{-8} BTC)	8 bytes
Script length	Positive integer	1-9 bytes
Script	A calculation which future values need to solve in order to unlock funds	Variable

Table 1.2: Record format of a TX Output

Data from [**?**] page `https://en.bitcoin.it/wiki/Transaction#Output`

The inputs, in turn, have a field `scriptSignature`. In order to verify that the input successfully unlocks the referenced output's funds, the output's `script` and the input's `scriptSignature` are concatenated and executed. If the execution terminates with `true` as the only value left in the stack, the transaction is valid.

Example of Script Execution

Here's an example of script execution:
Bitcoin's scripting language is deliberately non-Turing complete, lacking loops to enhance security (specifically avoiding DOS attacks on miners).

Field	Description	Size
TxID	Id of the transaction containing the referenced output	32 bytes
Output Index	The index within the transaction's outputs array	4 bytes
Script length	Positive integer	1-9 bytes
Script Signature	A calculation which future values need to solve in order to unlock funds	Variable

Table 1.3: Record format of a TX Input. Note that no value is specified as the value is determined by the output being spent

Data from [**?**] page `https://en.bitcoin.it/wiki/Transaction#Input`

1.4.3 Anonimity and Addresses

As the reader may have already noticed from the script example, wallets are identified by their public keys in transactions. When two wallet owners agree on a transaction, the receiving wallet generates a public key k_{pub} from its private key k_{priv}. From k_{pub} a bitcoin address is generated as follows:

```
Version = 1 byte of zeros // On the test network, this is 1 byte of ones
```

```
Key_hash = Version concated with RIPEMD-160(SHA-256(public key))
Checksum = 1st 4 bytes of SHA-256(SHA-256(Key hash))
Bitcoin Address = Base58Encode(Key_hash concat Checksum)
```

The payee communicates the address to the payer, who extracts the hash of k_{pub} from it and uses it to create an output with a locking script that basically says: "Whoever has this public key can access the funds in this output." When the recipient wants to unlock these funds, they reuse the public key they shared with the payer to craft an input that successfully those funds. This mechanism is called **Pay-to-PubKey-Hash**.
An important thing to note is that, in order to preserve anonimity, a different public key must be used to generate addresses for distinct transaction. Otherwise, it would be possible to link the two transactions to the same wallet.

1.4.4 Blocks and mining

For a transaction to be confirmed, it must be included in a **block**. Each block is composed of:

- **Header**: Contains metadata (version, timestamp), the previous block's hash, and a **nonce** (used in mining).
- **Body**: Holds transactions in a **Merkle tree** structure. The root of this tree is stored in the block header.

The structure of a Merkle tree is shown in 1.2 A merklee tree is chosen

- Enables pruning of old transactions.
- Allows lightweight nodes to verify transactions with partial tree data.

Blocks are crafted by miners, who:

1. Gather transactions from the mempool.
2. Verify them:

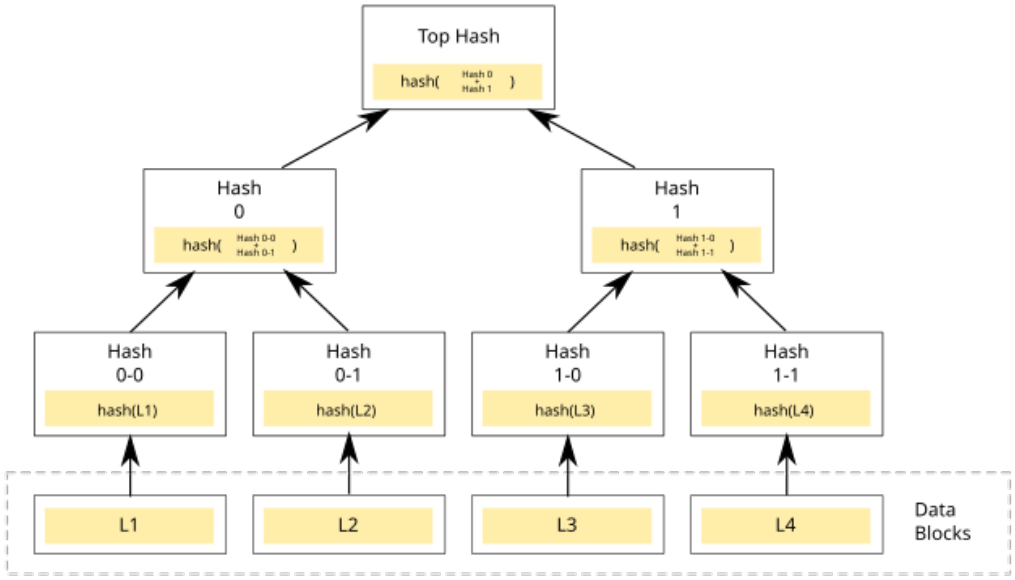

Figure 1.2: Merkle Tree structure: each leaf is a transaction hash, and each internal node is the hash of its children.

(a) Check UTxO validity.

(b) Validate scripts (locking/unlocking).

(c) Confirm metadata integrity.

A block is valid if it satisfies the consensus rules of the blockchain. This means that its hash satisfies the **Proof of Work** requirement and that the transactions contained inside are valid.

Proof of Work

As mentioned in 3.3.2, proof of work consensus requires solving complex cryptographic puzzles. In the bitcoin blockchain the puzzle is to find a hash whose numeric value is greater than a network-defined difficulty. This difficulty is adaptive and is adjusted every 2016 blocks (2 weeks approximately) so as to keep the block time to around 10 minutes. Miners adjust the **nonce**

in the block header to find a hash below a network-defined difficulty. This ensures:

1. Controlled block production to minimize the chances of forking.
2. Decentralization by preventing one miner from dominating.

Miner Rewards Miners consume a lot of energy and resources to find a Proof of Work so they are rewarded by the network with:

1. **Transaction fees**: Difference between inputs and outputs.
2. **Block rewards**: Newly minted Bitcoin (e.g initially `50 BTC` per block; this amount is halved every 210,000 blocks).

Forks and Consensus

When two miners create blocks simultaneously, a temporary fork occurs. Miners continue building on the branch they received first. The chain with more accumulated Proof of Work becomes the canonical chain, and the shorter branch is discarded.

1.4.5 Nodes

"Pure" nodes of the blockchain are supposed to store all of the blocks somewhere in order to verify by themselves the integrity of the chian and new blocks. However, as also noted in the original paper [**?**] the blockchain will likely outgrow the storage capabilities of some nodes at some point.
Thus the proposed solution was to allow for two types of nodes to exist:

- **Full nodes** store the entire blockchain and validate all transactions.
- **Lightweight nodes** store only block headers and query full nodes for details.

1.4.6 Network protocol

It is worth mentioning how the network layer operates as the implementation we provide in 3.3 will be similar. Bitcoin nodes communicate via a peer-to-peer protocol, ensuring decentralized communication between nodes. Transactions and blocks are propagated using a *gossip protocol*, where nodes relay received data to their peers. This dissemination ensures redundancy and robustness, even in the presence of node failures.
The messages that can be exchanged by nodes are defined in the Bitcoin wire protocol, which defines the structures for actions like broadcasting transactions, requesting block data, and maintaining connectivity. Key message types include:

- **INV (Inventory):** Used to advertise known transactions or blocks.
- **GETDATA:** Requests specific transactions or blocks by their identifiers.
- **BLOCK:** Contains full block data, including headers and transactions.

Communication between nodes typically occurs over TCP, with connections initiated via an handshake mechanism: a version message is exchanged and if both ends accept the other peer's version, they reply with *verack* messages and the connection is live. Typically after the handshake is completed, the nodes will use the *addr* and *getaddr* messages to exchange list of known peers. Nodes also periodically send *ping* and *pong* messages to monitor connection health.
To secure the network, transactions are identified by their hashed Transaction ID (TxID), and all data is verified through cryptographic signatures. The protocol is designed to handle network splits and eventual consistency, relying on the longest valid chain as the source of truth.

Chapter 2

Federated Learning and Machine Learning

2.1 Introduction to machine learning

One of the central topics of this document is Federated Learning, however, before introducing it in depth, we must first introduce the subject to which it belongs: machine learning. What is machine learning and what is its underlying problem? Machine learning is the discipline which seeks to answer the following question: *"how can a computer improve based on experience?"*. There are three main branches of machine learning:

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

Federated learning was originally born as a supervised learning technique, however recent work has been done to bring ito unsupervised [**?**] and reinforcement learning [**?**]. A short summary of all three learning methods is the following:

- Supervised learning about generalizing a dataset of examples to produce a general rule that can be applied to new examples.

- Unsupervised learning is about finding patterns and relationship between examples of the dataset.

- Reinforcement Learning studies how well a machine is able to adapt to a dynamic environmnet based on reward and penalties. When the machine does 'well' it receives a reward, otherwise it recieves a penalty.

For the sake of this document we will only cover supervised learning in a little more depth.
If the reader is interested in diving deeper into the topic of machine learning and gain a more thorough understanding of the techniques adopted and discussed here we suggest to take a look at [**?**].

2.2 Supervised Learning

Supervised Learning is the task of turning a dataset of input-output pairs into a general rule that should be able to produce the correct output even for inputs not necessarily seen in the original dataset.
The inputs and outputs are usually denoted respectively with the symbols x and y. Therefore, our rule will be the function f for which we would like $y = f(x)$.

2.2.1 Dataset

A dataset is a collection of unordered *observations* of input/output pairs. Each input is composed of a set of *features* or *attributes*, while the outputs contain *labels* or *targets*. Generally the inputs contain many features while the outputs are scalar values and every input and output of the dataset has the same features and labels, however this needn't hold true.
Suppose that the examples of the dataset have n features and that the number of examples in the dataset is m. The dataset can be represented as a pair of matrices X and Y where X is $m \times n$ and Y is $m \times 1$. The convention is to

represent vectors as columns, so the rows of X will be the transposed input vectors.
The features and the label can be categorized into:

- **Categorical**: For example relationship status (married, single, divorced, ...)
- **Real-valued**: For example weight or height
- **Binary**: Boolean values (true/false, 0/1, yes/no, ...)

As already stated, the goal is to produce a rule that generalizes well to the examples so that it can work on yet-unseen ones. This rule is called a *classifier* if the label is categorical, whereas it is called a *regressor* if it is real-valued.

2.2.2 Assumptions of supervised learning

We assume that the input/output examples contained in the dataset are chosen at random from the target population (the dataset is technically a *sample* of the population). Another assumption we make is that examples of the dataset are independent from each other and belong to the same distribution, which is called **IID Assumption** (independent and identically distributed). This assumption is quite strong and will also be discuted later in 2.5

2.2.3 Common Approach to Supervised Learning

A common approach to supervised learning is the so-called *discriminative approach*, which involves the following steps:

1. Define an error measurement.
2. Define a hypothesis space, which is the set of all the rules that we can accept.

3. Pick the rule inside the hypothesis space for which the error on the training set is minimized.

This approach has its own pitfalls: we might choose a rule that matches perfectly the training set but does not generalize well. This is called *overfitting* and can be seen in 2.1 polynomial below. The opposite of this is called *underfitting*. To mitigate this problem, we split the data into the **training set**

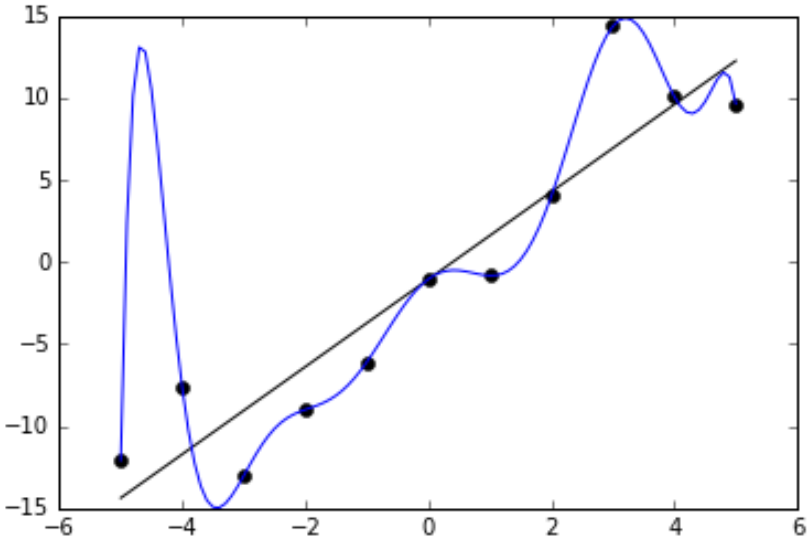

Figure 2.1: Overfitted data visualization. One can see that the polynomial fits perfectly the training set but does likely not generalize well.

Source: Wikimedia Commons, licensed under CC BY-SA 4.0.

and the **testing set**. The former is used for training our model and creating our rule, whereas the latter is used solely for testing our rule.

2.3 Gradient Descent Algorithm

Consider a function $f : \mathbb{R}^n \to \mathbb{R}$ at least of class C^1, that we want to minimize. In case of simple functions, it is possible to accomplish this task with precise mathematical methods. As an example consider the function $f(x) = x^2$. The minimum can be precisely found by studying its first and second derivatives.
However this is not always possible or practical to do. This is often the case in machine learning where the loss functions we want to minimize are very complex.
That's why optimization is usually performed via iterative methods, the most common of which is the gradient descent algorithm. There are multiple variations, but the general concept is to start somewhere in the parameter space (e.g $\mathbb{R}^n$), keep *"moving"* towards the direction of fastest decrease and stop once a satisfactory point of minimum is near.
The direction of fastest decrease of a differentiable function is given by the negative gradient of the function. This is true because by definition:

$$\frac{\partial f}{\partial v}(\vec{x}) = \vec{\nabla} f(\vec{x}) \cdot \vec{v} = \|\vec{\nabla} f(\boldsymbol{x})\| \cdot \|\vec{v}\| \cos\theta$$

Thus the vector which minimizes the directional derivative is the one in the opposite direction of the gradient (i.e. for which $\theta = -180°$).
The classic gradient descent algorithm works as follows:

1: Pick a starting vector $\vec{w}$ and a step size η.
2: **while** not convergence **do**
3: $\quad g \leftarrow \nabla f(\vec{w})$
4: $\quad \vec{w} \leftarrow \vec{w} - \eta g$

If the step size is too big, the minimum might be skipped; if it is too small, the algorithm might take a long time to converge. Usually, the step size is chosen adaptively.
When is convergence reached?

- The gradient is near to zero.

- The objective function is not changing by more than a certain threshold between steps.

2.3.1 Variations

Gradient descent techniques are a field of ongoing research thus it would not be feasible to cover all the variations. However we consider the case where the function f is a sum of functions $f(\vec{w}) = \sum_{i=1}^{m} f_i(\vec{w})$. In this case there are two canonical variations of the gradient descent algorithm: the batch and the stochastic gradient descent.

Batch Gradient Descent

The batch gradient descent is simply the same gradient descent algorithm described above with the difference that the gradient $\nabla f(\vec{w})$ is calculated via linear superposision of the gradients of the single functions $f_i(\vec{w})$:

$$\nabla f(\vec{w}) = \sum_{i=1}^{m} \nabla f_i(\vec{w})$$

So the algorithm becomes:

1: **while** not minimum **do**
2: $g = 0$
3: **for** $i = 1$ to m **do**
4: $g \leftarrow g + \nabla f_i(\vec{w})$
5: $\vec{w} -= \eta g$

This kind of algorithm is an *offline*, as $\vec{w}$ is changed only at the end.

Stochastic Gradient Descent

The stochastic variation of the gradient descent is different in that $\vec{w}$ is updated as soon as $\nabla f_i(\vec{w})$ is computed, without waiting to aggregate the gradients to compute $\nabla \vec{f}(\vec{w})$. In contrast to the previous, this is an *online* algorithm.

```
while not minimum do
    Shuffle points
    for i = 1 to m do
        g = ∇f_i(w⃗)
        w⃗ −= ηg
```

This version bounces back and forth much more than the batch version. For this reason, it is useful when we are far from the minimum but not as effective near the minimum. In fact, it is hard to figure out when to stop using this algorithm. The reader may have noticed that we shuffle the points here, unlike in the other version. This is because we might get stuck in a loop otherwise.

Mini-batch Gradient Descent

Mini-batch gradient descent is a blend between the two previous methods. We split the data into small batches and compute the gradient of the loss function on each batch, then we update the weights. This is the pseudocode:

```
B is the batch size
while not minimum do
    Shuffle points
    for i = 1 to m with step B do
        for j = i to (i + B) do
            g = ∇f_j(w⃗)
            w⃗ ← w⃗ − ηg
```

The reader may notice that this is a more general version of the two algorithms presented before. As a matter of fact, if we set $B = 1$ we get the stochastic gradient descent, while if we set $B = m$ we get the batch gradient descent.

2.4 Introduction to Federated Learning

To *federate* means "to join together under a central government or organization while keeping some local control" [**?**]. This principle forms the foundation of federated learning: uniting many independent devices under the coordination of a central server to collaboratively train a global machine learning model, all while ensuring that data remains decentralized and local to the devices.

Federated learning represents a significant paradigm shift in the field of machine learning, distinct from both traditional and distributed approaches. **Traditional machine learning** methods often rely on a centralized architecture, where all data is collected and stored in a central server. This centralization simplifies the training process but raises concerns related to data privacy, security, and scalability when dealing with sensitive or large-scale datasets.

Distributed machine learning, on the other hand, addresses scalability by distributing the computational workload across multiple nodes or servers. These nodes process subsets of data stored centrally or across a cluster. However, distributed learning still typically assumes that data can be transferred and aggregated freely, making it less suited for scenarios where privacy or data sovereignty is a primary concern.

Federated learning bridges the gap between these approaches by introducing a decentralized training methodology (with respect to the data). In federated learning, data remains on the devices where it is generated (e.g., smartphones, IoT devices, or edge servers), and only model updates—such as gradients or parameters—are transmitted to the central server for aggregation. This design inherently enhances privacy and security, minimizes data movement, and opens opportunities for leveraging massive amounts of distributed data that were previously inaccessible under traditional paradigms.

In this chapter, we will explore the mechanisms and architectures underpinning federated learning, starting with its foundational model, also called *vanilla federated learning*, and explore alternative approaches such as inte-

grating it with blockchain techonology.

2.5 Vanilla Federated Learning

Federated learning was first introduced by google researchers McMahan et al. in 2016 [**?**]. This version is oftern reffered to as vanilla federated learning and it was initially applied to train keyboard prediction models while prioritizing user privacy. Instead of transmitting raw data, devices locally train machine learning models and share only updated model parameters with a central server. This server aggregates these updates to refine a global model, ensuring data remains on devices and aligning with privacy and legal requirements.

2.5.1 Use Cases

Federated learning is well-suited for scenarios involving sensitive, decentralized data. The original use case, keyboard prediction, highlights its ability to leverage automatically labeled data generated in abundance by users. Another example is the automotive industry, where models for predicting battery life can be trained without centralizing data such as location, speed, and driving patterns [**?**]. Federated learning's privacy-preserving approach is critical in such applications.
Major tech companies have adopted federated learning for various purposes:

- Apple employs it to enhance Siri's voice recognition [**?**].
- Google uses it in Gboard for keyboard suggestions and in Google Assistant for speech recognition [**?**].
- Nvidia has developed FLARE, a framework for federated learning [**?**].

2.5.2 Architecture

The vanilla federated learning workflow involves the following steps:

1. The central server selects participating clients for a training round and sends them the updated global model.

2. Clients train the model on local data.

3. Updated model parameters are sent back to the server.

4. The server aggregates the updates to improve the global model.

5. The updated global model is distributed to the clients.

6. The process repeats until convergence or for a predefined number of rounds.

7. This architecture enables collaboration across distributed datasets while minimizing privacy risks.

Clients and datasets

As stated in the original paper [**?**], federated learning operates on a massively distributed scale, where the number of clients is large and the amount of data per client is on average of moderate to small size. Even though, in a usual federated learning scenario clients are every-day devices (e.g., smartphones, IoT, devices, and personal computers) with limited computing power, the small amount of local data makes the local training process relatively free.

Central Server for Aggregation

The central server orchestrates the process by sampling clients, collecting updates, and aggregating them into a global model. It operates without accessing raw data, instead, working solely with model updates, which it re-distributes for further training. This coordination ensures model consistency and adheres to privacy standards [**?**].

Aggregation Algorithm

The aggregation algorithm used by the central server plays a crucial role in the performance of the resulting model. One of the most common algorithms is the one introduced in the original paper: Federated Averaging (FedAvg) [**?**]. Let's define mathematically the federated learning problem. Suppose that we have K clients, each with a dataset D_k of size n_k. The loss function of each client is f_k and we want to minimize the globsl loss function:

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

We could use a simple batch gradient descent as described in 2.3.1 to accomplish the task. The central server would then aggregate the weights and perform the update as:

$$w_{t+1} = w_t - \eta \nabla f(w_t) = w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} \nabla f_k(w_t)$$

However, this in practice not feasible for two reasons:

1. We cannot rely on the fact that every client is always available.
2. This approach would likely end up having a slow convergence and communication overhead.

Federated Averaging is a slight variation of this algorithm and it introduces three parameters:

1. The batch size B
2. The number of local epochs E
3. The fraction of clients sampled C.

The server samples C clients at each round and each client uses performs local training minibatch gradient descent with batch size of B. The client performs E training rounds locally before sending the results to the server.

As we said in 2.5.2 the per-client cost of computation is relatively low. Thus this algorithm introduces E to provide a mechanism to reduce communication by using more computation: by performing more rounds locally the number of communication rounds with the server is reduced.

2.5.3 Advantages of Vanilla FL

Enhanced Privacy

The choice of keeping data local to the device reduces the risk of breaches and data leaks restricting the attack surface to the device itself only. This approach also complies with privacy regulations like GDPR[**?**] or CCPA[**?**]. However this does not mean that federated learning is a silver bullet for privacy concerns. Local gradients can still leak information depending on the model architecture and the aggregation algorithm used. In [**?**] the authors show how a model using words as features may be subsceptible to information leakage from gradients. They propose multiple solutions:

- Each client opens a bidirectional channel with all the other clients to generate masks complementary masks. This way, clients add the mask to their updates so that information leakage is minimized. And when the servers aggregates the weights the masks cancel out.

- Multiple architectures based on asymmetric encryption are explored.

However both of these approaches do not scale well as the number of clients grows because every client should communicate with every other client. Another solution is simply adding noise to the data but this comes with the tradeoff of reducing the model's accuracy. A more promising path to be explored is to use homomorphic encryption which allows to perform mathematical operations on encrypted data.

Scalability

By distributing training across numerous clients, federated learning avoids central data processing bottlenecks and accommodates large-scale systems. This scalability is particularly beneficial for applications involving IoT devices and smartphones [**?**].

2.5.4 Challenges of Vanilla FL

Heterogeneous Data

Unlike centralized training, where datasets are usually more homogeneus and can be analyzed beforehand *normalized*, In the case of federated learning data distributions often varies significantly across clients, posing challenges for model generalization. In technical terms this means that the data can be extremely unbalanced (meaning some clients may have plenty while others very few) and that the **IID** assumption which we said was at the core of supervised learning in 2.2.2 holds no more. However, as shown in the original article [**?**], federated learning proved to be robust to these perturbations.

Etherogeneous Clients

Another aspect to take into account, especially when developing aggregation algorithms, and architecturing the network is that clients engaging in federated learning may be very diverse and have different energy and computation capabilities. Consider FederatedAveraging: the number of local epochs E is a parameter that is the same for all clients. A smartphone of latest generation surely has more computational capabilities than a IIOT device such as a fridge. Tuning this parameter is challengenging and may lead to suboptimal results in any case.

Low-Quality or Malicious Contributions

Non-malicious but low-quality updates threaten the accuracy of the global model. There may also be malicious clients trying to inject a backdoor into the global model, trying to manipulate its predictions in a small subset of the parameters [**?**]. These two problems can be mitigated by using a **private** validation dataset on the server and rejecting all updates that do not satisfy it. However this introduces two new challenges:

1. Collecting the validation dataset.

2. Making sure that this dataset does not introduce any bias or filter out high quality contributions.

Centralization

Clients may not want to federate under a central server due to trust issues. The central server represents a single point of failure and if it was to be compromised, all members of the federated learning process would be at risk of receiving poisoned global models. Connecting to the previous section, clients may not know whether the server is validating the updates or may not trust the validation performed on them. Thus clients who are concerned with the security of the model likely would likely not participate.

Communication

As stated alredy, clients participating in federatd learning may be everyday devices such as smartphones or IIOT devices, where bandwidth and internet access may be limited or expensive. Federated Learning must be resilient to client crashes or communication failures and must require as less communication and bandwitdth usage as possible [**?**]. FedAvg already provides parameters to reduce communication overhead but the state of the art is still far from perfect. Another improvement that can be exploited to reduce bandwidth usage is to use a model compression algorithm (see [**?**])

Lack of Incentives

Client participation is voluntary, and without incentives, many devices may have no reason to join the process. Establishing a framework to reward contributions—through compensation or recognition—can improve participation and data diversity [**?**].

Chapter 3

Combining Blockchain and Federated Learning

3.1 Introduction

In this section we would like to explore the possibility of integrating blockchain technology with federated learning. As we mentioned in 2.5.4 centralization and lack of incentives are big problems in federated learning. In 3.1.2 we will discuss how implementing it on top of a blockchain network helps solve these problems. In 3.1.3 we will instead discuss the novel challenges and complexities introduced by mixing these two technologies.

3.1.1 Architecture

First of all let's discuss the workflow of the system and the possible implementation choices. Note that this is not a client-server model anymore. Thus the parties to which we referred as *clients* in 2.5 will now be referred to as *nodes* or *devices*:

1. The global model does not need to be pulled from a central server. Each node can retrieve it by inspecting the blockchain.

2. The nodes behaves as in 2.5.2: they train the local model on their own

data.

3. Once training is finished, the local update is broadcasted on the peer to peer network.

4. The update is put inside a block and becomes part of the blockchain.

5. When a new block is broadcasted to the network the clients update their model accordingly.

There are many implementation details that can be explored in more detail about this architecture and that's exactly what will be done now.

Global Model The global model will always be retrievable from the blockchain as all updates are recorded immutably. Thus one could reconstruct the global model by simply inspecting the blockchain history starting from the genesis block. In practice clients would just store it somewhere and gradually update it as new blocks are received. Clearly it is unreasonable to expect nodes to be online continuously, as the devices participating in federated learning often lack this capability 2.5. As a matter of fact when nodes come back online they can query other nodes in the network for the blocks they're missing and use those to perform the necessary updates.

Block creation This point was intentionally very vague in the presented overview. This is because it's heavily implementation-dependant: in a PoW blockchain the responsibility of creating new blocks and pooling updates is left to miners; in a PoS blockchain, the responsibility would fall to an elected node. Furthermore, should the updates be validated? And if so who should be responsible for doing so? Should the updates be stored singularly inside the block or only their aggregated value? Those question require careful analysis of the task at hand so it is not possible to provide a clear-cut answer beforehand.

3.1.2 Advantages of blockchain integration

Decentralized Aggregation

Decentralized aggregation as described in the last section, removes the need for a central server to handle model updates. This nullifies the concerns which were reaised in 2.5.4: there is no single point of failure anymore and the nodes do not need to trust anyone as they can verify the updates independently.

Incentivizing Participation

Implementing an incentive mechanism is not a possibility exclusive to blockchain. It is trivial to implement one in vanilla federated learning too: simply let every client have a balance on the central server and have the latter update it when contributions are received.

However this approach comes with the problems of centralization that have already been explained: clients have no guarantee that the server will distribute rewards fairly.

On the contrary a blockchain can be used to develop a transparent incentive mechanism, agreed upon by all nodes via the consensus algorithm such as demonstrated in [**?**].

Reputation and Accountability

Permanently recording contributions of each client in a transparent manner allows the development of a trustless reputation system, where clients that consistently provide high-quality updates can build credibility.

Conversely, malicious or unreliable contributors can be identified and penalized, enhancing model robustness and accountability. The reputation of nodes could also be integrated with the reward mechanism such that higher quality contributors are favored with more rewards whereas malicious contributors are penalized.

3.1.3 Challenges of Blockchain-Integrated FL

Model Size and Scalability

Integrating blockchain with federated learning introduces scalability challenges, particularly concerning model size. Storing large models or frequent updates directly on the blockchain is impractical, as it would overwhelm the network.
Here are some possible (non-exclusive) mitigations to this problem:

- Only store references to model updates on chain and store the data somewhere else. This doesn't necessarily mean that data should be stored centrally as a distributed storage system may be used such as IPFS. This would reduce the load on the blockchain but would introduce new problems such as data integrity and security.

- Have some nodes specialized in storage via Proof of Storage. This is the solution that is proposed in [**?**].

- Keep storing the updates on-chain and eventually prune old blocks when the storage amount exceeds the chain capabilities. This approach introduces non-trivial technical problems such as ensuring that all nodes consent and become aware of the pruning.

- Split nodes into lightweight and full nodes such as in 1.4.5.

- Use model compression to reduce data size.

As discussed in 2.5.4 also here it is vital to use a *relaxed* block frequency and perform more computation locally. In any case, even after performing the due optimizations, integrating blockchain in a federated learning task may not always be possible if we're dealing with devices with very limited storage capabilities.

Increased Overhead

Blockchain networks inherently introduce latency due to the need for consensus and block validation. This latency can slow down the learning task, as each model update must wait for confirmation on the blockchain before it is available for aggregation. As a result, blockchain-integrated federated learning may struggle to meet real-time or near-real-time model update requirements.

Another situation which would introduce a huge overhead would be forking. Forks are common in public blockchain networks, particularly those using Proof of Work (PoW) consensus. When a fork occurs, different parts of the network temporarily diverge, which means some nodes will train local updates on a model whereas some other nodes will do so on a different modal. Once the fork is resolved one of the two divergenct models will have to be discarded, leading to wasted computation. Thus minimizing forking risks is essential, potentially requiring a shift to alternative consensus mechanisms like Proof of Stake (PoS) or consortium-based approaches.

3.1.4 Validation

We previously said that, because blockchain allows updates to be transparent to all participants, no trust is required as updates can be validated independently. This is true if we assume that we are able to integrate validation in the consensus mechanism; however this is not trivial to achieve.

Suppose that nodes perform independent validation on the updates present in newly created blocks. This would break the consensus mechanism: some nodes may consider a block valid whereas other nodes may not.

We would get to the same conclusion even if we assigned validation to a subset of nodes (e.g the ones responsible for block creation). In order for validation to be included in the consensus mechanism it needs to be deterministic and consistent on all nodes. However this introduces a new problem: a malicious actor that is able to see the validation dataset would be able to generate a

backdoor in the model [**?**]. This problem can be addressed by using a DON (Decentralized Oracle Network) that can queried by the nodes to get a "yes" or "no" answer on the validity of the update [**?**]. This however raises the question of who should provide the validation dataset and the trustworthiness of the DON.

3.2 Discussion of implementation choices

Before moving on to our implementation 3.3 we would like to discuss pros and cons of different implementation choices. Refer to 1.3 for an explaination of the technical terms.

3.2.1 Consensus Mechanism

Proof of Work as consensus mechanism has been used in works such as BlockFL [**?**]. However PoW is not a good fit for our use case as it is very energy consuming and has a higher risks of forking compared to other consensus mechanisms. Proof of stake is more promising because it is energy efficient which is a strong requirement when dealing with low-power devices such as the ones participating in federated learning. Consortium-based consensus mechanisms are more centralized but are the most natural choice if our federated learning task is private or multi-layer. For instance in the case of a task distributed across hospitals, the hospitals could be the members of the consortium.

3.2.2 Public or Private

Let's first consider the fact that implementing a robust blockchain is by no means an easy feat and still a field of ongoing research. Leveraging a public blockchain for federated learning tasks would allow to reuse existing and battle-tested technology. Furthermore public blockchains are more decentralized and secure than private chains due to their higher level of participation.

However their cost may be prohibitive for this use case. Let's take ethereum as an example: the cost of storing 256 bits on chain is 20,000 gas according to the white paper [**?**]. This makes the cost of a kilobyte worth of storage 640,000 gas, which is about 20 USD at the time of writing. This makes storing weights on a public blockchain unfeasible.
Another aspect to consider is access-control: public blockchains do not offer access control mechanisms out of the box. Although, smart contracts could be used to implement application-layer access control, data would still be stored on chain and be accessible to everyone. If the federated learning task is private then a private blockchain is the most natural choice.

3.2.3 Using smart contracts

Smart contracts could be used to implement the logic of federated learning directly on chain. This includes updating the model, validating the updates, rewarding the nodes and so on. Nowadays, the existing frameworks for creating private blockchains, such as Cosmos or Hyperledger Fabric, come with smart contract support. However, following an approach similar to that of bitcoin 1.4 with limited or no scripting capabilities at all could be more lightweight. This is especially considering that a private chain reserved to federated learning would not need the power of smart contracts. As a matter of fact the implementation presented in this document is not based on smart contracts, as we'll see in 3.3.

3.3 Proposed implementation

In 3.1 we discussed extensively about the pros and cons of blockchain and federated learning integration. We also mentioned the main challenges that arise when trying to combine these two technologies. We should be explicit on the fact that it is a non-goal of this project to deal with all, if any of them. The main goal of this project is to provide a working implementation of a federated learning combined with blockchain and compare the performance of it with client-server vanilla FL. However this does not mean that no effort has been put into trying to develop a robust solution. In particular, great care has been put into the design of the system to improve upon designs suggested in papers such as [**?**], [**?**], [**?**]. In facts:

1. The model proposed in [**?**] uses the ethereum blockchain to implement a reward system. As mentioned in 3.2 storing the model and updates on a public chain would be unreasonable, thus the models are cleverly stored on IPFS instead of on chain. However this makes aggregating the updates on chain impossible so it still relied on a central server incurring in all problems related to centralization.

2. The model proposed in [**?**] defines a custom blockchain specific to federated learning. This is the same path that will be taken in this project. However the article based the architecture on PoW consensus, which is not suitable for a FL scenario as we argued in 3.2.

3. The model proposed in [**?**] is very similar to the one proposed in this project. It is basically a more featureful version of [**?**]: it integrates a reputation system in the consensus algorithm to minimize malicious updates to the model. The main problem is that it still relies on a variation of PoW.

The implementation presented here was thought to be a more efficient implementation of the last one of those articles. It still includes a reputation system but is based on a pure proof of stake consensus which substitutes the

resource expensive PoW. To sum up, the main goals of this implementation are:

1. To implement a fairly robust blockchain network specialized for a federated learning task.

2. To equip the blockchain network with a mechanism for validating updates so as to minimize the impact of malicious nodes.

3. To make the blockchain operations as lightweight as possible, in order to be friendly to low-power devices.

3.3.1 Network layer

The network protocol was heavily inspired by Bitcoin's, which we discussed in 1.4.6. For the sake of the experiment we left out the version exchange messages leaving the connection to be started by directly sharing known peers. The moment a node comes online it asks known peers to share the blockchain state so that it can synchronize with the network. As common in peer to peer networks we follow the *gossip* protocol to share information about the blockchain: pending transactions and blocks are all broadcasted to all known peers. Akin to bitcoin we implemented persistent connections and a ping/pong messaging system to detect unhealthy peers.

3.3.2 Consensus layer

As far as the consensus algorithm is concerned, we chose to implement the Tendermint algorithm. This is because the algorithm is robust and lays on strong mathematical foundations [**?**]. In particular, it is a byzantine fault tolerant algorithm, meaning that it can guarantee correct proceeding of the operations of the network even if some malicious nodes are present.
We want to give a basic understanding of how the algorithm works. First thing first let's deine the network model. Suppose that each node of the network has a voting power proportional to its stake or reputation and the

total voting power of the network is n. Let f be the number such that $n = 3f + 1$, then the algorithm works as follows:

1. **Proposal message**: A node is elected as block proposer and proposes a block; a round is started.

2. **Prevote message**: Nodes receive the block and cast votes in favour or against it depending on whether they consider it as valid.

3. **Precommit message**: When a node receives at least $2f/3$ prevotes in favour of the block it sends a precommit message and locks the chosen block. It cannot vote for any other blocks in the same round.

4. **Commit**: When a node receives at least $2f/3$ precommits in favour of a block it adds it to the chain.

5. **New round**: If a block is not committed in a round a new round is started.

The algorithm also implements timeouts in order not to incur into deadlocks. It is guaranteed that if at least $2f + 1$ nodes are honest the algorithm will always reach consensus.
We refer the reader to [**?**] if they're interested in a more technical explaination of the protocol and the proof of its correctness.

3.3.3 Blockchain layer

The blockchain layer is the core of the system. It is where the model updates and the reputation of the nodes are stored. The reputation of a node is determined by the amount of chain-native tokens it owns. These tokens can be seen as a form of currency. This currency is minted after successful updates of the model.
The general overview of the application-layer flow is the following:

- **Model update request**: A node trains a model on its local dataset and wants to share this update. It crafts a transaction containing the update and broadcasts it to the network. This transaction is called an **update transaction**.

- **Validation**: This transaction is stored inside the mempool of a validator. The validator tests the update to see whether it is malicious or low quality. If this is the case the update is discarded.

- **Block creation and reward**: When one of the validators who received the update is chosen as proposer, it gathers the transaction along with all the other valid transactions inside a block. A **coinbase transaction** is also added to it, in order to reward all the contributors.

- **Consensus**: The block is then broadcasted to the network. The nodes of the chain will then reach agreement on whether to accept the block or not based on the consensus algorithm [**?**].

Let's dive into more details about the choises now.

Reward

Rewarding nodes with tokens provides an incentive mechanism to encourage participation in the network. This is one of the lackings of Vanilla FL as we discussed in 2.5.4 and we try to overcome it in this way in our implementation. Ideally we would want to reward nodes based on the quality of their update. For simplicity we suppose that the rewards are fixed in our system.
In facts it is not trivial to determine the quality of an update: in contrast to centralized machine learning, federated learning participants dont't share a global testing dataset. Thus quality would have to be judged using local datasets.
In this scenario a validator node may judge an update as very high quality whereas another one may judge it as barely acceptable. This would lead to consensus issues, because the former would decide to reward generously the

contributor, whereas the latter would not. We cannot even decide to just leave the responsibility of judging the quality of the update to the single proposer, because this introduces bias in the reward system.

Security All transactions contain the public key (address) of the creator node and a signature to prove the ownership of the update (see 3.3.5). Participants of the network can be addressed by their public key, similarly to bitcoin 1.2. This guarantees integrity of messages exchanged with peers in the network and is essential to build a reputation system.
Although this is a good start there is still a major flow in the proposed architecture. Signatures don't provide confidentiality and everyone can read signed data and re-sign it as theirs. This is a problem because a peer node can perform a **man-in-the-middle** attack on update transactions and steal the update by signing it with its own key. This is a problem that can be solved with a more sophisticated protocol or/and using more advanced cryptography such as commitments or homomorphic encryption. For the sake of this experiment we ignore this problem as we are not concerned with network-level attacks.

Validators The main idea behind having validator nodes is to provide a mechanism against malicious updates. As we stated already in 3.1.3, in a blockchain based on PoW, we cannot implement a network-wide validation mechanism, because asking every node to validate blocks with their own independent dataset would break the consensus algorithm. On the other hand if we were to delegate the validation to miners only, then powerful attackers could easily craft malicious blocks.
The solution to this problem is to incorporate a voting mechanism in the consensus algorithm. This is what has been done by choosing the Tendermint algorithm. If at least 2/3 of the votes of the network are casted in favour of a proposed block then it is valid.
However, we don't want all nodes of the network to be held accountable for validation. This is because being a validator requires:

1. Having an high quality validation dataset.
2. Having enough computational power to frequently validate transactions.
3. Being frequently available, otherwise the network would be stuck.

The majority of federated learning devices doesn't satisfy all three of those requirements. Incidentally, restricting the number of validators contributes to making network convergence faster and is more communication efficient.
If restricting the validators to a network subset we must also take care of prevenging malicious nodes from becoming validators however. This problem can be solved by implementing a reputation system based on the amount of tokens held by nodes. If we assume that malicious nodes will be penalized and not rewarded well over the course of the FL task then only honest nodes will own the amount of required tokens to become validators.
Note that restricting the number of validators also secures the network against *sybil attacks* (a kind of attack where a malicious actor creates a large number of identities to gain control of the network.)

Staking In order to preventing validators from disrupting the network we require them to stake funds. This successfully prevents *nothing at stake attacks*: ideally validators that misbehave should be punished by burning their stake, either entirely or partially. However, for the purpose of the experiment we did not implement it.
In our network, a staking event is just a transaction where a node asks to stake some of its funds via a **stake transaction**.

Block proposers In order to prevent chaos and risks of forking only one amongst the set of validators is chosen to propose a block in a given round. This is also mandated by the consensus algorithm that we chose 3.3.2. In our network the block propser is chosen with a deterministic random algorithm. This is done via seeding the RNG using the current block height and round number. 3.1

Codice 3.1: Random proposer selection

3.3.4 Technology stack

The implementation of the project was primarily carried out in Python, which is the predominant language in the field of machine learning. Python offers an extensive ecosystem of high-quality libraries, making it an ideal choice for tasks related to both federated learning and blockchain integration. For the machine learning component, we utilized PyTorch, a versatile and widely-adopted library that simplifies model development and training.
For federated learning, we leveraged Flower, a Python library specifically designed to facilitate the implementation of traditional federated learning systems. While Flower excels in client-server-based architectures, integrating it with a blockchain network required modifications to its configuration. These adjustments were necessary to adapt the framework to a decentralized, peer-to-peer architecture.
On the blockchain side, the network layer initially employed WebSocket communication. However, the asynchronous nature of WebSocket messaging presented challenges in ensuring reliable and consistent communication at the application level. To address these issues, we transitioned to using gRPC (Google Remote Procedure Call), a language-agnostic RPC framework that runs on top of HTTP/2.0. gRPC allows for efficient, bi-directional streaming of messages, with message formats defined using the Protocol Buffers syntax. While gRPC inherently follows a client-server paradigm, this limitation was overcome by designing each node to function as both a client and a server. This dual-role approach enabled seamless peer-to-peer communication while retaining the reliability and performance benefits of gRPC.

3.3.5 Messages and Data structures

Here are the common data structures used in the network and blockchain layers:

This is the RPC service definition:

3.4 Evaluation and Results

3.4.1 Evaluation Metrics

The main goal of the evaluation is to assess and compare the performances of our blockchain implementation against flower's Vanilla FL implementation. We want to explore two scenarios:

- All nodes of the system are honest.
- Some nodes of the nodes of the system are malicious.

The metric for the comparison will be the accuracy of the model. We will showcase the behaviour of the loss function and the accurancy across different rounds. The data and accuracy are obtained via evaluation of the model against a test dataset on a per-round basis.

3.4.2 Experimental Setup

The experiments were conducted using multi-class logistic regression on the Iris dataset, which is a classic dataset used in machine learning. The dataset is composed of 150 samples of iris flowers: the features of an example are the width and lenght of petal and sepal of the flower; the label is the kind of flower (versicolor, setosa or virginica). In order to train the classifier we preprocessed the dataset by one-hot encoding the labels. The loss function used for this experiment was the cross-entropy loss function.
The experiment was carried out on a personal computer; due to limited resources and the small size of the dataset, the number of nodes partaking in federated learning was set to 8. In the case of the blockchain implementation only two validators were used. When testing with malicious contributions we set the average amount of malicious contribution to be around 1/4 of the total contributions per round. In all experiments nodes trained for fifteen epochs locally. The total number of rounds of the federated learning task was set to 10.

3.4.3 Expected Results

We expect that the blockchain implementation will have a better accuracy than the standard vanilla FL model in the scenario where some participants are malicious. This is the result that was demonstrated in [**?**]. On the other hand we expect the client-server federated learning task to perform and converge way faster than the blokchain implementation in the case of honest participants. None of the two implementations are expected to have the upper hand in terms of accuracy in this scenario. As a matter of fact, implementing FL on a blockchain shouldn't impact the training quality as it represents just a different means to gather the weights.

3.4.4 Results showcase

The results of the experiments are shown in the figures below. Let's consider first the case where nodes are honest 3.1 and 3.2. As expected, the vanilla FL implementation converges nicely and with high accuracy very quickly. The blockchain implementation on the other hand as a steep drop at the start but then proceeds to converge all the same. The drop at the start may be due to the fact that nodes are still synchronzizing as it coincides with the bootstrap of the blockchain.
The results of 3.3 and 3.4 are also in line with our expectations. The blockchain implementation is able to handle malicious nodes and is more stable whereas the vanilla FL implementation is more bouncy.

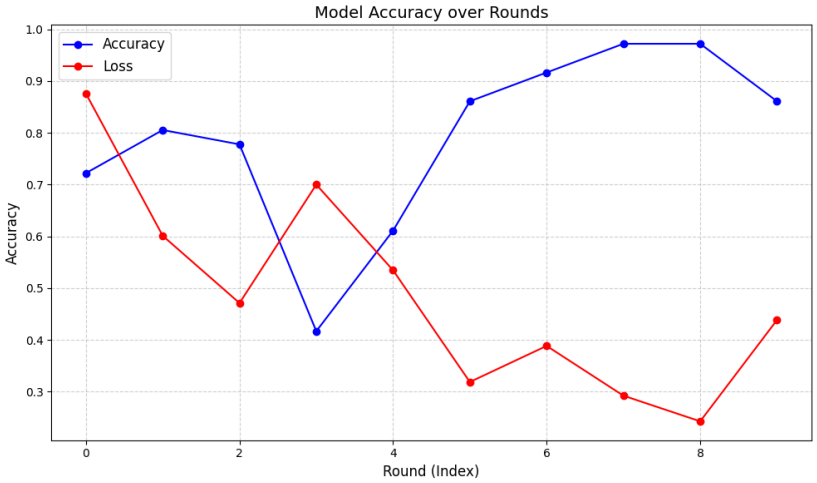

Figure 3.1: Federated Learning on a blockchain where nodes are all honest

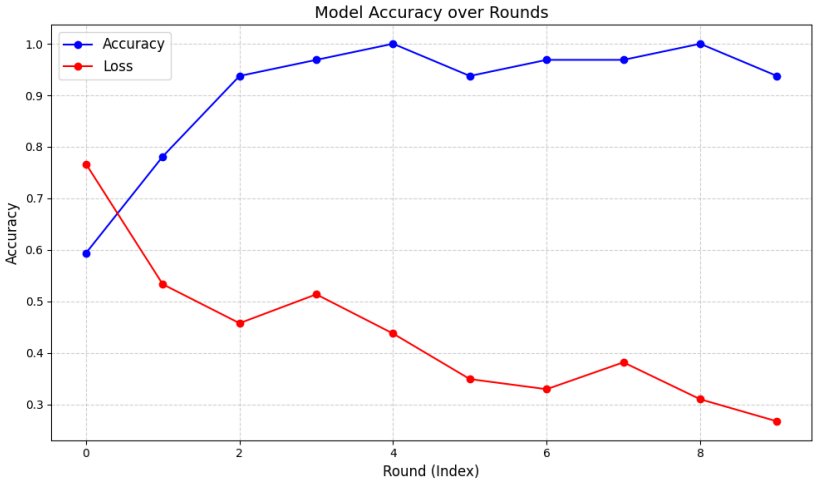

Figure 3.2: Vanilla Federated Learning where nodes are all honest

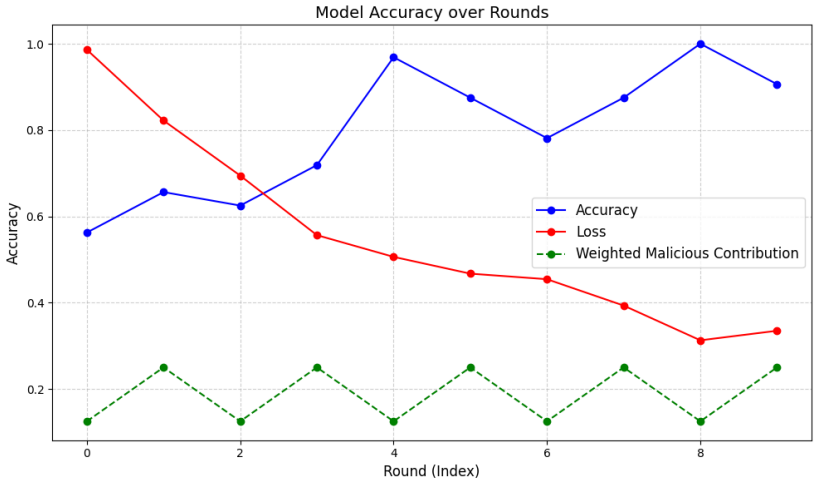

Figure 3.3: Federated Learning on a blockchain where some nodes are malicious

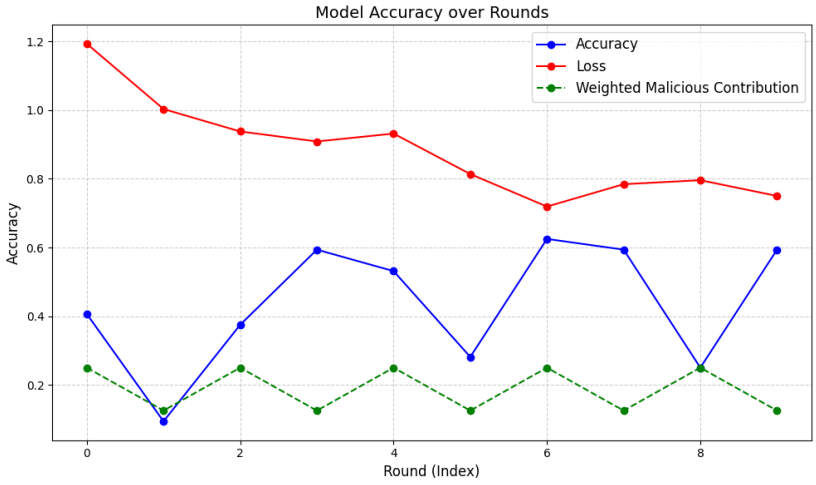

Figure 3.4: Vanilla Federated Learning where some nodes are malicious

Conclusion

After all the topics discussed it's time to sum up and draw the conclusions. This thesis has explored the integration of Federated Learning and Blockchain, focusing on their potential to address critical challenges in privacy, security, and decentralization. Federated Learning facilitates collaborative model training without centralizing data, while Blockchain enhances trust through its decentralized and immutable ledger.

Key Findings

- **Enhanced Security and Privacy**: Combining Federated Learning with Blockchain mitigates reliance on a central server, addressing privacy concerns and trust issues inherent in traditional FL systems.
- **Improved Transparency and Trust**: Blockchain's transparency ensures model integrity, while smart contracts and consensus mechanisms provide accountability.
- **Scalability Challenges**: The integration introduces significant computational and communication overhead, especially in environments with constrained resources.

Contributions

- A Blockchain-augmented FL framework was proposed, demonstrating its feasibility through both theoretical analysis and practical implementation.

- Comparative evaluations against vanilla federated learning revealed that using blockchain is a valid choice as it does not significantly impact the performance of the learning task.

Remarks

In our experiment we have dealt with a very simple dataset of limited size and complexity. Furthermore we only used ten nodes and the communication happened not even on a local network but via the loopback address of the machine. While the results are promising, they should be taken with a grain of salt as they may not be representative of real-world scenarios.

Future Directions

When discussing the challenges we put at stake many problems at once but only addressed few of them in our implementation.
Furthermore we have not dealt with other practical limitations. For instance if we're interested in deploying a real-world Blockchain FL system we would have to consider the fact that most user-owned devices have access to the internet only via NAT and therefore are not able to accept inbound connections. This would make the idea of using a peer to peer network unfeasible.
Another very important problem we have not dealt with is storage scalability. As we have highlited in 3.1.3 storing all the updates of the model on the blockchain can outgrow storage capabilities of FL participants very fast.
Our conclusion is that the integration of FL and Blockchain is promising but remains in its early stages.

Bibliography

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," bitcoin.org, Tech. Rep., 2008.

[2] Bitcoin wiki. [Online]. Available: https://en.bitcoin.it/wiki/

[3] M. Servetnyk, C. C. Fung, and Z. Han, "Unsupervised federated learning for unbalanced data," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[4] J. Qi, Q. Zhou, L. Lei, and K. Zheng, "Federated reinforcement learning: techniques, applications, and open challenges," *Intelligence & Robotics*, 2021. [Online]. Available: http://dx.doi.org/10.20517/ir.2021.02

[5] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, 1st ed. The MIT Press, 2012.

[6] O. U. Press. (2024) Federate. Accessed: 2024-11-17. [Online]. Available: https://www.oxfordlearnersdictionaries.com/definition/english/federate

[7] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016. [Online]. Available: https://arxiv.org/abs/1602.05629v1

[8] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, M. D. Mueck, and S. Srikanteswara, "Energy demand prediction

with federated learning for electric vehicle networks," 2019. [Online]. Available: https://arxiv.org/abs/1909.00907

[9] A. Inc. (2019) Accessed: 2024-11-18. [Online]. Available: https://web.archive.org/web/20221207175040/https://nips.cc/Expo/Conferences/2019/Schedule?talk_id=40

[10] G. Inc. Accessed: 2024-11-18. [Online]. Available: https://federated.withgoogle.com/

[11] NVIDIA. Nvidia flare. Accessed: 2024-11-18. [Online]. Available: https://github.com/NVIDIA/NVFlare

[12] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[13] E. Parliament and C. of the European Union. (2016) General data protection regulation (gdpr). Accessed: 2024-11-18. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679

[14] C. L. Information. (2018) California consumer privacy act (ccpa). Accessed: 2024-11-18. [Online]. Available: https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375

[15] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," 2016. [Online]. Available: https://arxiv.org/abs/1611.04482

[16] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," 2019. [Online]. Available: https://arxiv.org/abs/1807.00459

[17] H. Wang, S. Sievert, Z. Charles, S. Liu, S. Wright, and D. Papailiopoulos, "Atomo: communication-efficient learning via atomic sparsification," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 9872–9883.

[18] N. T. Cam and V. T. Kiet, "Flwrbc: Incentive mechanism design for federated learning by using blockchain," *IEEE Access*, vol. 11, pp. 107 855–107 866, 2023.

[19] A. Baldominos and Y. Saez, "Coin.ai: A proof-of-useful-work scheme for blockchain-based distributed deep learning," *Entropy*, vol. 21, no. 8, p. 723, Jul. 2019. [Online]. Available: http://dx.doi.org/10.3390/e21080723

[20] C. Mazzocca, N. Romandini, M. Mendula, R. Montanari, and P. Bellavista, "Truflaas: Trustworthy federated learning as a service," *IEEE Internet of Things Journal*, vol. 10, no. 24, pp. 21 266–21 281, 2023.

[21] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, 2020.

[22] E. Foundation. (2015) Ethereum. Accessed: 2024-11-20. [Online]. Available: https://github.com/ethereum/wiki/wiki/white-paper

[23] H. Chen, S. A. Asif, J. Park, C.-C. Shen, and M. Bennis, "Robust blockchained federated learning with model validation and proof-of-stake inspired consensus," 2021. [Online]. Available: https://arxiv.org/abs/2101.03300

[24] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on bft consensus," 2019. [Online]. Available: https://arxiv.org/abs/1807.04938

[25] P. Kairouz, H. B. McMahan, and B. A. et. Al., "Advances and open problems in federated learning," 2021. [Online]. Available: https://arxiv.org/abs/1912.04977

[26] L. Meng, Y. Guo, S. Song, and C. Cui, "A research and analysis of blockchain federated learning," in *2023 IEEE International Conference on Electrical, Automation and Computer Engineering (ICEACE)*, 2023, pp. 94–98.

[27] N. Agrawal, D. Mishra, and S. Agrawal, "A comprehensive survey on blockchained federated learning system and challenges," in *2023 IEEE International Conference on ICT in Business Industry & Government (ICTBIG)*, 2023, pp. 1–4.