Computer Communications 236 (2025) 108112

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Decentralized coordination for resilient federated learning: A blockchain-based approach with smart contracts and decentralized storage

Stefano Ferretti *, Lorenzo Cassano, Gabriele Cialone, Jacopo D'Abramo, Filippo Imboccioli

Department of Computer Science and Engineering, University of Bologna, Mura Anteo Zamboni, 7, Bologna, 40126, Italy

ARTICLE INFO

Keywords:
Resilient federated learning
Blockchain
Deep learning
Smart contracts

ABSTRACT

Machine Learning (ML) in distributed environments increasingly deals with sensitive data (like healthcare or financial records) that cannot be centrally stored or processed due to privacy concerns. Federated Learning (FL) addresses this by enabling model training across decentralized devices, but faces significant challenges including system reliability, node failures, and trust issues among participants. Traditional FL approaches often rely on centralized coordinators, creating single points of failure and potential security vulnerabilities. This paper presents a novel approach to FL that leverages smart contracts, blockchain, and decentralized storage to enhance the traceability and reliability of the learning process. Our proposed system architecture is fully decentralized, eliminating single points of failure and promoting cooperation through a rewarding mechanism. Unlike previous approaches that neglect node fault tolerance, we introduce a smart contract based scheme for managing node failures and electing the aggregator node. The presence of the smart contract, executed on a decentralized permissioned blockchain, provides reliability guarantees and eliminates the need for costly distributed algorithms in terms of message exchange. An experimental study is conducted to evaluate various aspects of the FL system. We present results related to the accuracy and effectiveness of the FL system on ML models. We also examine the performance related to the distribution of the weights of the ML model based on the use of IPFS. Furthermore, we analyze the performance of the smart contract in terms of gas consumption. Lastly, we investigate the impact of failures combined with incentive policies and aggregator election algorithms on the FL system. Our findings demonstrate the viability of the proposed approach, paving the way for more robust, reliable, and efficient FL systems.

1. Introduction

In today's digital age, organizations and individuals generate vast amounts of sensitive data that could greatly benefit from Machine Learning (ML) analysis, particularly in critical domains such as healthcare, finance, and cybersecurity. However, this data often cannot be shared or centrally processed due to privacy concerns, regulatory requirements, and the risk of unauthorized access. Traditional ML approaches, which require centralizing all training data, are therefore unsuitable for these sensitive scenarios. Federated learning (FL) is an innovative Machine Learning (ML) paradigm that addresses these challenges by enabling multiple participants to collaboratively train a common model, while maintaining the confidentiality of their respective datasets [1]. This framework is particularly advantageous for enhancing data-driven learning algorithms, as it upholds the privacy of data owners and involved data subjects, while still allowing them to benefit from collective knowledge.

In this sense, FL significantly contributes to the concept of data sovereignty, which emphasizes the right of individuals and organizations to retain control over their data [2]. By enabling participants to train a shared ML model without exchanging raw data, FL supports the principle that data should remain within the legal jurisdiction or geographic boundaries of its origin. This approach aligns with the growing global emphasis on data protection regulations, such as the General Data Protection Regulation (GDPR) in the European Union, which mandates strict data governance and privacy standards [3]. The decentralized nature of FL allows for the creation of robust ML models while ensuring that each participant's data remains under their governance, thus empowering them with the autonomy to manage and dictate the terms of their data's utilization. This paradigm shift not only fosters trust among stakeholders but also paves the way for a more ethical and responsible use of data in the digital age.

FL's potential is notably significant in several application realms, extending the possibilities offered by ML systems, going from healthcare

* Corresponding author.
E-mail addresses: s.ferretti@unibo.it (S. Ferretti), lorenzo.cassano2@studio.unibo.it (L. Cassano), gabriele.cialone@studio.unibo.it (G. Cialone), jacopo.dabramo@studio.unibo.it (J. D'Abramo), filippo.imboccioli@studio.unibo.it (F. Imboccioli).

https://doi.org/10.1016/j.comcom.2025.108112
Received 26 June 2024; Received in revised form 20 November 2024; Accepted 22 February 2025
Available online 19 March 2025

to intrusion detection system, up to crowd-sensing [4–8]. As concerns medical diagnostics, for example, the use of FL can safeguard patient data privacy and yet permit the aggregation of knowledge (i.e., ML model parameters) coming from diverse data sources [9]. Nonetheless, an important challenge within this domain is the assurance of secure data sharing by FL collaborators, adhering to the established protocol. The crucial point is the creation of a dependable and secure logging and coordination system that can effectively manage the concerted actions of all parties involved. Thus, given the distributed nature of FL, traceability becomes a critical aspect. It involves tracking the participation of each node, the changes made to the model, and the contributions of each participant. This is crucial for ensuring accountability, verifying the integrity of the learning process, and facilitating auditability. However, traditional FL methods usually fail in providing a robust and tamper-proof traceability mechanism. This is where blockchain technologies come into play [10].

In this paper, we introduce DeSCo-FL (Decentralized Smart Contract based Federated Learning), a FL framework that incorporates blockchain technology to synchronize the training procedure and guarantee the resiliency of the system. The synergy of blockchain with FL not only ensures the integrity and immutability of the training process but also solidifies the commitment to individual data privacy. Our proposed system leverages the inherent security features of blockchain to establish a transparent and tamper-proof environment, fostering trust among participants and providing reliability also in case of node failures. Through this integration, we aim to address the critical concerns of data security and privacy that are paramount in applications where sensitive data are used, e.g. healthcare, pervasive applications.

Compared to the approaches available in the literature, our approach is based on the use of a system grounded on smart contracts. The blockchain not only serves as a trusted log for recording weights but also transforms into a platform where the smart contract coordinates the actions of the participants in the FL system. The decentralized and distributed nature of the blockchain ensures availability for the execution of the smart contract. Furthermore, this smart contract based coordination approach provides traceability guarantees on the training phase.

Another difference between our approach and some seminal ones that used blockchain and FL together, is that we do not use the blockchain ledger to store ML model weights directly. Rather, we resort to a decentralized storage, namely IPFS [11], as the place where weights are stored, while uploading in the smart contract the digests of these weights. This approach has some benefits. In fact, the upload of the digest into the blockchain provides guarantees on the correctness and untamperability of model weights. At the same time, decentralized storage systems are cheaper and more responsive than blockchains. This allows uploading even complex ML models made of a large number of parameters.

The proposed system is designed to handle with participant failures. Typically, the literature on FL systems focuses on the mechanisms to exchange and aggregate the weights of ML models, assuming that the underlying distributed system is reliable. This assumption is reasonable in some contexts where participants have sufficiently reliable connection capabilities, such as in a public health context, where the participants are hospitals collecting patient data. However, this assumption cannot be taken for granted in other contexts such as crowd-sourcing, pervasive communications, vehicular networks, and scenarios with limited network connectivity. The use of a coordination system based on smart contracts allows defining change policies (election) of the node that plays the role of "Aggregator", i.e., the node that receives ML model parameters from others and computes a new set of parameters, which are then distributed back to the "Collaborators" for further training. In this paper, we show how it is possible to implement such policies and the impact they can have on the FL.

Considering the fact that the role of the Aggregator can change over time, and given the presence of a blockchain system based on smart contracts, we have equipped the system with a rewarding mechanism to incentivize nodes to follow the protocol. This approach encourages active and consistent participation, fostering a more robust and reliable FL environment. The rewarding mechanism serves as a motivational tool, promoting adherence to the protocol and enhancing the overall system effectiveness. This approach, therefore, not only ensures the smooth functioning of the FL system but also contributes to the integrity and success of the learning process.

An initial version of this study was previously published in [6,12]. The fundamental concept on the use of FL, blockchain and data storage technologies, in conjunction with the coordination of smart contracts, remains consistent. However, the present paper represents a comprehensive revision and expansion of the original work. Specifically, in [6] a preliminary version of the smart contract-based architecture was presented, under the assumption that no node failures occur. In [12], another preliminary version of the work was present, with an initial ML study based on a different dataset compared to the one used in this work. This was set in a simplified scenario where Collaborators may fail, but it was assumed that the Aggregator would always remain online, acting as a reliable server. The enhancements introduced in this paper significantly extend the scope and allow for the handling of possible failures of nodes participating to the FL process. This provides additional evidence supporting the feasibility of implementing resilient FL strategies in practice. In essence, the contributions of this work are directed in the following ways:

- In DeSCo-FL, we introduce an innovative coordination system for the traceability of the FL process, based on smart contracts, blockchain, and decentralized storage. The system architecture is completely decentralized, thus avoiding the presence of a single point of failure. At the same time, the FL system is based on the presence of Collaborators and one of these nodes is elected as the Aggregator. This role can change over time among the various Collaborators. The use of rewarding mechanisms provides incentives to cooperate.
- We present a system for managing node failures and electing the Aggregator node based on a decentralized system. The presence of the smart contract (executed on a permissioned blockchain) provides reliability guarantees and does not require the use of costly distributed algorithms in terms of message exchange. In the paper, we discuss three possible variants of Aggregator election.
- We conducted an experimental study aimed at studying the various aspects involved in the FL system, showing the viability of the proposed approach. First, we present results related to the accuracy and effectiveness of the FL system on ML models. Second, we study the performance related to the distribution of the weights of the ML model based on the use of a decentralized storage, i.e., IPFS. Third, we study the performance of the smart contract in terms of gas consumption. Fourth, we study the impact of failures combined with incentive policies and Aggregator election algorithms in the FL system.

The remainder of this paper is organized as follows. Section 2 provides the background needed for the rest of the paper, as well as the state of the art. Section 3 describes the proposed DeSCo-FL system architecture. Section 4 describes the methodology to assess the system, and presents the obtained experimental results. Section 5 provides some concluding remarks.

2. Background

Federated Learning (FL) is a Machine Learning (ML) approach where multiple edge devices collaboratively learn a shared prediction model while keeping all the training data on the original computing nodes, decoupling the ability to do ML from the need to store the data in the cloud. This approach addresses critical issues such as data privacy,

data security, data access rights and access to heterogeneous data. It is not the aim of this section to provide a complete and broad discussion on possible FL models, since there is a vast literature on the topic. The interested reader may find interesting surveys that cover all these aspects, e.g., [13,14].

With this in view, in this section we will primarily deal with the combination of FL with the use of blockchain technologies, trying to motivate why the union of these two technologies can be of help in various applications. In fact, the convergence of FL and blockchain can potentially address several challenges in the data-driven world. For instance, the integration of these technologies can enhance privacy-preserving data analysis, secure multi-party computation, and robust decentralized systems.

2.1. Federated learning

There is plenty of application scenarios where it is impractical to share data among participants involved in a ML study. There may be technical impediments, i.e., the costs for transferring data and train the models locally with the whole dataset is prohibitive, and legal impediments, i.e., the treated data are sensitive ones, so data cannot be disclosed to external sources for computation. FL can be of help in these contexts [14,15]. FL allows the training of ML models without exposing the training data [1]. This is done by training the models locally on each data source, and then only exchanging the model parameters between devices. The FL protocol consists of two alternating steps:

- Local training: Each data source trains a model on its own data. The nodes that locally train their ML model based on their data are referred as "Collaborators".
- Parameter aggregation: The ML model parameters from each device or data source are aggregated to create a new global model. Usually, the aggregation of the local values computed by Collaborators is performed by a node referred as the "Aggregator".

This process is repeated until the model converges. The local training step can be done using any ML algorithm. However, some algorithms are more suited for FL than others. For example, algorithms that are less sensitive to the amount of data are more likely to work well with FL [16]. The parameter aggregation step can be done using a variety of methods. A common approach is to use a weighted average of the model parameters from each device. The weights can be assigned based on the size of the dataset from each device, or they can be assigned randomly. FL can be used to train a variety of ML models, including classification models, regression models, and clustering models.

FL has several advantages over traditional ML methods. It can protect the privacy of the training data, it can be used to train models on distributed data, and it can be more efficient than traditional methods. Thus, FL is well suited to train models on sensitive data, such as medical data or financial data. Moreover, it is a good solution to employ when data is distributed across multiple devices, such as smartphones or IoT devices. In this kind of scenarios, it can be more efficient than traditional ML methods, as it does not require the transmission of large amounts of data.

In this work, during our tests we used two specific FL weight update methods, inspired by [17]. The first approach, is a classic one called *FedAvg Aggregation*, that assumes that all Collaborators contribute equally. Thus, when weights need to be updated at the end of a loop iteration, the novel weight is just the average value of weights received from the Collaborators. The second approach, Federated Proximal (*FexProx*), removes the uniformity assumption of *FedAvg*, by adding a regularization term that is added to the computation of the average weight values [17].

2.2. Decentralizing trust

Trust is essential but difficult to establish in a distributed system where participants are often unknown. In a traditional centralized system, trust is often placed in a single authority or organization. However, this can make the system vulnerable to corruption or failure [3]. Indeed, centralized systems are often associated with inefficiencies and single points of failure. Moreover, these systems are susceptible to corruption, manipulation, and data breaches, eroding trust and undermining the integrity of the entire system. The need for decentralized trust mechanisms has become increasingly apparent, prompting researchers and practitioners to explore alternative solutions.

Blockchain and Distributed Ledger Technologies (DLTs) can help to decentralize trust by providing a secure and transparent way to verify transactions and agreements. By leveraging cryptographic techniques and decentralized networks, blockchain technologies provide a transparent, immutable, and tamper-resistant platform for recording and verifying transactions. In addition, these technologies also introduce the concept of smart contracts, which eliminate the need for intermediaries to oversee and enforce agreements, by relying on code and cryptographic protocols. Furthermore, smart contracts enhance security and trust. The code underlying smart contracts is immutable and tamper-resistant, ensuring that the terms of the agreement cannot be altered once deployed on the blockchain. This eliminates the risk of fraudulent activities or disputes arising from ambiguous contract terms. Needless to say, it is fundamental to ensure that smart contracts are free from vulnerabilities, as they form the backbone of a decentralized system application, in this case, a FL system, and any security flaw could potentially lead to significant issues on the efficacy of the ML application [18].

2.3. Federated learning and blockchain

In the literature, there is a plethora of papers that already introduced the idea of using blockchain as a support for FL [7,19,20]. Actually, many of them are in fact short papers proposing somehow the use of these two technologies only, e.g., [21–25] However, many examples of applications are available, going from on-device FL [26], dynamic resource allocation and client scheduling in wireless communications [27,28], digital twins and 6th generation (6G) mobile networks [29], vehicular networks [30], crowd-sensing [31–33], smart healthcare [6,34], supply chains in industrial settings [10].

The idea is to leverage the inherent synergy between the use of a decentralized ML system and the blockchain technology to build a transparent and decentralized coordination platform. The rationale is that security and privacy of the model are guaranteed by the adoption of a trusted ledger. Moreover, this approach helps towards a more user-centric model, that enhances the perception of safety around sensitive data [35]. Although many proposals use blockchain as a secure way to store the global model, to the best of our knowledge, all these scientific contributions do not take into account issues concerned with possible failures of participants, which is one of the main aspects considered in DeSCo-FL.

Fig. 1 shows a simplified view of the different system architectures that are usually proposed in the literature. The model on the top-left part is the classic client/server based FL system, with one Aggregator that receives data from Collaborators. We already discussed on the simplicity and effectiveness of this approach, as well as its pitfalls due, essentially, to the presence of a single-point of failure, i.e., the Aggregator.

Some approaches resort to a classic FL approach, with the presence of an Aggregator and multiple Collaborators, augmented with a blockchain (see top-right system model in Fig. 1). These approaches typically assume the Aggregator being always online. Thus, in this case the idea is to pass through the blockchain to store the ML model parameters, instead of using message passing. Thus, Collaborators publish

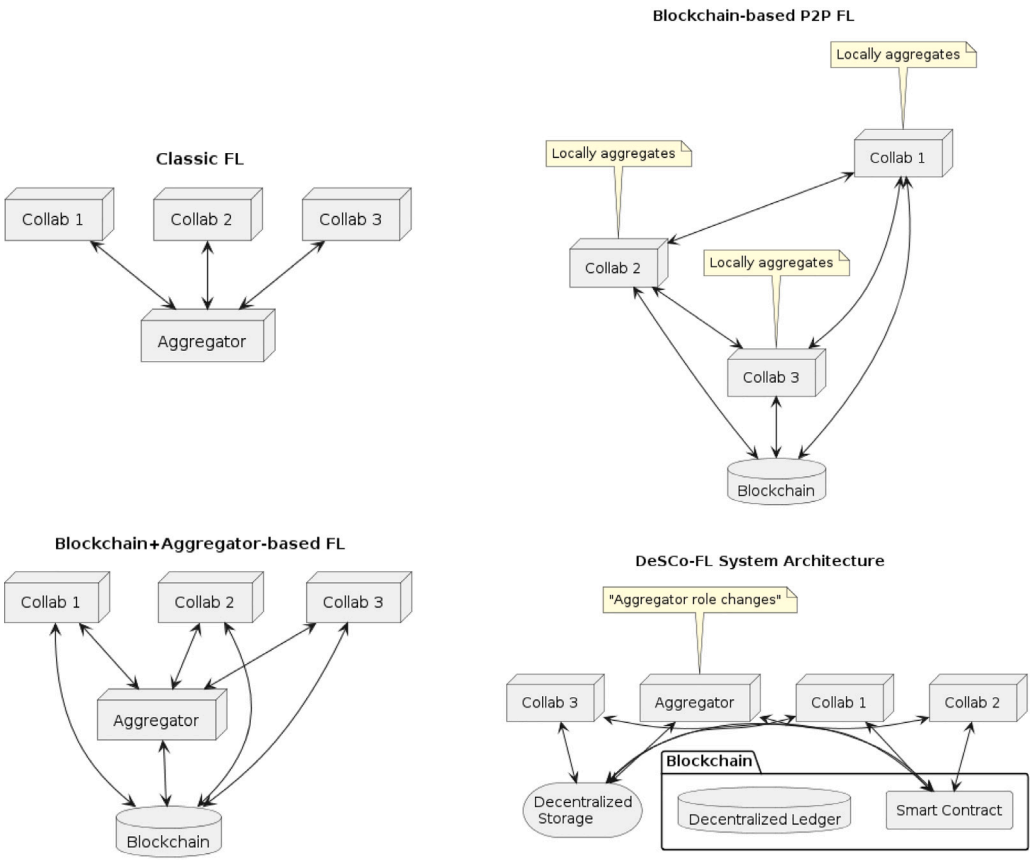


Fig. 1. FL system architectures.

their parameters in the blockchain; then, the Aggregator retrieves them and computes ad updated version for the global model. Examples of works that adopt this approach are [23,24,36–40]. Some of them add the presence of an external decentralized storage such as IPFS [5,41], as DeSCo-Fl does.

Other solutions, instead rely on the blockchain as the data storage where ML parameters are to be stored and retrieve by all participants (see bottom-left system model in Fig. 1). The idea is that the use of a decentralized blockchain eliminates the necessity for a central Aggregator in the FL training. In its place, a shared unalterable ledger is employed to consolidate the global model and disseminate global updates to learning Collaborators for direct computation at the device level. This decentralization of model aggregation not only removes the risk of single-point failures, enhancing training reliability, but also lightens the load of a centralized aggregation, particularly when the FL system is composed of numerous Collaborators, e.g., edge computing and crowd-sensing applications. ML model parameters are added to unchangeable blocks for information exchange among Collaborators during training. The distribution of blocks across the entire network enables all participants to verify and track the training progress. Examples of proposals in this direction are [22,42–44] In this kind of pure peer-to-peer solutions, it is required that all nodes aggregate their data on their own. Thus, the aggregation model is shared and executed in parallel by multiple nodes. This requires redundant computations that we avoid in DeSco-FL, while avoiding the presence of a single point-of-failure typical of a client/server FL model. Moreover, in the case of a peer-to-peer local aggregation, if a node goes offline for a certain amount of time, it might need to check the complete history of the FL updates, in order to check and collect the most recent ones. The use of a smart contract eases this process, since it allows to get the latest model updates by simply calling a related function. In the next section, we explain all the details of our DeSCo-FL proposed protocol, while in the rest of this section we review some of proposals available in the literature.

It is interesting to observe that the usual approach is to see the blockchain as a ledger where models parameters can be registered. Thus, in most papers, a discussion on the mining activity is combined somehow to the FL process. Only in few cases, these proposals foresee the presence of smart contracts [25,45]. Smart contracts can indeed be of help in order to coordinate the learning process or to measure how each FL Collaborator contributes to the global model by implementing a reward-based incentive mechanism [46–48] An example of this concept is presented in [34], where an incentive-based approach is introduced. This approach involves the creation of a Non-Fungible Token (NFT) marketplace that manages access to patients' historical medical data within a medical Federated Learning (FL) system. In our approach, the smart contract not only assumes the responsibility of providing rewards but also orchestrates the FL process. It regulates all phases of FL training and manages the election of the Aggregator.

2.4. Fault-tolerance in federated learning

Fault tolerance is another important consideration in FL, as nodes in every kind of distributed systems may frequently join and leave the system, leading to stragglers and failures. At the time of writing, only few works dealt with this issue in FL, such as [49], where authors propose an evaluation of fault tolerance in FL systems, specifically examining the impact of unreliable clients on model performance. The authors analyze two real-world classification problems and find that simple FL algorithms can perform surprisingly well despite unreliable clients, suggesting that traditional assumptions about fault tolerance may need reevaluation.

In [50], some strategies are studied for ensuring fault tolerance in distributed learning environments, including FL. The work highlights

common failure modes such as hardware errors and adversarial attacks, proposing robust distributed stochastic gradient descent techniques to mitigate these issues.

In [51], a scheme called SAFARI is presented to addresses challenges of communication efficiency and fault tolerance. SAFARI uses sparse learning on local clients to reduce communication overhead. It employs a similarity-based compensation method to handle missing model updates due to unreliable communications.

The paper in [52] presents a framework to address challenges in FL with non-IID data, including vulnerability to Byzantine attacks and gradient inversion attacks. The work classifies clients into clusters based on data similarity to mitigate intra-cluster heterogeneity. A privacy-preserving Byzantine fault tolerance strategy using cosine similarity is implemented within each cluster.

With respect to our approach, none of these works make use of smart contracts to cope with fault-tolerance issues.

3. DeSCo-FL system architecture

In this section, we first discuss the components of the DeSCo-FL system architecture. Then, we introduce the main protocol for the training of FL system. Finally, we present the protocol to handle nodes failures, with special attention to the failures of the Aggregator node and the election of a novel one.

3.1. System components

Our system architecture is comprised of four key components. The first two are the classic actors involved in a typical FL system, while the last two ones are those of typical decentralized systems [53]:

- **FL Collaborators**: these are the nodes participating to FL model training. They generate (or obtain from local sources) data, locally train their models, and upload the trained parameters to the system. Collaborators may have a different throughput, i.e., different computational capabilities and different datasets (of different sizes) to be analyzed during the training.
- **FL Aggregator**: it collects the parameters of the ML model used in FL and aggregates them in order to calculate novel measures for these parameters.
- **Permissioned Blockchain**: This blockchain executes a Federated Learning Smart Contract (FLSC). This smart contract autonomously regulates the exchange of parameters among participants, ensuring that no one, including the Aggregator, can deviate from the protocol. It also prevents tampering with the parameters, as the digests of these parameters are stored in the blockchain, thus enhancing security [54]. FLSC manages interactions by organizing the FL process in phases, accepting only certain types of inputs during each phase. This method offers a simpler and more secure way to manage the FL training process. In this work, we assume that no fees are associated to blockchain transactions. This is possible since we envision the use of a permissioned blockchain and it implies that there is no economic disincentive to publish in the blockchain.
- **Decentralized Storage**: specifically the InterPlanetary File System (IPFS), which stores the parameters of the ML model.

An overview of the DeSCo-FL system architecture is the bottom-right sub-figure in Fig. 1.

The choice of a permissioned blockchain without transaction fees for the DeSCo-FL system is driven by the need to handle sensitive data securely. In privacy-sensitive domains such as healthcare, financial services, or industrial applications, permissioned blockchains offer several advantages:

- **Access Control**: Participating nodes must be authorized, ensuring that only legitimate entities participate in the model training process.
- **Privacy Guarantees**: The closed nature of the network provides an additional layer of privacy protection for sensitive training data.
- **Performance Optimization**: Without the need for costly consensus mechanisms like Proof of Work, permissioned networks can achieve higher throughput and lower latency.
- **Regulatory Compliance**: In regulated industries, permissioned networks make it easier to implement and demonstrate compliance with data protection requirements.

However, we acknowledge that public blockchain implementations could be advantageous in scenarios where the training data is not sensitive or has been adequately anonymized.

3.2. DeSCo-FL protocol

The diagram of the interactions among the four types of system components, when no node failures occur, is shown in Fig. 2. (We will describe how to deal with node failures in the next subsection, after having introduced the main behavior of the system components.) Interactions between participants never occur directly, but only through FLSC (and IPFS, as far as data exchange is concerned). This scenario also allows for a certain anonymity among users, although the system is permissioned, so the presence of sybils or malicious should be drastically reduced.

The protocol works as follows.

Open and start phases

1. The (initial) Aggregator opens a FL Smart Contract (FLSC) for handling the FL interactions and adds multiple Collaborators (only one Collaborator is depicted in the figure for the sake of simplicity).
2. Collaborators subscribe to the FL Smart Contract.
3. The Aggregator publishes the model to be used and the related information to compile it in the FL Smart Contract.
4. Collaborators retrieve the model from the FL Smart Contract and adopt it.

Learning phase

5. Each Collaborator performs the training using its own dataset.
6. Periodically, Collaborators publish their own updated local parameters on IPFS and send the digest of these parameters (plus info to retrieve them) to FLSC. Parameters are ciphered using, in sequence, the private key of the Collaborator and the public key of the Aggregator.
7. The Aggregator retrieves and decrypts all these parameters using, in sequence, its own private key and the public key of the Collaborator. This approach ensures that only the Aggregator can read the data (confidentiality) and guarantees that the data have been generated by that specific Collaborator (verifiability). Then, the Aggregator further checks the validity of the parameters by verifying that their digest is equal to the hashed value stored in the smart contract.
8. The Aggregator updates the ML model parameters based on an aggregation function designed to compute the best parameters to be used.
9. The Aggregator uploads the updated ML model parameters (ciphered using an identical approach as the one explained above) on IPFS. Moreover, the Aggregator stores the digests of the parameters in the smart contract.
10. Collaborators retrieve the updated model parameters from IPFS, decrypt them and check their validity through FLSC (as above). Then, Collaborators adopt these model parameters during the next iteration of the training process.

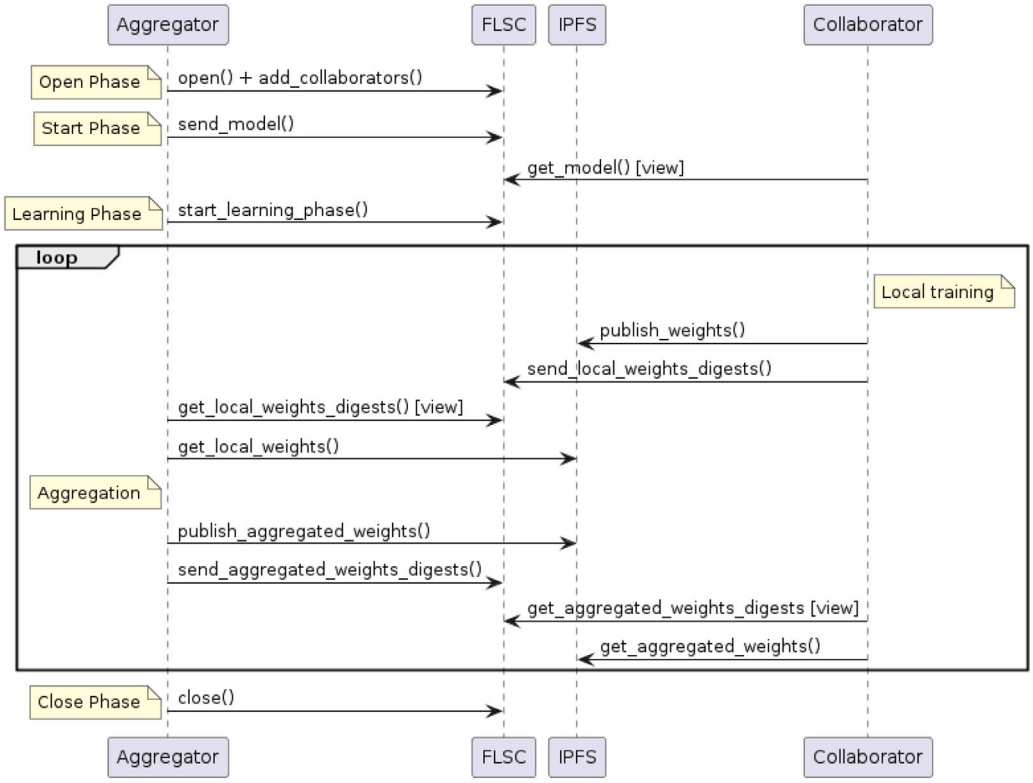


Fig. 2. System diagram (no failures case).

Regarding the task of updating model parameters calculated by all Collaborators, our primary objective in this paper is not to introduce a novel FL aggregation technique. Instead, our focus is on the feasibility of FL and blockchain technologies to enhance the verifiability and reliability of parameter exchange, as well as the resiliency of the whole FL system. Consequently, in this study, we opt to employ and compare the two cited seminal aggregation algorithm for computing parameter updates, i.e., *FedAvg* and *FedProx* [13].

3.3. The FL smart contract

We implemented the FLSC in Solidity, to be executed on top of an Ethereum Virtual Machine compliant blockchain. The contract structure includes state management, collaborator management, and functions for data exchange. The contract defines four states for FL: OPEN, START, LEARNING and CLOSE, corresponding to the state phases described in Fig. 2. These states govern the progression of the FL process, ensuring that it proceeds according to predefined rules. Furthermore, the phase definitions facilitate synchronization among all parties involved in FL, effectively addressing issues related to node failures. This implies that if a node fails or is late in transmitting the hash of its weights, the training process is not hindered, and weight updates proceed without considering the contribution of that particular Collaborator. (More on these aspects is discussed in next subsections.)

Collaborators are added to the contract during the OPEN state, allowing only authorized users to participate. Collaborators can be added by the contract owner, and their interactions are monitored throughout the FL process. In Fig. 2, we highlight the view functions, which correspond to methods within the smart contract that do not alter the contract state. These functions do not involve gas consumption for their execution and provide read-only access to specific data or parameters. The contract includes security measures to restrict unauthorized access and ensure that actions are carried out only by authorized users. During each round of the FL, Collaborators can perform specific actions only once. Finally, the contract emits events to signal state transitions during the FL process, providing transparency and auditability.

In Listing 1, we show an excerpt of the interface of the contract code. The smart contract implements a role-based access control system leveraging OpenZeppelin's AccessControl framework. This design represents an advancement over traditional owner-centric smart contracts, enabling more flexible and secure management of the federated learning process.

More specifically:

1. **Role-Based Access Control:** The contract implements a hierarchical permission structure with:
   - *Administrator:* i.e., the creator of the contract, that manages the overall contract governance by adding collaborators. The system thus maintains a clear separation between administrative duties (managed through `DEFAULT_ADMIN_ROLE`) and operational coordination (managed through the aggregator role). This separation enables flexible governance while maintaining operational security.
   - *Aggregator:* oversees the FL process and manages state transitions. Thanks to the OpenZeppelin's *AccessControl* contract, the aggregator role is defined using a unique 32-byte identifier generated by applying the Keccak-256 hash function to the string "AGGREGATOR_ROLE" [55]. This approach follows Ethereum's standard practices for role-based access control, ensuring efficient gas consumption and providing a secure identifier for permission verification throughout the contract's lifecycle.
   - *Collaborators:* participate in the training process by contributing model updates. They are added by the Administrator during the OPEN state phase only.
2. **Access Control Granularity:** The contract implements fine-grained access control through custom modifiers:

```
contract FederatedLearning is AccessControl {
  // Core FL data structures [some omitted for the sake of brevity]
  bytes32 public constant DEFAULT_ADMIN_ROLE; // contract creator
  bytes32 public constant AGGREGATOR_ROLE = keccak256("AGGREGATOR_ROLE");
  enum FL_STATE { CLOSE, OPEN, START, LEARNING }
  FL_STATE public fl_state; // init to FL_STATE.CLOSE
  address[] public collaborators;
  address public aggregator;
  bytes public aggregatedWeightsInfo;
  mapping(address => bytes) public weightsInfo;
  uint256 public roundTimeout;

  // Core events [some omitted for the sake of brevity]
  event EveryCollaboratorHasCalledOnlyOnce(string functionName);
  event AggregatedWeightsReady();
  event RoundProceeded();
  event TimeoutReported(address reporter);

  // Access control modifiers
  modifier onlyAggregator();
  modifier onlyAuthorized();

  // Collaborator and FL management (only admin)
  function open() public onlyRole(DEFAULT_ADMIN_ROLE);
  function addCollaborator(address collaborator) public
                      onlyRole(DEFAULT_ADMIN_ROLE);
  function sendModel(bytes memory _model) public onlyRole(DEFAULT_ADMIN_ROLE);
  function close() public onlyRole(DEFAULT_ADMIN_ROLE);

  // Failure handling functions
  function reportAggregatorFailure() public onlyAuthorized;

  // FL process management functions
  function learning() public onlyAggregator;
  function sendAggregatedWeightsInfo(bytes memory _weights) public onlyAggregator;
  function electNewAggregator() internal;
  function reportTimeout() public onlyAuthorized;
  function distributeRewards() internal;

  // Collaborator Functions
  function getModel() public view onlyAuthorized returns (bytes memory);
  function getLocalWeightsInfo() public view onlyAuthorized;
  function sendLocalWeightsInfo(bytes memory _weights) public onlyAuthorized;
  function getAggregatedWeightsInfo() public view
                     onlyAuthorized returns (bytes memory);
}
```

Listing 1: Core components of the Federated Learning smart contract

- `onlyRole(DEFAULT_ADMIN_ROLE)`: this modifier is defined in the OpenZeppelin's AccessControl contract, that restricts the addition of Collaborators to the Administrator of the contract;
- `onlyAggregator`: Restricts critical FL management functions, that can be issued by the Aggregator only;
- `onlyAuthorized`: Enables both the Aggregator and registered Collaborators to participate in the FL process.

3. **Failure Handling:** The contract has a mechanism for handling both collaborator and aggregator failures:
   - Timeout-based detection of node failures;
   - Collaborative reporting system for timeout triggers;
   - Dynamic aggregator election with multiple policies.
4. **State Management:** The contract maintains the already mentioned four distinct states governing the FL process:
   - `OPEN`: Enables collaborator registration;
   - `START`: Initiates the training process;
   - `LEARNING`: Manages weight submissions;
   - `CLOSE`: Concludes the training round.

The phase of weight aggregation and updating occurs in two different cases:

- In the case where all Collaborators have called the function `sendLocalWeightsInfo()` for that round. At this point, the last collaborator emits the event `EveryCollaboratorHasCalledOnlyOnce(''sendLocalWeightsInfo'')`. The Aggregator listens for this event and is thus informed that it is necessary to create a new aggregated version of the weights.
- In the case where the number of `TimeoutReported` events emitted by the Collaborators exceeds a certain threshold. This occurrence also triggers the weight updating phase.

Upon the occurrence of one of these two cases, the Aggregator is responsible for retrieving the weight digests from the FLSC, along with the information needed to retrieve the weights, aggregate them, update the weights, and upload the information to retrieve them via `sendAggregatedWeightsInfo()`. Subsequently, the Aggregator emits an `AggregatedWeightsReady()` event, which prompts the collaborators to retrieve the new weights.

If the Aggregator fails to perform this task within a certain timeout, the Collaborators can agree on the failure of the Aggregator through a `reportAggregatorFailure()`, resulting in the automatic election of a new Aggregator (`electNewAggregator()`) when the number of notifications exceeds a certain threshold.

3.4. On the use of rewards in DeSCo-FL

All the activities performed by the participants, i.e., Aggregator and Collaborators, might be rewarded or not, depending on the application scenarios. In our previous work in this topics [6], we assumed that no rewards were used. This is quite reasonable in scenarios such as public health, for instance. In fact, in these kinds of applications we can assume that Collaborators are hospitals and all are honestly motivated to follow the protocol and share their results for social good. In these scenarios, not using rewards allows for simplicity in the FL protocol that results simpler and faster, focusing solely on the learning task. However, in other application scenarios, e.g., crowd-sensing based applications, the absence of rewards could lead to a lack of incentive to contribute valuable data or computational resources. Moreover, potential for free riding appears, i.e., some Collaborators might take advantage of the contributions of others without contributing much themselves.

Thus, in a broader vision, using a rewarding mechanism can introduce some potential advantages in a FL environment:

- Incentivization: Rewards can incentivize participants to contribute more data or computational resources, potentially improving the overall performance of the FL model.
- Fairness: A reward system can ensure fairness by compensating participants based on their contributions. This could encourage more active participation and collaboration.
- Quality Control: If rewards are tied to the quality of the contributed data or model updates, this could motivate participants to provide higher quality inputs.

On the other hand, implementing a fair and effective reward system can be complex. It requires careful design to avoid potential pitfalls and abuses. In fact, the introduction of rewards could potentially amplify a security issue within the FL system, known as a poisoning attack. This type of attack occurs when malicious training nodes manipulate their local models or data to introduce harmful information or disrupt the model training process within the FL system. Such an attack can lead to inaccurate predictions or biases, thereby affecting the overall performance and reliability of the system [40]. Moreover, if these malicious nodes are rewarded, it could further incentivize them to upload deceptive parameters, or even just redundant data, exacerbating the problem. Hence, the reward system, while beneficial, must be designed with these considerations in mind to ensure the integrity and effectiveness of the FL system.

For this reasons, in this work we adopt a rewarding solution that stays in the middle between not having rewards at all and tracking and verifying all the contributing of participants. In practice, our rewarding scheme provides incentives to participate as Collaborator and

Aggregator. Thus, the protocol does not face directly with issues due to potential poisoning attacks, but we assume this issue can be mitigated by the presence of a permissioned blockchain and a decentralized storage, where all contributions are traced. The check on the veracity of FL data and model parameters might need the presence of verifiers, typically employed in other decentralized application scenarios, e.g., crowd-sensing and proof-of-location applications [56,57].

The Aggregator earns a fee for each round. This encourages nodes to try to assume and maintain this role. Aggregating data costs computational time, so in general it could be expected that only a subset of nodes can apply to act as an Aggregator. In this study, however, this aspect is not considered.

Collaborators earn a fee every time they send the weights of their model. This encourages them to participate. We assume that Collaborators are positive in participation. In other words, they are not encouraged to do less than is possible (they are not "lazy"). In this study, we focus on failures due to technical impediments. Consequently, the fee is independent of the amount of data used to locally train the model. This choice was made for various reasons:

- In certain scenarios it is not said that it is the merit of the Collaborator to process the various data. For example, in a healthcare context, the number of patients managed is not attributable to a merit of the ML system deployed in the hospital.
- Having a fee proportional to the amount of data could favor malicious behaviors, in which a Collaborator pretends to have had at his disposal a greater amount of data, compared to the real quantity. The use of possible solutions, which give incentives proportional to the amount of data used, is left to future developments. In fact, solutions based on zero knowledge proof and use of decentralized (and anonymized) identities of patients could be studied, to avoid the generation of synthetic fake data.
- In addition to the aspects related to the quantity of data, there would then be to consider the aspect related to the speed of processing the data available. The assumption of non-laziness, facilitates the study and the model. However, this is also an interesting future development.

In any case, the purpose of the fees is to provide a tangible incentive for participants to contribute to the FL process. It is worth noting that in our system design we refer to a permissioned blockchain, whose associated tokens may have minimal economic value. Therefore, their use in such a context can represent a symbolic reward that acknowledges the effort and resources contributed by the participants. This symbolic reward can foster a sense of fairness and motivation among participants, encouraging them to actively engage in the FL process. However, the context of using this solution can be viewed within a broader vision. Imagine being in an edge computing environment, where ML data analysis involves studying patterns from local observations tied to a specific territory, e.g., crowd-sensing based applications. The collaborators participating in the FL are thus analysis nodes that contribute to creating a decentralized ML environment that improves through the exchange of ML model weights. On the other hand, these nodes are maintained by entities (or individuals) participating in an ecosystem of services, among which data analysis is just one. The cryptocurrency earned can then be exchanged in other scenarios, transferring it from the permissioned blockchain system to a more general context, for example, through bridging techniques. Clearly, for the FL rewarding service to function effectively, the FL setting must be recognized within this ecosystem. This recognition would ensure that the reward can be utilized to access other services.

3.5. Node failure management

Due to the decentralized nature of the data storage (IPFS) and of the blockchain, we can assume that these system components are generally available. However, we should assume that participants to the FL model can fail. Two types of node failures are to be handled, i.e., the failure of a Collaborator, and the failure of the Aggregator.

Our protocol operates in a setting with a fixed set of participants $\mathcal{P} = \{p_1, \ldots, p_N\}$, where N is the total number of nodes. Unlike traditional FL approaches, our model explicitly addresses node failures and recovery scenarios while maintaining a pre-defined set of trusted participants. Each participant p_i maintains a local dataset D_i and participates in the training process according to the FL protocol. The smart contract maintains the list of authorized participants, their roles (Collaborator or Aggregator), and their current status (active or failed).

3.5.1. Collaborator failure

As explained in the previous section, a training round ends when all Collaborators publish their weights in the data storage and notify the FLSC through the transmission of the related digest. Upon transmission by all Collaborators, the FLSC triggers an event that is collected by the Aggregator, which then retrieves the model parameters and aggregates them. However, during a training round, if a Collaborator fails, it would not be able to upload its weights and notify the FLSC, which could halt the training phase. This issue can be addressed by having other nodes set a timeout. Once this timeout expires, online nodes can invoke a specific function of the FLSC to transition to the aggregation of the ML parameters gathered from active Collaborators.

It is important to note that this is a delicate situation for the FL system. Given that the FLSC is a smart contract, it cannot set a timeout independently and must be "awakened" by an external component. Assigning this task solely to the Aggregator is not feasible, as it could be faulty, like other participants. Moreover, even if we assume that Collaborators are honest, relying on a single node could lead to scenarios where a node prematurely awakens the FLSC while other Collaborators are still training their models.

```
// Collaborative timeout handling
function reportTimeout() public onlyAuthorized {
    require(fl_state == FL_STATE.LEARNING, "Not in learning state");
    require(block.timestamp >= roundStartTime + roundTimeout,
            "Timeout not reached");
    require(!hasReportedTimeout[msg.sender], "Already reported timeout");

    hasReportedTimeout[msg.sender] = true;
    timeoutReportCount++;

    emit TimeoutReported(msg.sender);

    // Proceed only if enough reports received
    if (timeoutReportCount >= timeoutReportThreshold)
        emit RoundProceeded();
}
```

Listing 2: Sketch of the `reportTimeout()` function.

A likely straightforward solution is to establish the timeout within the FLSC. Collaborators and the Aggregator are enabled to call the `reportTimeout()` function, in order to trigger the FLSC and check if the timeout expired (see Listing 2). Online Collaborators are motivated to report that the timeout has expired, as they will not receive rewards at the end of the training phase without doing so.

The `reportTimeout()` function implements a collaborative timeout reporting mechanism. When invoked by an authorized node, it first verifies that: (i) the FL process is in the learning state, (ii) the round timeout has actually expired, and (iii) the caller has not already reported a timeout for the current round. If these conditions are met, the timeout report is recorded and, once a minimum threshold of unique reports is reached, the contract automatically proceeds to the aggregation phase by emitting the `RoundProceeded()`, to be catched by the Aggregator.

3.5.2. Aggregator failure and election

As previously mentioned, the Aggregator can also experience failures. If the Aggregator goes offline, the FLSC elects a novel Aggregator. The process of determining whether the Aggregator is offline can be carried out in the same manner as the failure detection for Collaborators (via the `reportAggregatorFailure()` function). It is important to note that the responsibility of selecting the new Aggregator lies with the FLSC. Therefore, there is no need to implement distributed leader election algorithms.

In this work, we study three different policies for choosing the novel Aggregator:

- **Round Robin**: the aggregator is elected based on a Round Robin approach, i.e.,

  $$\text{aggregatorId} = \text{aggregatorId} + 1 \mod N$$

  being N the number of participants;
- **Uptime**: referring to uptime reliability studies in networked systems, it can be often assumed that the longer the time since the last failure (uptime), the greater the probability that the node will stay online in the future [58]. Thus, in this case the Aggregator becomes the node with highest uptime value;
- **Most Active**: the rationale is to give more chance to those who have held the role of Aggregator for longer (and have earned more) in the past, assuming they are more reliable. In other words, this is a "rich gets richer" strategy.

These strategies might perform differently, based on the failure rates and on the availability of the involved nodes. We will study their behavior in different scenarios in the next section.

3.5.3. Ensuring data consistency in IPFS uploads

To maintain data consistency even in the presence of intermittent faults during IPFS uploads, our protocol basically implements a two-phase commit mechanism. First, a Collaborator uploads its model weights to IPFS and awaits confirmation. Only after receiving a successful IPFS content identifier (CID) does the Collaborator proceed to register these weights on the FL smart contract. This sequencing ensures that if weights are registered on the blockchain, the corresponding data is guaranteed to be available on IPFS.

Furthermore, our protocol includes an integrity verification step where the smart contract compares the digest of the uploaded data with the one computed locally by the participant. This verification serves two purposes: it ensures the data was not corrupted during upload and prevents malicious participants from registering invalid data. If any failure occurs during the IPFS upload process (e.g., network interruption, node failure), the weights are not registered on the smart contract, and the protocol's timeout mechanisms handle the failure as described in Section 3.5.2. This approach guarantees that the system never enters an inconsistent state where weights are registered but unavailable or corrupted.

3.6. Fault model and recovery mechanisms

Our protocol handles both transient and permanent node failures through a timeout-based mechanism. Here, we delineate our fault model assumptions and recovery procedures.

3.6.1. Fault model

The system operates under the following assumptions for both Collaborators and Aggregators:

- Nodes can experience arbitrary delays or failures;
- Failed nodes may recover at any time;
- Multiple nodes may fail simultaneously;
- Node failures are fail-stop (nodes either work correctly or stop).

3.6.2. Handling delayed and failed nodes

For Collaborators experiencing delays or failures during a training round, the protocol implements a straightforward exclusion mechanism. If a node fails to submit its model weights within the established timeout period, the round proceeds without its contribution. This approach prioritizes system progress over complete participation, acknowledging that in practical FL deployments, perfect availability cannot be guaranteed. In other words, the late arrival of a Collaborator does not cause problems for the continuation of the distributed algorithm, although it obviously reduces the precision of the update, since the contribution of the Collaborator is lost.

When a previously failed node recovers, it can seamlessly rejoin the training process in subsequent rounds without any explicit recovery procedure. This is possible because the set of participating nodes remains fixed throughout the training process and no explicit re-registration is required after recovery.

As concerns the specific duration of the timeouts, it is worth noting that determining the optimal time threshold and frequency of global updates depend on several specific factors. In our view, update frequency can be considered as a hyper-parameter to be tuned based on the requirements of the application and the characteristics of the federated learning system.

3.6.3. Consensus threshold management

The protocol requires a minimum number of nodes to report timeouts or node failures before taking action. This threshold is initially fixed but can be dynamically adjusted by the contract administrator to prevent potential deadlocks in scenarios with frequent failures. For instance, if multiple node failures reduce the number of active participants below the reporting threshold, the administrator can adjust this parameter to maintain system liveness.

3.6.4. Aggregator election during failures

The Aggregator election process follows similar timeout principles. If an Aggregator becomes unresponsive, a new election is triggered after the timeout period expires. If failures occur during the election process itself, the timeout mechanism ensures the system eventually proceeds with available nodes.

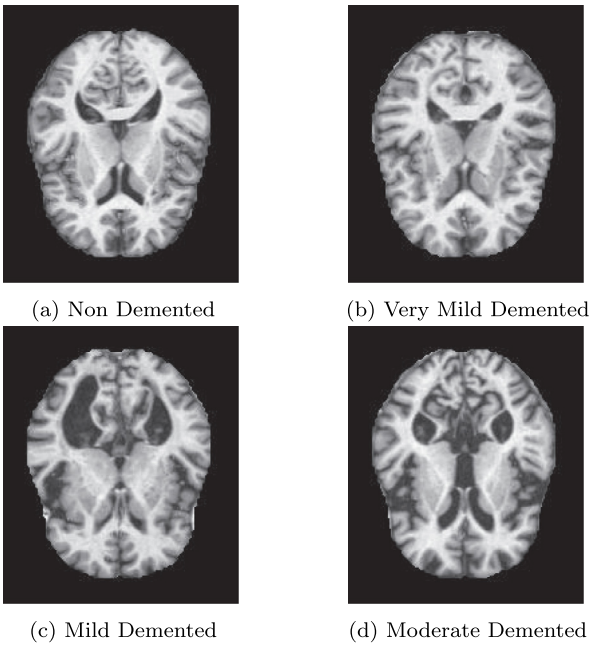
3.6.5. Limitations

It is worth noting that this fault model has some limitations: (i) the consensus threshold remains static until manually adjusted; (ii) multiple simultaneous failures could temporarily block progress if the threshold cannot be met (iii) the recovery from such simultaneous failures of a majority of nodes would require some external intervention.

While this timeout-based approach may introduce delays in the training process, especially if nodes experience frequent failures, it provides a simple mechanism for handling node unavailability. More reliable and accurate algorithms can be plugged and implemented into the protocol, that could cope with more complex and sophisticated failure handling mechanisms.

4. Performance evaluation

Given the complexity of the system, we decided to evaluate different aspects in isolation. In fact, DeSCO-FL involves: (i) FL techniques for the decentralized analysis of datasets, which are partitioned across multiple sources; (ii) blockchain technologies, particularly the FLSC, to coordinate the training phases, (iii) IPFS, used for the decentralized yet secure exchange of information, thanks to data encryption, (iv) mechanisms to cope with node failures that should, on one side, ensure that an Aggregator is elected to avoid that the FL is blocked and, on the other side, provide incentives to participate as Collaborators and act, when needed, as Aggregator.



(a) Non Demented (b) Very Mild Demented

(c) Mild Demented (d) Moderate Demented

Fig. 3. Samples of four image categories of Alzheimer's disease.

4.1. The dataset

The dataset we utilized for testing our FL system architecture is related to image classification in healthcare, which is a typical use-case application for FL. In particular, we used the Alzheimer Dataset, accessible on Kaggle at this URL: https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images. This dataset comprises axial brain anatomy images captured through Magnetic Resonance Imaging (MRI), a non-invasive imaging technique that employs strong magnetic fields and radio waves to generate detailed images of the internal structures of the body, including the brain. These axial MRI sections of the brain are commonly used to visualize and analyze various anatomical structures and pathologies. The dataset contains a total of 6,399 grayscale images, each with dimensions of 208 × 176 pixels. These images represent different stages of Alzheimer's disease and are categorized into four classes: Non Demented, Very Mild Demented, Mild Demented, and Moderate Demented. Fig. 3 shows four samples of images of the four classes.

For the purpose of our analysis, the dataset has been divided into three subsets: a training set, a validation set, and a test set. The training set comprises 70% of the dataset, while the validation and test sets each contain 15% of the images. Where not differently stated, the dataset was uniformly distributed across Collaborators, i.e., each Collaborator had its own unique subset of samples, that was equivalent in number to other ones. This means that the contributions of various Collaborators are comparable. In other words, we do not assume that the data from a participant is more significant or particular than others (this would generate a series of possible policies different from those examined in this study).

4.2. Federated learning performance

Collaborators used the same ML approach for classification, i.e., a three-layered, two-dimensional Convolutional Neural Network (CNN) model, coupled with a final fully connected layer at the end for the final classification of each image. The models are trained with the same setup: an Adam optimizer, a learning rate set to 10e-3, batch size equal to 32, softmax as the last activation function. The classic cross-entropy is used as the loss function.

Table 1
FL performance - number of collaborators = 10.

ML method	Accuracy	F1_score
Centralized	0.99	0.99
FedAvg	0.96	0.98
FedProx	0.98	0.98

Table 2
Classification results for *FedAvg* and *FedProx*. Number of Collaborators = 10. Accuracy: *FedAvg*=0.96, *FedProx*=0.98.

	Precision	Recall	F1-score	Support
NonDem	0.95/0.98	0.98/0.98	0.97/0.98	512
VeryMildDem	0.98/0.96	0.92/0.97	0.95/0.97	359
MildDem	0.99/0.99	0.99/0.99	0.99/0.99	144
ModerateDem	1.00/1.00	1.00/1.00	1.00/1.00	10

As concerns the aggregation algorithm, we used the mentioned *FedAvg* and *FedProx* [17]. In *FedAvg*, we tried with different values for the μ hyper-parameter, finding that after the tuning the best results were obtained with $\mu = 0.001$. Thus, we show results obtained according to this setting.

In this first part of this section, we show some preliminary results on FL schemes, just to provide an initial overview on the measured effectiveness of these approaches. The aim, however, is to focus on the performance of a general system where Collaborators may fail. The interested reader on more performance results on the FL approach, without failures, can refer to [6,12].

4.2.1. FL performance based on the aggregation scheme

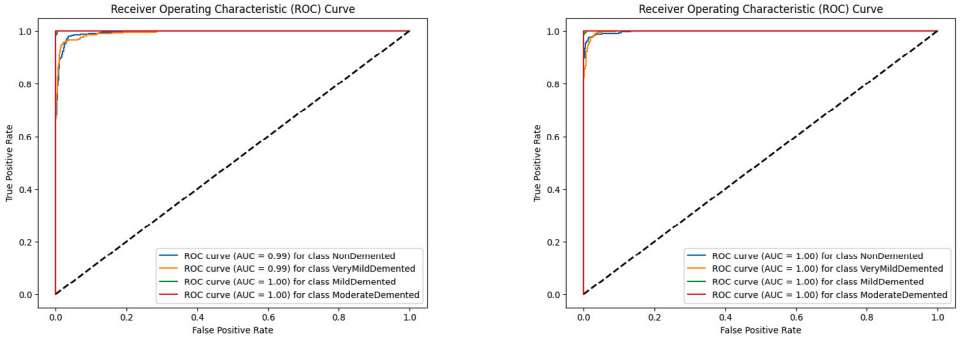
Table 1 shows the averaged accuracy and F1 score obtained by the two FL aggregation techniques, against a centralized approach, i.e., the one that uses a classic ML, where the whole dataset is stored at a central node. In this case, the FL system used 10 Collaborators. Without any surprise, best results are obtained with the centralized approach. Having all the instances in the same data-store eases the training, and this leads to best results. However, we discussed already that in certain scenarios this is not possible. Thus, the centralized approach should be taken as an upper bound for the FL methods. In this respect, it is interesting to observe that the FL schemes perform quite well, especially *FedProx* that is only 0.01 scores below centralized.

Table 2 shows FL results related to each class. In each cell, the first value corresponds to *FedAvg* and the second value corresponds to *FedProx*. While both methods seem to perform well, *FedProx* shows a slight improvement in most categories w.r.t. *FedAvg*. This is confirmed also by the general level of accuracy, which is higher with *FedProx*.

Fig. 4 shows the ROC curves for both aggregation schemes, *FedAvg* (left chart) and *FedProx* (right chart). Each chart shows a different curve for each class of the dataset. The AUC values are reported as well in the legend, for each class. It is possible to appreciate how both schemes are able to obtain very good performance.

4.2.2. FL performance based on the number of collaborators

Fig. 5 shows the accuracy and F1 score of the FL system, for both the aggregation methods, when different numbers of Collaborators are used. We can notice small variations for *FedProx*, while *FedAvg* seems to be affected when the amount of Collaborators increases significantly. This is probably due to the limited size of the dataset. In fact, when we increase the number of Collaborators, the dataset is split into multiple parts and, evidently, with *FedAvg* each Collaborator is not able to improve the overall performance.



(a) *FedAvg* (b) *FedProx*

Fig. 4. ROC curves. Number of Collaborators = 10.

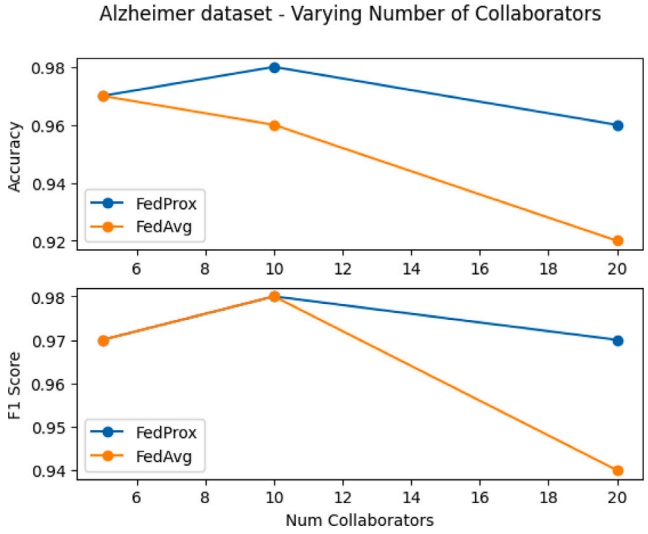


Fig. 5. Accuracy and F1 score performances with a varying number of Collaborators.

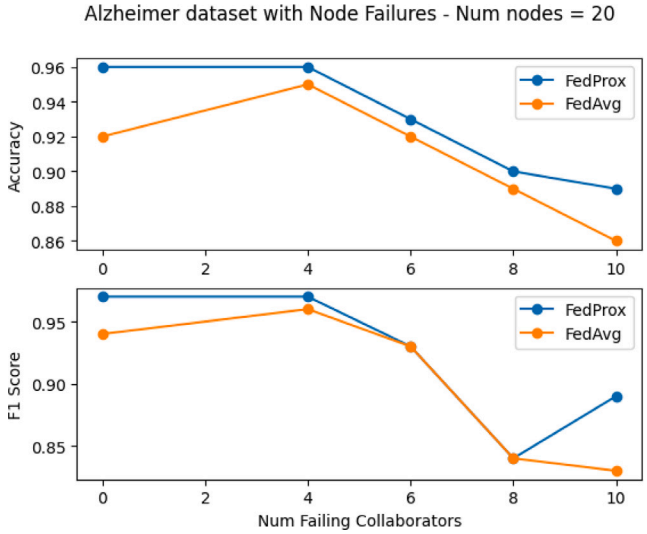


Fig. 6. Accuracy and F1 score performances with node failures.

4.2.3. FL performance with node failures

Fig. 6 reports both the accuracy and F1 score as a function of the number of failing nodes, when the total amount of involved nodes (active and inactive) was equal to 20 Collaborators. Both aggregation methods, *FedProx* and *FedAvg*, are reported. Accuracy and F1 scores of each method change as the number of failing nodes increases. In general, the trend is that, as expected, as the number of failing nodes increases, the performance decreases. This is due to the fact that part of the whole dataset was distributed among nodes (also failed ones) and the loss of the contribution of certain collaborators affects the training. It is worth pointing out that we obtained similar results for other settings with different number of nodes in the system (not shown here for the sake of space limits).

4.3. Federated learning smart contract gas consumption

In this section, we show the gas consumption analysis for the smart contract that governs the interactions in the FL system. Fig. 7 shows gas fees for different non-view methods, as a function of the number of Collaborators involved in the FL system. Each line represents a different method. As expected, there are methods whose Gas consumption is unaffected by the number of participants, i.e., *open(), send_model(), start_learning_phase(), send_aggregated_weights_digests(), send_local_weights_digests()*. In these cases, it is interesting to observe the comparison of Gas consumption with respect to other methods, so as to assess the impact of that specific operation for the whole FL management. The most expensive operation is *send_local_weights_digests()*, while smaller amounts of Gas are required for basic initialization phases such as *open(), start_learning_phase()*. For the sake of a clearer view, in the chart we show mean values only, since we measured very limited standard deviations. This outcome was indeed expected. For instance, as concerns *send_local_weights_digests()*, each Collaborator registers in the smart contract a hash of the weights it just uploaded to IPFS. Thus, it basically sends a fixed length datum. When gas fees increase with the amount of nodes in the system, such increment is linear. This result confirms the viability of the approach, especially in those situations when the number of Collaborators involved in the FL system is limited and does not require continuous updates to the set of participants. In this latter scenario, maybe some optimization techniques might be necessary.

As concerns the different variants of the method *aggregator_offline()*, the Gas required is, in general limited with respect to the other FLSC methods. A linear increment on the Gas consumption can be appreciated, with respect to the number of Collaborators, for the variants "Most Active" and "Uptime", while the "Round Robin" maintains mostly a constant Gas consumption. This is clearly due to the algorithm for the Aggregator selection, i.e., a "Round Robin" policy simply passes the role to the next Collaborator Id, while the other two approaches involve a comparison among the active Collaborators, hence the higher this number the higher the workload.

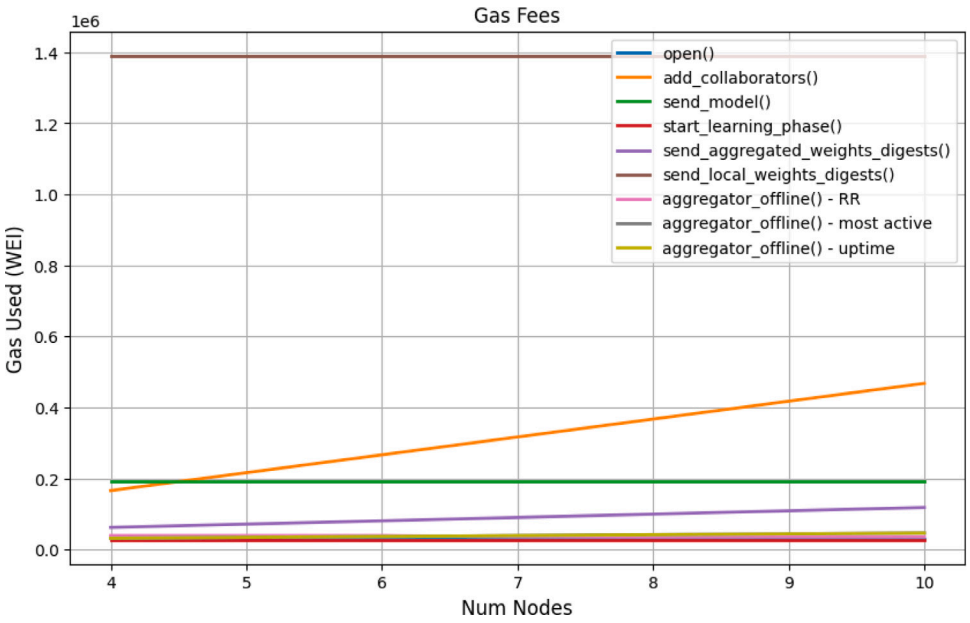


Fig. 7. FLSC: GAS consumption.

4.4. IPFS delays

This section presents our experimental evaluation of IPFS data upload performance using real-world deployments. We compared two different IPFS configurations:

- **IPFS Proprietary**: A dedicated IPFS node deployed on our infrastructure and connected to the main IPFS network, specifically configured to handle our FL system's messages.
- **IPFS Service**: The Infura service [59], which provides free access to the IPFS public network.

Table 3 summarizes our experimental results, comparing both configurations across varying numbers of collaborators [60]. Our evaluation focuses on two key metrics:

- Upload latency: time between message submission to the IPFS node and its response
- Error rate: percentage of failed requests (HTTP status codes 500 or 504)

For each configuration, we report the average latency, standard deviation, and confidence interval. Failed requests were excluded from latency calculations. The experiments simulated different numbers of Collaborators submitting their computed parameters to IPFS at each FL round. Given that model parameters typically constitute a relatively small set of numbers, we fixed the message size to 10KB for all tests.

The two configurations differ significantly in their deployment context. The *IPFS Proprietary* setup uses our dedicated node that exclusively handles FL system messages. In contrast, the *IPFS Service* configuration relies on Infura's public nodes, which concurrently process requests from users worldwide.

Our results show that the *IPFS Proprietary* configuration achieves consistently lower average latencies with zero errors across all collaboration levels. This superior performance can be attributed to its dedicated nature and optimized configuration for FL-specific requests. However, we observed an interesting scalability pattern: the *IPFS Proprietary* setup shows a marked increase in response times as the number of Collaborators grows, suggesting potential scalability challenges with higher workloads.

Conversely, the *IPFS Service* configuration maintains more stable performance across different collaboration levels, albeit with higher average latencies. This stability indicates better robustness and adaptability to varying workloads, likely due to Infura's enterprise-grade infrastructure and load balancing capabilities.

The results presented in Table 3 highlight a significant difference in the standard deviation (Std) between the proprietary scheme and the IPFS service scheme. Specifically, the IPFS service scheme exhibits a much higher standard deviation across all collaborator configurations.

One of the primary reasons for the high standard deviation in the IPFS service scheme is the variability in the testing environment. The tests were conducted on different days, which introduces temporal variability in network conditions and server loads. Such variability can significantly impact the performance metrics, leading to higher standard deviations. Additionally, the Infura's architecture (exploited by the IPFS Service configuration) involves routing requests to different IPFS nodes based on availability and load balancing. This dynamic routing mechanism can result in varying response times, contributing to the observed high standard deviation. Thus, one explanation for this high variability is that the Infura API connects requests to different IPFS nodes, leading to different performance. Further investigation is required to confirm this behavior.

4.5. Impact of node failures

In this study, we conducted simulations to assess the impact of node failures on the FL process. Specifically, we aimed to understand how the roles, incentives, and contributions of each node vary with the probability of failure. To this end, we present results for the considered Aggregator election policies ("Round Robin", "Uptime", "Most Active") in different scenarios where offline probabilities and failure duration are varied. In these experiments, the number of nodes participating to the FL system was equal to 30. We experimented with different numbers of nodes, yielding similar outcomes; thus, we omit these results for the sake of conciseness. The simulator was a time-stepped simulator, written in Python. For each scenario, we executed a corpus of 100 simulations, each one running for 10^4 simulation steps.

During each round, we assume that each active node is able to process a certain amount of data, whose quantity is modeled through a positive half normal distribution. In any case, this value does not impact the selection of the Aggregator, nor the reward for being an active Collaborator.

Table 3
IPFS test results.

Collaborators	Scheme	Avg	Std	Err%	ConfInt
10	Proprietary	0.19	0.08	0.0	(0.18, 0.20)
	Service	9.49	6.18	0.0	(9.09, 9.90)
20	Proprietary	0.20	0.13	0.0	(0.19, 0.20)
	Service	8.48	5.42	5.17	(8.22, 8.74)
30	Proprietary	0.25	0.20	0.0	(0.24, 0.25)
	Service	8.23	7.57	1.02	(7.92, 8.55)
40	Proprietary	0.59	0.77	0.0	(0.57, 0.62)
	Service	7.50	9.10	0.0	(7.18, 7.83)
50	Proprietary	0.57	0.71	0.0	(0.55, 0.59)
	Service	9.86	7.25	0.43	(9.63, 10.10)
60	Proprietary	1.42	1.79	0.0	(1.36, 1.47)
	Service	7.92	6.73	0.0	(7.72, 8.12)
70	Proprietary	2.65	3.32	0.0	(2.56, 2.74)
	Service	6.22	4.68	0.0	(6.09, 6.34)
80	Proprietary	3.64	4.31	0.0	(3.53, 3.75)
	Service	7.85	6.81	0.0	(7.67, 8.02)

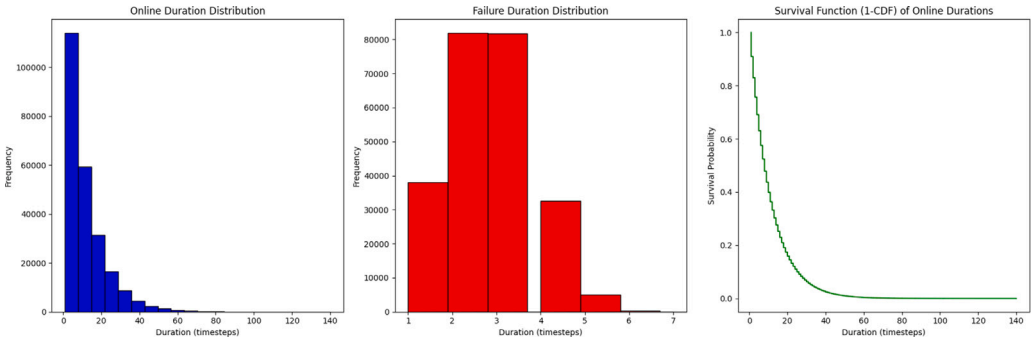


Fig. 8. Online and Offline Distributions with Uniform Offline Probability, Gaussian failure duration.

4.5.1. Single class of nodes uniform offline probability, Gaussian failure duration

In this first scenario, all nodes have the same probability to fail and the same probability of time to recovery. This scenario is plausible in contexts where the Collaborators are fairly equivalent. An example could be a healthcare environment, where each Collaborator represents a computational environment within a hospital, a typical use case for FL systems [6].

Fig. 8 shows the distribution and survival function of online and failure durations. The first two histograms represent the frequency distribution of online and failure durations, respectively. The third plot illustrates the survival function (1-CDF) of online durations, showing the probability of durations exceeding a given time.

Fig. 9 shows a set of charts resulting from the set of experiments. Each graph displays the results for each Aggregator election scheme. Specifically, the following are reported: reward as Aggregator, reward as Collaborator, total reward, the number of steps as Aggregator, and throughput. The results are shown as box-plots. A box-plot is a standardized way of displaying the dataset based on the use of a box (indicating first and third quartiles), a vertical line that identifies the so-called whiskers and a horizontal line indicating the median value.

From the figure, it can be observed that the three schemes show comparable results for throughput and rewards as Collaborator. These results were expected, given that the policies are focused on the election of the Aggregator; hence, according to these schemes the Collaborators behave in the same way. Instead, focusing on the reward as Aggregator and the simulation time (steps) during which the nodes have assumed this Aggregator role, while Round Robin and Uptime have fairly comparable results, with only a wider level of variability for the "Uptime" policy, the results for the "Most Active" policy are quite different. In fact, a much wider box-plot is observed, denoting a wider dispersion among the various participating nodes, with some outliers. This therefore indicates that the policy effectively operates according to the "rich get richer" approach.

In this scenario, Uptime policy has no effect, since all nodes are equivalent in terms of probability of failure and the amount of time they remain online.

To validate the results, we performed the following statistical tests:

- **T-test on log-transformed data:** We log-transformed the data to address potential issues with non-normality, and then applied a two-sample t-test to assess the statistical significance of the differences between the groups. The log-transformation helped to normalize the distributions and satisfy the underlying assumptions of the t-test, allowing us to assess the statistical significance of the differences between the groups in a robust manner;
- **Wilcoxon test:** Given the indications of non-normality in the data, we also applied the non-parametric Wilcoxon signed-rank test, a non-parametric version of the paired t-test;
- **Mann–Whitney U test:** Additionally, we used the Mann–Whitney U test, another non-parametric test, to evaluate the differences between the results obtained from the employed schemes.

The data showed clear signs of non-normality, with a significant presence of outliers. Therefore, the non-parametric tests (Wilcoxon and Mann–Whitney U) were particularly relevant for validating the statistical significance of the observed differences between the groups. We conducted pairwise comparisons between the three experimental schemes shown in Table 4.

The t-test, Wilcoxon test, and Mann–Whitney U test all indicate that there are no statistically significant differences between the Round

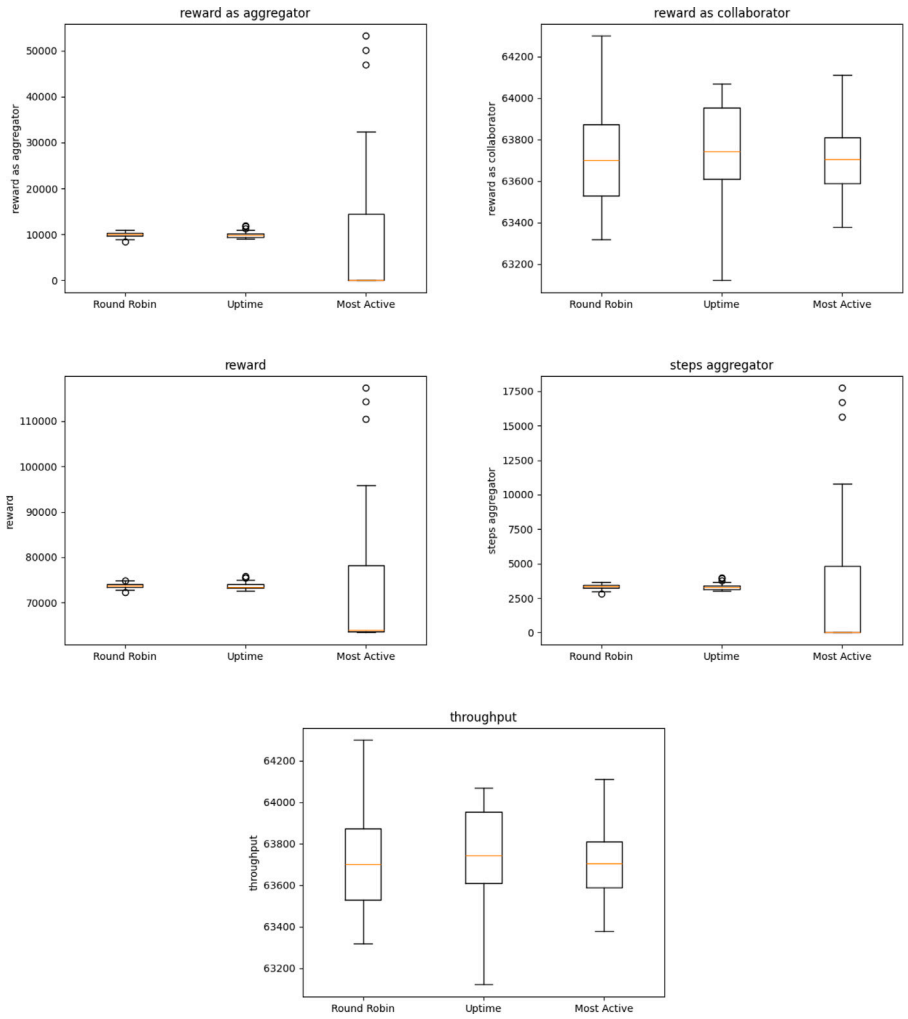


Fig. 9. Results for Single Class with Uniform Offline Probability, Gaussian failure duration.

Robin and Uptime schemes for any of the metrics evaluated (Total Rewards, Throughput, Steps as Aggregator, Aggregator Rewards, Collaborator Rewards).

Instead, the Mann–Whitney U test shows a statistically significant difference in Total Rewards, Steps as Aggregator, Aggregator Rewards between the Round Robin and Most Active schemes. Similarly, a statistically significant difference exists between Round Robin and Most Active.

4.5.2. Three classes with uniform offline probability, Gaussian failure duration

In this scenario, we established three classes of nodes, each with distinct failure probabilities per FL round: 0.1, 0.2, and 0.3. The number of nodes in each class was equal, with 10 nodes per class. Each node that went offline remained so for a number of rounds determined by a Gaussian probability distribution, with the experiments reporting an average offline duration of 3 rounds.

Fig. 10 shows the distribution and survival function of online and failure durations, for the three mentioned classes.

In this case, having different results to be shown for each single class, in Figs. 11–15 we show all the results, for each metrics and for each Aggregator policy. Thus, three charts (one per policy) are shown for each metrics.

From the charts, it is evident that the class of nodes with the lowest probability of failure receives higher rewards and have higher throughput. This is valid for all the three policies. Again, there are no notable differences among the three policies for what concerns the rewards as Collaborators. Instead, significantly different results are obtained for the rewards as Aggregator. In this case, in fact, with the Uptime policy almost only nodes in the class with lower failure probability are selected as Aggregator. The Most Active policy has similar results, but with a wider variability among nodes in this class. Again, this is explained by the rich get richer nature of the policy. Once a particular subset of nodes is chosen more than others, there will always be a preferential choice towards them when they are online. Since in this case, the probability of being online is the same for all the nodes in the first class, on average the first selected nodes will continue to be elected as Aggregator, more often than others.

The statistical analysis of the data in Table 5 shows that, in this scenario, there are no statistically significant differences between the three experimental schemes (Round Robin, Uptime, Most Active) for any of the evaluated metrics.

4.5.3. Single class of nodes Weibull online duration probability, Gaussian failure duration

In this set of experiments, based on [58] we have modeled the duration of time for which a node remains online, using a Weibull probability distribution (shape parameter equal to 2, with a scaling factor of 10). The idea was to simulate an approach more similar to a peer-to-peer system. The duration of time for which a node remains offline is still modeled through a Gaussian distribution, as in the previous case.

Table 4
Uniform - statistical tests comparison.

(a) RR vs UP

Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	−0.269	0.789	0.968	0.492
Throughput	−0.940	0.351	0.299	0.212
Steps as Aggregator	0.069	0.945	0.670	0.311
Aggregator Rewards	0.069	0.945	0.670	0.311
Collaborator Rewards	−0.940	0.351	0.299	0.212

(b) RR vs MA

Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.572	0.572	0.245	**0.008****
Throughput	0.182	0.856	0.855	0.830
Steps as Aggregator	**5.271**	**0.000*****	0.245	**0.008****
Aggregator Rewards	**5.265**	**0.000*****	0.245	**0.008****
Collaborator Rewards	0.182	0.856	0.855	0.830

(c) UP vs MA

Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.590	0.560	0.253	**0.007****
Throughput	1.188	0.240	0.198	0.201
Steps as Aggregator	**5.270**	**0.000*****	0.253	**0.007****
Aggregator Rewards	**5.263**	**0.000*****	0.253	**0.007****
Collaborator Rewards	1.188	0.240	0.198	0.201

RR = Round Robin, UP = Uptime, MA = Most Active
t-p = t-test *p*-value on log-transformed data
Wilc-p = Wilcoxon test p-value
MW-p = Mann–Whitney U test p-value
Significance levels: *** p <0.001, ** p <0.01, * p <0.05 (bold values indicate statistical significance).

Table 5
Three classes - statistical tests comparison.

(a) RR vs UP

Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	−0.004	0.997	0.746	0.923
Throughput	−0.001	0.999	0.792	0.888
Steps as Aggregator	−0.009	0.992	0.584	0.959
Aggregator Rewards	−0.009	0.992	0.584	0.959
Collaborator Rewards	−0.001	0.999	0.792	0.888

(b) RR vs MA

Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.001	0.999	0.952	0.923
Throughput	0.010	0.992	0.746	0.900
Steps as Aggregator	−0.043	0.966	0.808	0.865
Aggregator Rewards	−0.043	0.966	0.808	0.865
Collaborator Rewards	0.010	0.992	0.746	0.900

(c) UP vs MA

Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.005	0.996	0.655	0.994
Throughput	0.012	0.991	0.746	0.853
Steps as Aggregator	−0.034	0.973	0.931	0.819
Aggregator Rewards	−0.034	0.973	0.931	0.819
Collaborator Rewards	0.012	0.991	0.746	0.853

RR = Round Robin, UP = Uptime, MA = Most Active
t-p = t-test *p*-value on log-transformed data
Wilc-p = Wilcoxon test p-value
MW-p = Mann–Whitney U test p-value
Significance levels: *** p <0.001, ** p <0.01, * p <0.05 (bold values indicate statistical significance).

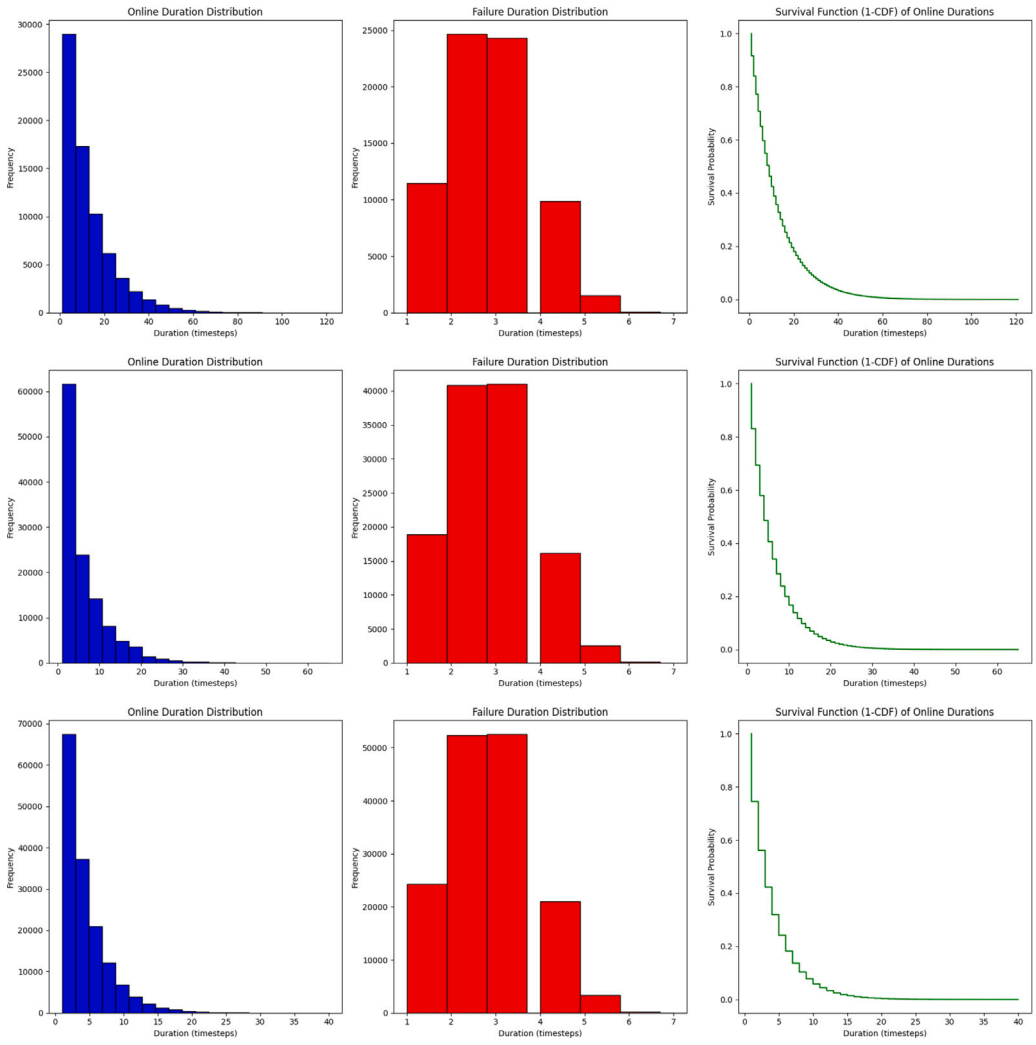


Fig. 10. Online and Offline Distributions with Three classes with Uniform offline Probability, Gaussian failure duration.

Fig. 16 shows the distribution and survival function of online and failure durations.

Results are in line with the previous case, with an even more pronounced "rich get richer" phenomenon for the Most Active policy. We varied the shape parameter of the Weibull distribution, obtaining different results that, however, lead to the same considerations, from a qualitative point of view (see Fig. 17).

The statistical analysis of the results is presented in Table 6. For the Weibull scenario, the t-test, Wilcoxon test, and Mann–Whitney U test did not find any statistically significant differences between the Round Robin and Uptime schemes. However, when comparing Most Active and the other two schemes, the analysis shows some notable differences. In fact, the Mann–Whitney U test indicates a highly significant difference in Total Rewards and the Aggregator-related metrics (Steps as Aggregator, Aggregator Rewards).

4.5.4. Single class of nodes exponential online duration probability, Gaussian failure duration

In these set of experiments, we also modeled the online duration probability using an exponential distribution. The exponential distribution is a powerful tool in modeling the time between events in a Poisson point process, i.e., a well known mathematical model used in performance evaluation for generating random points in time. It is often used to model the lifespan of an object or the time between events, such as the time between node failures in a network. This distribution was chosen due to its memoryless property, which means that the remaining time until the next event does not depend on how much time has passed already [61].

Fig. 18 shows the distribution and survival function of online and failure durations.

Fig. 19 shows results for the considered metrics measured in this scenario. From a qualitative point of view, these charts lead to similar considerations as before. Focusing on metrics related to the Aggregator, i.e., reward as Aggregator and the steps as Aggregator, it is evident once more that Most Active policy elects always the same nodes as Aggregator, when they are online. The Round Robin policy shows the smallest box-plots, meaning that as expected it treats all nodes in the same way. Interestingly, however, some outliers are present testifying the fact that some nodes had more chances to become Aggregator in time. The Uptime policy show a larger box-plot with respect to Round Robin, but without outliers.

Conversely, when we focus on the reward as Collaborator, it is the Uptime policy that shows some few outliers with a smaller box-plot, with respect to the other two policies. However, the general trend is that according to these schemes, Collaborator nodes are treated in the same way.

The statistical analysis of the Exponential scenario data, shown in Table 7, reveals no statistically significant differences between the Round Robin and Uptime schemes, as in previous cases. Again, there is a statistically significant difference with Most Active, confirming the outcomes obtained in other scenarios.

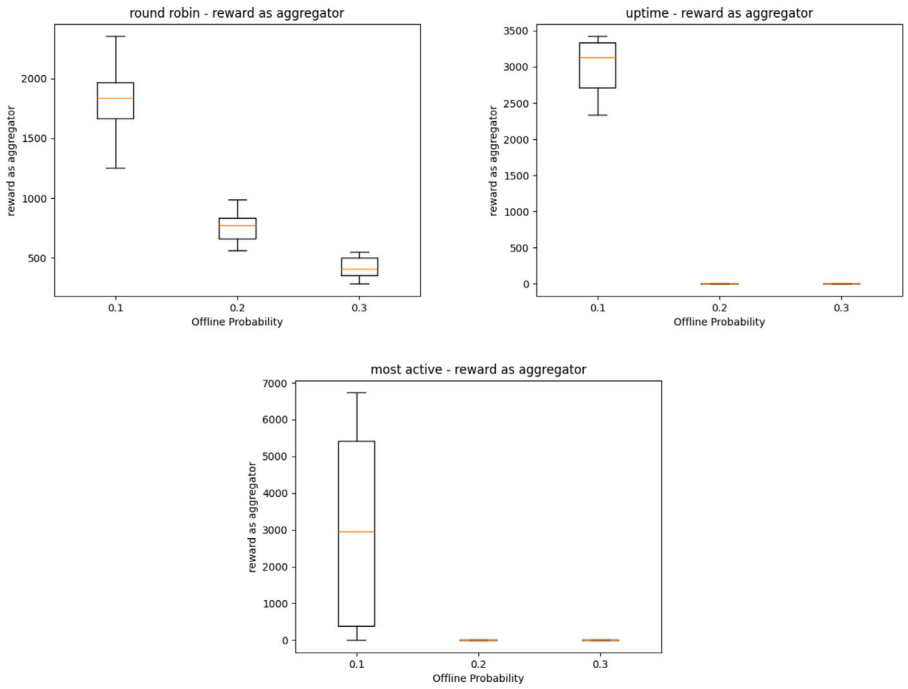


Fig. 11. Results for Three classes with Uniform offline Probability, Gaussian failure duration - Reward as Aggregator.

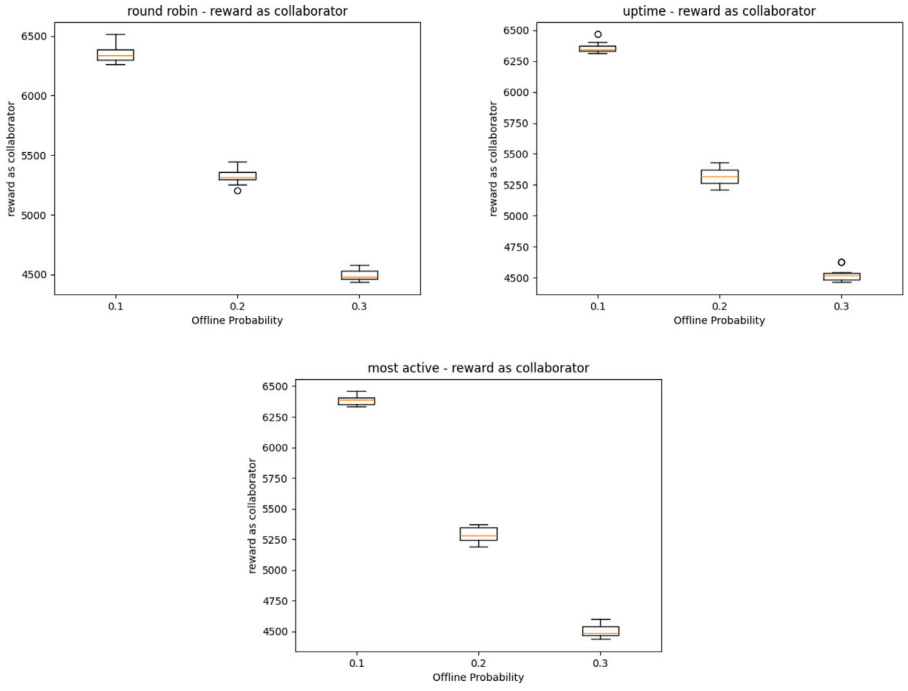


Fig. 12. Results for Three classes with Uniform offline Probability, Gaussian failure duration - Reward as Collaborator.

4.5.5. Summary of key findings across node failure scenarios

Our extensive analysis across different node failure scenarios reveals several consistent patterns in the behavior of the three Aggregator election policies. First, regarding the rewards and behavior of Collaborator nodes, all three policies (Round Robin, Uptime, and Most Active) show comparable results across all scenarios, with minimal variations in throughput and Collaborator rewards. This consistency stems from the fact that these policies primarily focus on Aggregator election rather than Collaborator management.

The most significant differences emerge in the Aggregator-related metrics (rewards and election frequency). The Most Active policy consistently demonstrates a "rich get richer" phenomenon across all scenarios, leading to higher variability in Aggregator rewards and more concentrated Aggregator selections among a smaller subset of nodes. This effect is more evident in the Weibull online duration scenario. The Round Robin policy, by design, shows the most equitable distribution of Aggregator roles and rewards, while the Uptime policy typically falls between these two extremes.

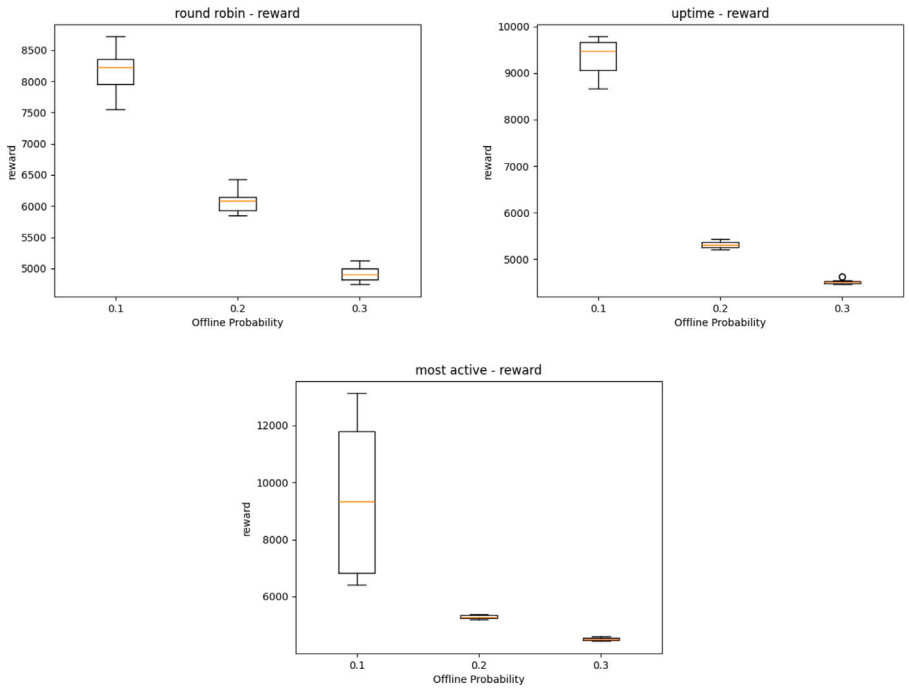


Fig. 13. Results for Three classes with Uniform offline Probability, Gaussian failure duration - Reward.

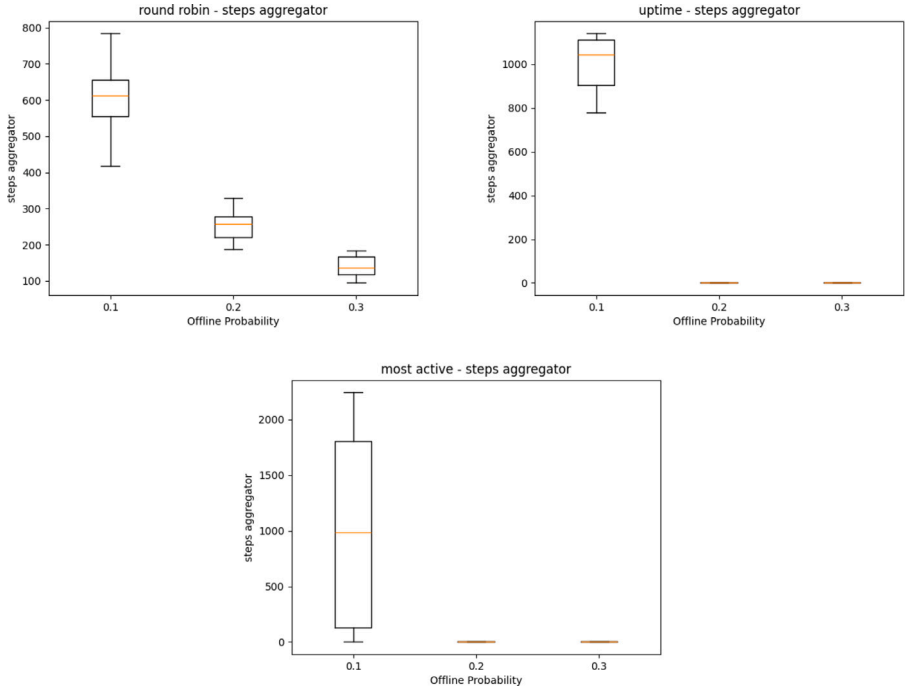


Fig. 14. Results for Three classes with Uniform offline Probability, Gaussian failure duration - Number of Steps as Aggregator.

In scenarios with heterogeneous node classes (varying failure probabilities), nodes with lower failure probabilities receive higher rewards and achieve better throughput, regardless of the election policy. However, this effect is amplified in the Uptime and Most Active policies, where more reliable nodes are significantly more likely to be selected as Aggregators.

These findings suggest that while all policies maintain similar system-wide performance metrics, they differ substantially in how they distribute Aggregator responsibilities and rewards, with important implications for system fairness and node participation incentives.

4.5.6. Time to completion

An additional noteworthy result pertains to the "time to completion", i.e., the time required to conduct a certain number of training rounds for FL. This metric aims to evaluate whether the choice among the three studied policies is superior to the others in terms of time efficiency. Fig. 20 displays the results obtained for the three policies, using a node configuration identical to the previous one (30 nodes), but in this case, with a Uniform and equal probability of failure for all nodes. Each curve represents the average obtained for a certain policy

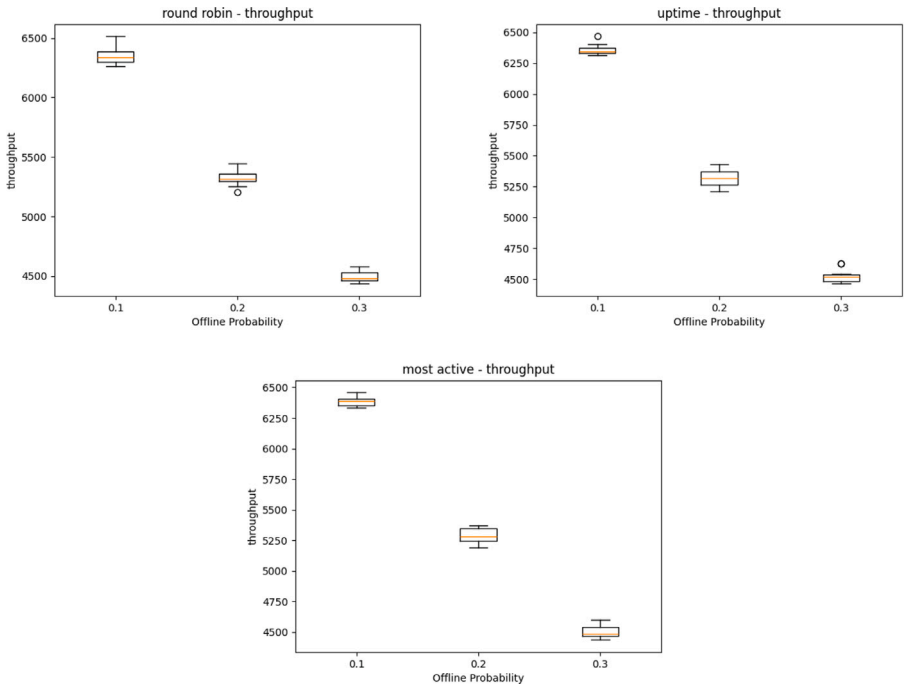


Fig. 15. Results for Three classes with Uniform offline Probability, Gaussian failure duration - Throughput.

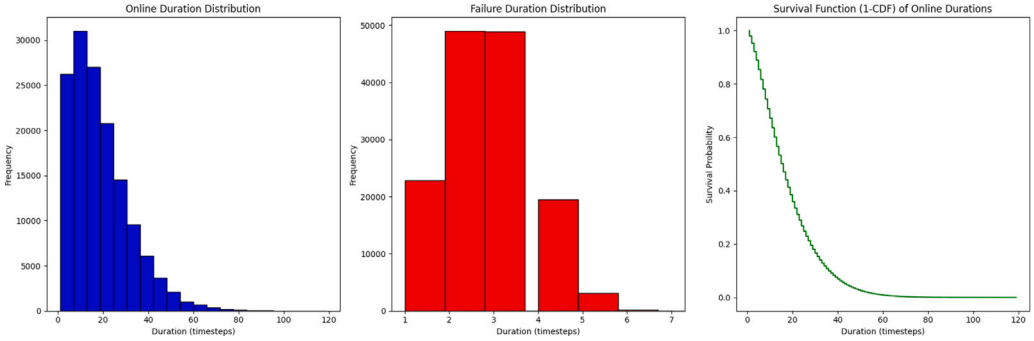


Fig. 16. Online and Offline Distributions with Weibull Online Duration Probability, Gaussian failure duration.

by repeating the same simulated scenario. The shaded areas represent the obtained standard deviation.

It is noticeable that in this case, the Round Robin policy proved to be the best, as it generally takes less time to complete the work in terms of simulation timesteps. The Most Active policy has similar but slightly worse results, while the Uptime policy is decidedly worse than the others. This outcome is, however, quite easily explainable. In this simulated scenario, the nodes have, on average, the same probability of failure. Choosing the node that has been active the longest does not offer advantages in this case; rather, it is detrimental, as this policy is based on the assumption (not respected in this case) that a node that has been very active will continue to be so in the future.

Fig. 21 shows the same type of results of the previous figure, but in the scenario where uptime probabilities were modeled through a Weibull distribution, as a function of the shape parameter. The shape parameter influences the shape of the distribution, in a way such as increasing its value indicates that the failure rate increases with time. When the shape value is below 1, the failure rate decreases over time. This means that there is a sort of significant "infant mortality" and the failure rate decreases over time. A shape value equal to 1 indicates that the failure rate is constant over time, i.e., the Weibull distribution reduces to an exponential distribution. (Thus, this chart comprises also the exponential distribution scenario.) A shape parameter higher than 1 corresponds to a sort of "aging" process, with nodes that are more likely to fail as time goes on.

It is interesting to observe how the time to completion increases, in general. It is confirmed that Uptime and Most Active require higher simulation timesteps. The time increases with the value of shape, but this was expected, as the higher the shape the higher the probability of failures, that correspond to changes in the configuration of the FL system.

Overall, the results suggest that a simple approach such as a Round Robin Aggregation election performs well in general, as it is able to carry on the FL system without introducing overload to specific participants. The devised scheme appears to be able to respond to node failures, by electing a novel Aggregator upon necessity, thus making the FL process resilient.

5. Conclusions

In this paper, we have introduced DeSCo-FL, a FL system that makes use of smart contracts and blockchain technologies, with the objective of achieving a secure approach to managing data and perform ML in a distributed system that engages multiple data sources. The

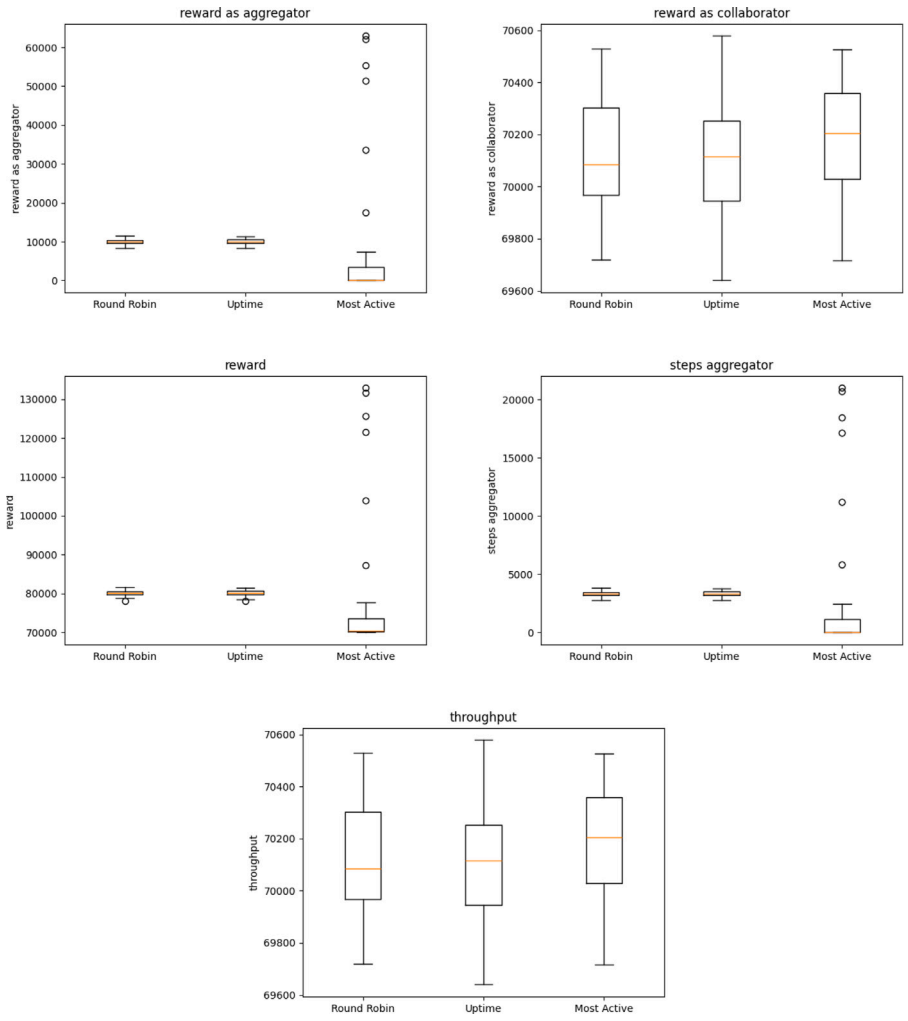


Fig. 17. Results for Single Class with Weibull Online Duration Probability, Gaussian failure duration.

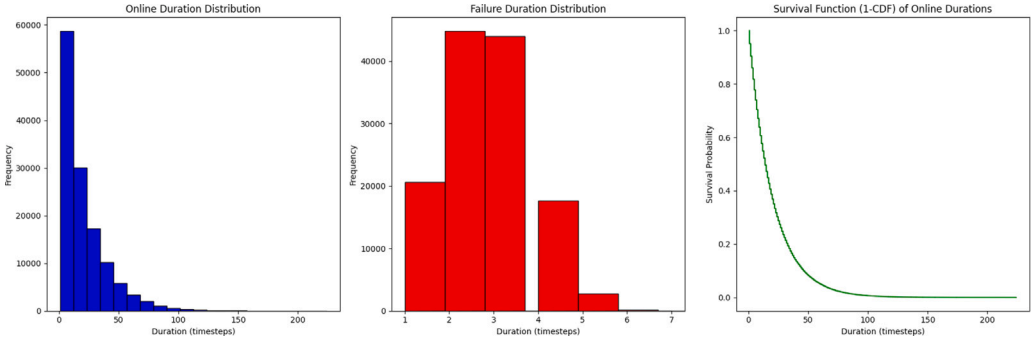


Fig. 18. Online and Offline Distributions with exponential Online Duration Probability, Gaussian failure duration.

architecture of this framework is designed to safeguard data privacy, as it facilitates the exchange of encrypted model parameters, thereby eliminating the need to handle sensitive data directly. Our system incorporates the Inter-Planetary File System (IPFS) and smart contracts, establishing a secure and immutable data storage infrastructure that significantly increases data security during FL processes. Our scheme is mainly based on the presence of the FL smart contract that is in charge of coordinating all the interactions among FL node participants, as well as handling node failures and selecting the Aggregator node. The implementation of the smart contract on a decentralized, permissioned blockchain offers assurances of reliability, thereby obviating the need for resource-intensive distributed algorithms that require extensive message exchange. The effectiveness and feasibility of our proposed system have been demonstrated through an experimental assessment involving different aspects of the FL system.

There are numerous possible future works to this study. From a ML point of view, it is possible to imagine a series of improvements concerned with the sophistication of ML models. Another possibility might be the adoption of some Low-Rank Adaptation (LoRA) techniques applied to FL [62], that could decrease the cost of the training and the

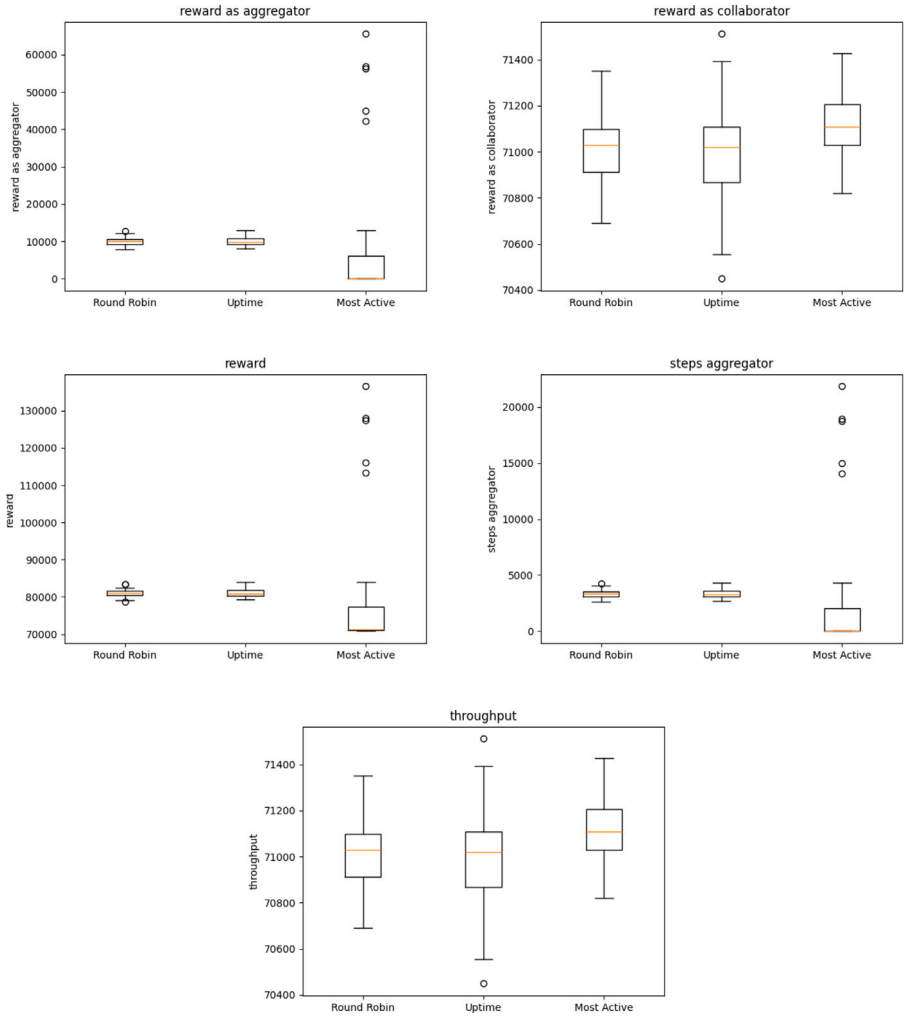


Fig. 19. Results for Single Class with exponential Online Duration Probability, Gaussian failure duration.

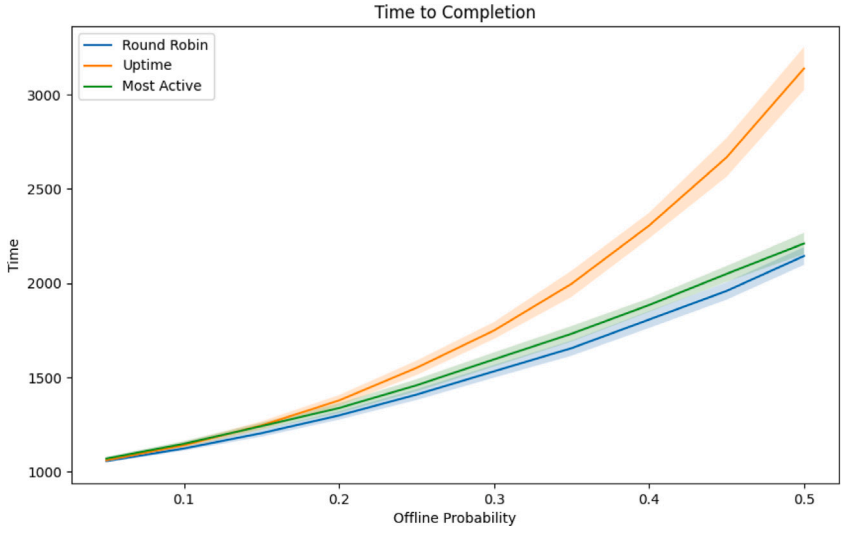


Fig. 20. Simulated time to completion for the three Aggregator Election approaches. Single class with Uniform offline Probability, Gaussian failure duration.

Table 6
Weibull - statistical tests comparison.

(a) RR vs UP				
Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	−0.135	0.893	0.887	0.807
Throughput	−0.470	0.640	0.612	0.673
Steps as Aggregator	0.006	0.995	0.952	0.859
Aggregator Rewards	0.006	0.995	0.952	0.859
Collaborator Rewards	−0.470	0.640	0.612	0.673

(b) RR vs MA				
Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.600	0.553	0.070	**0.000*****
Throughput	−1.780	0.080	0.092	0.115
Steps as Aggregator	**6.221**	**0.000*****	0.070	**0.000*****
Aggregator Rewards	**6.172**	**0.000*****	0.070	**0.000*****
Collaborator Rewards	−1.780	0.080	0.092	0.115

(c) UP vs MA				
Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.608	0.548	0.067	**0.000*****
Throughput	−1.257	0.214	0.237	0.234
Steps as Aggregator	**6.221**	**0.000*****	0.070	**0.000*****
Aggregator Rewards	**6.172**	**0.000*****	0.070	**0.000*****
Collaborator Rewards	−1.257	0.214	0.237	0.234

RR = Round Robin, UP = Uptime, MA = Most Active
t-p = t-test *p*-value on log-transformed data
Wilc-p = Wilcoxon test p-value
MW-p = Mann–Whitney U test p-value
Significance levels: *** p <0.001, ** p <0.01, * p <0.05 (bold values indicate statistical significance).

Table 7
Exponential - statistical tests comparison.

(a) RR vs UP				
Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.081	0.936	0.919	0.762
Throughput	0.457	0.649	0.839	0.695
Steps as Aggregator	−0.018	0.986	0.871	0.923
Aggregator Rewards	−0.018	0.986	0.871	0.923
Collaborator Rewards	0.457	0.649	0.839	0.695

(b) RR vs MA				
Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.588	0.561	0.064	**0.000*****
Throughput	**−2.263**	**0.027***	0.055	**0.028***
Steps as Aggregator	**6.409**	**0.000*****	0.064	**0.000*****
Aggregator Rewards	**6.425**	**0.000*****	0.064	**0.000*****
Collaborator Rewards	**−2.263**	**0.027***	0.055	**0.028***

(c) UP vs MA				
Metric	t-stat	t-p	Wilc-p	MW-p
Total Rewards	0.581	0.566	0.067	**0.000*****
Throughput	**−2.313**	**0.025***	**0.031***	**0.020***
Steps as Aggregator	**6.410**	**0.000*****	0.067	**0.000*****
Aggregator Rewards	**6.426**	**0.000*****	0.067	**0.000*****
Collaborator Rewards	**−2.313**	**0.025***	**0.031***	**0.020***

RR = Round Robin, UP = Uptime, MA = Most Active
t-p = t-test *p*-value on log-transformed data
Wilc-p = Wilcoxon test p-value
MW-p = Mann–Whitney U test p-value
Significance levels: *** p <0.001, ** p <0.01, * p <0.05 (bold values indicate statistical significance).

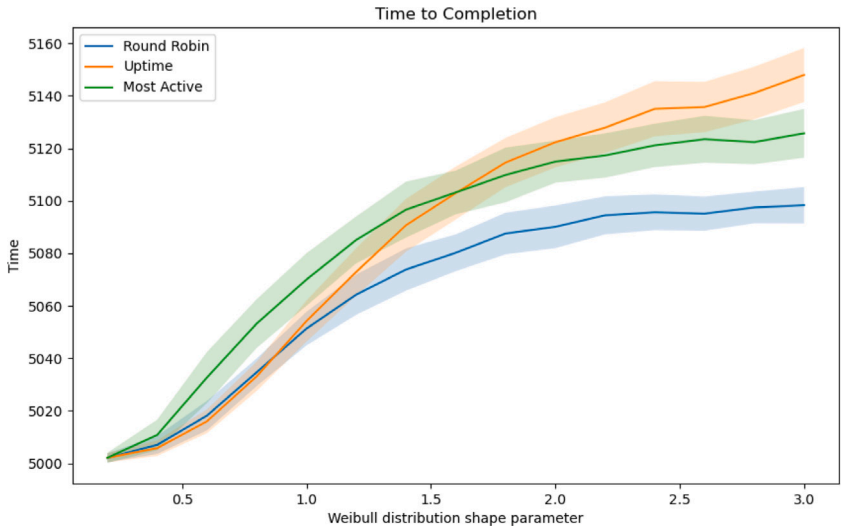


Fig. 21. Simulated time to completion for the three Aggregator Election approaches. Single class with Weibull uptime probability, Gaussian failure duration.

number of employed parameters. This could represent an advantage, since a reduced set of weights would reduce the time required for hash computation before sending the digests to the smart contract, and data retrieval and transmission over IPFS.

Other variations that could also impact the performance of the decentralized system might be related to the use of multi-party computation techniques or split learning [63]. In split learning, different parts of a machine learning model are executed on different nodes of an edge computing system. Besides changing the way ML is done, this has implications on the traceability of the process and accountability of the involved nodes. Therefore, this approach would necessitate a review of the data exchange management and the role of smart contracts.

As concerns the decentralization of the FL system, it would be possible to add more sophistication to the rewarding strategy, so that Collaborators get rewarded for their contribution, provided this contribution can be somehow trusted. This would require some auditing scheme to maintain Collaborators' reputations [64]. Such reputation might be stored in a smart contract based service. Such a kind of approach would most likely require the use of verifiers in charge to check the veracity of the produced data (and model parameters). Moreover, the decentralized management of the system could be based on data governance mechanisms in non-trusted environments that make use of appropriate authorization mechanisms based on multi-party computation [3,65].

While our current implementation demonstrates linear gas cost scaling with the number of participating nodes, several promising optimization techniques could be explored in future work to enhance system scalability. These optimizations could significantly reduce gas costs and improve efficiency for large-scale deployments. An example worth of investigation relates to the use of Merkle tree data structures, that could optimize the storage and verification of model updates, potentially reducing gas costs for larger networks.

One other potential area for improvement in this work is the development of a formal proof that demonstrates the proposed protocol maintains the necessary properties for the proper functioning of a dynamic federated learning system, even under certain fault model assumptions.

Finally, an interesting research topic which could open up for new research synergies within the FL field relates to the verifiable computation approaches, such as incrementally verifiable computation for proof of learning [66]. Verifiable computation enables a node to outsource the evaluation of a function to an untrusted server while maintaining the ability to verify the correctness of the returned result. Incremental verification allows for progressive verification of computation steps, making it particularly suitable for iterative processes like ML training [67]. In fact, this paradigm becomes particularly relevant for ensuring the integrity of training procedures and model inference. Exploring how these techniques can be employed in a dynamic FL scenario seems thus relevant.

CRediT authorship contribution statement

Stefano Ferretti: Writing – original draft, Supervision, Software, Project administration, Funding acquisition, Conceptualization. **Lorenzo Cassano:** Software. **Gabriele Cialone:** Software. **Jacopo D'Abramo:** Software. **Filippo Imboccioli:** Software.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Stefano Ferretti reports financial support was provided by European Union - NextGenerationEU. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is partially supported by the European Union - NextGenerationEU within the framework of PNRR Mission 4 - Component 2 - Investment 1.1 under the Italian Ministry of University and Research (MUR) programme "PRIN 2022" - grant number 2022N2NH42 SmartShires – CUP: H53D23003570006.

Data availability

Link to data and code is provided in the paper.

References

[1] T. Brisimi, et al., Federated learning of predictive models from federated Electronic Health Records, Int. J. Med. Inform. 112 (2018) 59–67, http://dx.doi.org/10.1016/j.ijmedinf.2018.01.007.

[2] O.A. Wahab, A. Mourad, H. Otrok, T. Taleb, Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems, IEEE Commun. Surv. Tutor. 23 (2) (2021) 1342–1397, http://dx.doi.org/10.1109/COMST.2021.3058573.

[3] M. Zichichi, S. Ferretti, G. D'Angelo, V. Rodríguez-Doncel, Data governance through a multi-DLT architecture in view of the GDPR, Clust. Comput. (2022) 1–32.

[4] S. Agrawal, S. Sarkar, O. Aouedi, G. Yenduri, K. Piamrat, M. Alazab, S. Bhattacharya, P.K.R. Maddikunta, T.R. Gadekallu, Federated learning for intrusion detection system: Concepts, challenges and future directions, Comput. Commun. 195 (2022) 346–361, http://dx.doi.org/10.1016/j.comcom.2022.09.012, URL https://www.sciencedirect.com/science/article/pii/S0140366422003516.

[5] M. Gupta, M. Kumar, Y. Gupta, A blockchain-empowered federated learning-based framework for data privacy in lung disease detection system, Comput. Hum. Behav. 158 (2024) 108302, http://dx.doi.org/10.1016/j.chb.2024.108302, URL https://www.sciencedirect.com/science/article/pii/S0747563224001705.

[6] F. Imboccioli, G. Cialone, S. Ferretti, Decentralization of learning and trust in healthcare: Blockchain-driven federated learning for alzheimer's MRI image classification, in: Proc. of the IEEE PerCom Workshops, IEEE, Biarritz, France, 2024.

[7] M. Aledhari, R. Razzak, R.M. Parizi, F. Saeed, Federated learning: A survey on enabling technologies, protocols, and applications, IEEE Access 8 (2020) 140699–140725, http://dx.doi.org/10.1109/ACCESS.2020.3013541.

[8] S. Montagna, M.F. Pengo, S. Ferretti, C. Borghi, C. Ferri, G. Grassi, M.L. Muiesan, G. Parati, Machine learning in hypertension detection: A study on world hypertension day data, J. Med. Syst. 47 (1) (2023) 1, http://dx.doi.org/10.1007/S10916-022-01900-5.

[9] J. Lee, J. Sun, F. Wang, S. Wang, C.-H. Jun, X. Jiang, Privacy-preserving patient similarity learning in a federated environment: Development and analysis, JMIR Med Inf. 6 (2018) http://dx.doi.org/10.2196/medinform.7744.

[10] V. Mothukuri, R.M. Parizi, S. Pouriyeh, A. Dehghantanha, K.-K.R. Choo, FabricFL: Blockchain-in-the-loop federated learning for trusted decentralized systems, IEEE Syst. J. 16 (3) (2022) 3711–3722, http://dx.doi.org/10.1109/JSYST.2021.3124513.

[11] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, Y. Psaras, Design and evaluation of IPFS: a storage layer for the decentralized web, in: Proceedings of the ACM SIGCOMM 2022 Conference, SIGCOMM '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 739–752, http://dx.doi.org/10.1145/3544216.3544232.

[12] L. Cassano, J. D'Abramo, S. Munir, S. Ferretti, Trust and resilience in federated learning through smart contracts enabled decentralized systems, in: Proc. of the 7th IEEE International Conference on Blockchain (Blockchain 2024), IEEE, Copenhagen, Denmark, 2024.

[13] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, W. Zhang, A survey on federated learning: challenges and applications, Int. J. Mach. Learn. Cybern. 14 (2) (2023) 513–535, http://dx.doi.org/10.1007/s13042-022-01647-y.

[14] K.K. Coelho, M. Nogueira, A.B. Vieira, E.F. Silva, J.A.M. Nacif, A survey on federated learning for security and privacy in healthcare applications, Comput. Commun. 207 (2023) 113–127, http://dx.doi.org/10.1016/j.comcom.2023.05.012.

[15] R.S. Antunes, C. André da Costa, A. Küderle, I.A. Yari, B. Eskofier, Federated learning for healthcare: Systematic review and architecture proposal, ACM Trans. Intell. Syst. Technol. 13 (2022) http://dx.doi.org/10.1145/3501813.

[16] Y. Shanmugarasa, H.-y. Paik, S.S. Kanhere, L. Zhu, A systematic review of federated learning from clients' perspective: challenges and solutions, Artif. Intell. Rev. 56 (2) (2023) 1773–1827, http://dx.doi.org/10.1007/s10462-023-10563-8.

[17] T. Li, et al., Federated optimization in heterogeneous networks, in: Proc. of Machine Learning and Systems 2020, MLSys 2020, March 2020, mlsys.org, 2020, URL https://proceedings.mlsys.org/book/316.pdf.

[18] M. Rossini, M. Zichichi, S. Ferretti, On the use of deep neural networks for security vulnerabilities detection in smart contracts, in: Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops, IEEE, 2023, pp. 74–79.

[19] T. Duong, K.K. Todi, U. Chaudhary, H.-L. Truong, Decentralizing air traffic flow management with blockchain-based reinforcement learning, in: INDIN 2019, Vol. 1, 2019, pp. 1795–1800, http://dx.doi.org/10.1109/INDIN41052.2019.8972225.

[20] T.-T. Davarakis, G. Palaiokrassas, A. Litke, T. Varvarigou, Reinforcement learning with smart contracts on blockchains, Future Gener. Comput. Syst. 148 (2023) 550–563, http://dx.doi.org/10.1016/j.future.2023.06.018.

[21] Z. Zhou, C. Guo, X. Zhang, R. Wang, L. Zhang, M. Imran, A blockchain-based data sharing marketplace with a federated learning use case, in: 2023 IEEE International Conference on Blockchain and Cryptocurrency, ICBC, 2023, pp. 1041–1044, http://dx.doi.org/10.1109/ICBC56567.2023.10174981.

[22] L. Meng, Y. Guo, S. Song, 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, http://dx.doi.org/10.1109/ICEACE60673.2023.10442645.

[23] R.N. Alief, M.A. Paramartha Putra, A. Gohil, J.-M. Lee, D.-S. Kim, FLB2: Layer 2 blockchain implementation scheme on federated learning technique, in: 2023 International Conference on Artificial Intelligence in Information and Communication, ICAIIC, 2023, pp. 846–850, http://dx.doi.org/10.1109/ICAIIC57133.2023.10067038.

[24] M. Aloqaily, I. Al Ridhawi, F. Karray, M. Guizani, Towards blockchain-based hierarchical federated learning for cyber-physical systems, in: 2022 International Balkan Conference on Communications and Networking, BalkanCom, 2022, pp. 46–50, http://dx.doi.org/10.1109/BalkanCom55633.2022.9900546.

[25] Y. Deng, T. Han, N. Zhang, FLeX: Trading edge computing resources for federated learning via blockchain, in: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, 2021, pp. 1–2, http://dx.doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484628.

[26] H. Kim, J. Park, M. Bennis, S.-L. Kim, Blockchained on-device federated learning, IEEE Commun. Lett. 24 (6) (2020) 1279–1283, http://dx.doi.org/10.1109/LCOMM.2019.2921755.

[27] X. Deng, J. Li, C. Ma, K. Wei, L. Shi, M. Ding, W. Chen, H.V. Poor, Blockchain assisted federated learning over wireless channels: Dynamic resource allocation and client scheduling, Trans. Wirel. Comm. 22 (5) (2023) 3537–3553, http://dx.doi.org/10.1109/TWC.2022.3219501.

[28] E. Bandara, X. Liang, S. Shetty, R. Mukkamala, A. Rahman, N.W. Keong, Skunk — A blockchain and zero trust security enabled federated learning platform for 5G/6G network slicing, in: 2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON, 2022, pp. 109–117, http://dx.doi.org/10.1109/SECON55815.2022.9918536.

[29] Y. Lu, X. Huang, K. Zhang, S. Maharjan, Y. Zhang, Low-latency federated learning and blockchain for edge association in digital twin empowered 6G networks, IEEE Trans. Ind. Inform. 17 (7) (2021) 5098–5107, http://dx.doi.org/10.1109/TII.2020.3017668.

[30] Y. Lu, X. Huang, K. Zhang, S. Maharjan, Y. Zhang, Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles, IEEE Trans. Veh. Technol. 69 (4) (2020) 4298–4311, http://dx.doi.org/10.1109/TVT.2020.2973651.

[31] Y. Wang, Z. Su, N. Zhang, A. Benslimane, Learning in the air: Secure federated learning for UAV-assisted crowdsensing, IEEE Trans. Netw. Sci. Eng. 8 (2) (2021) 1055–1069, http://dx.doi.org/10.1109/TNSE.2020.3014385.

[32] Y. Zhao, J. Zhao, L. Jiang, R. Tan, D. Niyato, Z. Li, L. Lyu, Y. Liu, Privacy-preserving blockchain-based federated learning for IoT devices, IEEE Internet Things J. 8 (3) (2021) 1817–1829, http://dx.doi.org/10.1109/JIOT.2020.3017377.

[33] S. Salim, B. Turnbull, N. Moustafa, A blockchain-enabled explainable federated learning for securing internet-of-things-based social media 3.0 networks, IEEE Trans. Comput. Soc. Syst. (2021) 1–17, http://dx.doi.org/10.1109/TCSS.2021.3134463.

[34] S. Sai, V. Hassija, V. Chamola, M. Guizani, Federated learning and NFT-based privacy-preserving medical-data-sharing scheme for intelligent diagnosis in smart healthcare, IEEE Internet Things J. 11 (4) (2024) 5568–5577, http://dx.doi.org/10.1109/JIOT.2023.3308991.

[35] A. Ruggeri, R. Di Salvo, M. Fazio, A. Celesti, M. Villari, Blockchain-based strategy to avoid fake AI in ehealth scenarios with reinforcement learning, in: ISCC 2021, 2021, pp. 1–7, http://dx.doi.org/10.1109/ISCC53001.2021.9631523.

[36] X. Huang, X. Deng, Q. Chen, J. Zhang, Aflchain: Blockchain-enabled asynchronous federated learning in edge computing network, in: 2023 IEEE 97th Vehicular Technology Conference, VTC2023-Spring, 2023, pp. 1–5, http://dx.doi.org/10.1109/VTC2023-Spring57618.2023.10199280.

[37] F. Yang, M.Z. Abedin, P. Hajek, An explainable federated learning and blockchain-based secure credit modeling method, European J. Oper. Res. 317 (2) (2024) 449–467, http://dx.doi.org/10.1016/j.ejor.2023.08.040, URL https://www.sciencedirect.com/science/article/pii/S0377221723006677.

[38] R. Fotohi, F. Shams Aliee, B. Farahani, Decentralized and robust privacy-preserving model using blockchain-enabled federated deep learning in intelligent enterprises, Appl. Soft Comput. 161 (2024) 111764, http://dx.doi.org/10.1016/j.asoc.2024.111764, URL https://www.sciencedirect.com/science/article/pii/S1568494624005386.

[39] R. Kumar, C.M. Bernard, A. Ullah, R.U. Khan, J. Kumar, D.K. Kulevome, R. Yunbo, S. Zeng, Privacy-preserving blockchain-based federated learning for brain tumor segmentation, Comput. Biol. Med. 177 (2024) 108646, http://dx.doi.org/10.1016/j.compbiomed.2024.108646, URL https://www.sciencedirect.com/science/article/pii/S0010482524007315.

[40] Y. Liu, Z. Jia, Z. Jiang, X. Lin, J. Liu, Q. Wu, W. Susilo, BFL-SA: Blockchain-based federated learning via enhanced secure aggregation, J. Syst. Archit. 152 (2024) 103163, http://dx.doi.org/10.1016/j.sysarc.2024.103163, URL https://www.sciencedirect.com/science/article/pii/S1383762124001000.

[41] R.U. Haque, A. Touhidul Hasan, M.A.M. Al-Hababi, Y. Zhang, D. Xu, SSI-FL: Self-sovereign identity based privacy-preserving federated learning, J. Parallel Distrib. Comput. 191 (2024) 104907, http://dx.doi.org/10.1016/j.jpdc.2024.104907.

[42] D.C. Nguyen, M. Ding, Q.-V. Pham, P.N. Pathirana, L.B. Le, A. Seneviratne, J. Li, D. Niyato, H.V. Poor, Federated learning meets blockchain in edge computing: Opportunities and challenges, IEEE Internet Things J. 8 (16) (2021) 12806–12825, http://dx.doi.org/10.1109/JIOT.2021.3072611.

[43] Z. Lei, K. Gai, J. Yu, S. Wang, L. Zhu, K.-K. Raymond Choo, Efficiency-enhanced blockchain-based client selection in heterogeneous federated learning, in: 2023 IEEE International Conference on Blockchain, Blockchain, 2023, pp. 289–296, http://dx.doi.org/10.1109/Blockchain60715.2023.00053.

[44] K. Pratim Kalita, D. Boro, D. Kumar Bhattacharyya, An efficient consensus algorithm for blockchain-based federated learning, in: 2023 International Conference on Intelligent Systems, Advanced Computing and Communication, ISACC, 2023, pp. 1–7, http://dx.doi.org/10.1109/ISACC56298.2023.10083761.

[45] M.R. Behera, S. Upadhyay, S. Shetty, Federated learning using smart contracts on blockchains, based on reward driven approach, 2021, ArXiv abs/2107.10243.

[46] Y. Qu, M.P. Uddin, C. Gan, Y. Xiang, L. Gao, J. Yearwood, Blockchain-enabled federated learning: A survey, ACM Comput. Surv. 55 (4) (2022) http://dx.doi.org/10.1145/3524104.

[47] X. Bao, C. Su, Y. Xiong, W. Huang, Y. Hu, FLChain: A blockchain for auditable federated learning with trust and incentive, in: BIGCOM, 2019, pp. 151–159, http://dx.doi.org/10.1109/BIGCOM.2019.00030.

[48] Y. Li, C. Chen, N. Liu, H. Huang, Z. Zheng, Q. Yan, A blockchain-based decentralized federated learning framework with committee consensus, IEEE Netw. 35 (1) (2021) 234–241, http://dx.doi.org/10.1109/MNET.011.2000263.

[49] V. Huang, S. Sohail, M. Mayo, T.L. Botran, M. Rodrigues, C. Anderson, M. Ooi, Keep It Simple: Fault Tolerance Evaluation of Federated Learning with Unreliable Clients , in: 2023 IEEE 16th International Conference on Cloud Computing, CLOUD, IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 1–3, http://dx.doi.org/10.1109/CLOUD60044.2023.00024, URL https://doi.ieeecomputersociety.org/10.1109/CLOUD60044.2023.00024.

[50] S. Koyejo, Towards fault-tolerant federated and distributed machine learning, Proc. AAAI Symp. Ser. 3 (1) (2024) 306, http://dx.doi.org/10.1609/aaaiss.v3i1.31220, URL https://ojs.aaai.org/index.php/AAAI-SS/article/view/31220.

[51] Y. Mao, Z. Zhao, M. Yang, L. Liang, Y. Liu, W. Ding, T. Lan, X.-P. Zhang, SAFARI: Sparsity-Enabled Federated Learning With Limited and Unreliable Communications , IEEE Trans. Mob. Comput. 23 (05) (2024) 4819–4831, http://dx.doi.org/10.1109/TMC.2023.3296624, URL https://doi.ieeecomputersociety.org/10.1109/TMC.2023.3296624.

[52] X. Lin, Y. Li, X. Xie, Y. Ding, X. Wu, C. Ge, SF-CABD: Secure Byzantine fault tolerance federated learning on Non-IID data, Know.-Based Syst. 296 (C) (2024) http://dx.doi.org/10.1016/j.knosys.2024.111851.

[53] M. Zichichi, S. Ferretti, G. D'Angelo, On the efficiency of decentralized file storage for personal information management systems, in: Proceedings of the IEEE Symposium on Computers and Communications, IEEE, 2020, http://dx.doi.org/10.1109/ISCC50000.2020.9219623.

[54] L. Serena, G. D'Angelo, S. Ferretti, Security analysis of distributed ledgers and blockchains through agent-based simulation, Simul. Model. Pr. Theory 114 (2022) http://dx.doi.org/10.1016/j.simpat.2021.102413.

[55] OpenZeppelin Team, Access control mechanisms in blockchain smart contracts, in: Blockchain and Smart Contract Security, OpenZeppelin Documentation, 2022, URL https://docs.openzeppelin.com/contracts/4.x/access-control.

[56] L. Gigli, F. Montori, M. Zichichi, L. Bedogni, S. Ferretti, M. Di Felice, On the decentralization of mobile crowdsensing in distributed ledgers: an architectural vision, in: Proc. of the Proc. of the 2024 IEEE 21th Annual Consumer Communications & Networking Conference, CCNC 2024, IEEE, 2024, pp. 1–7.

[57] M. Bonini, M. Zichichi, G. D'Angelo, S. Ferretti, Proof of location through a blockchain agnostic smart contract language, in: Proc. of the 43rd IEEE International Conference on Distributed Computing Systems, ICDCS 2023, IEEE, 2023.

[58] D. Stutzbach, R. Rejaie, Understanding churn in peer-to-peer networks, in: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, IMC '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 189–202, http://dx.doi.org/10.1145/1177080.1177105.

[59] Infura Inc, Infura: Secure and scalable access to ethereum apis and ipfs gateways, 2020, https://infura.io/.

[60] M. Zichichi, S. Ferretti, G. D'angelo, A framework based on distributed ledger technologies for data management and services in intelligent transportation systems, IEEE Access 8 (2020) 100384–100402, http://dx.doi.org/10.1109/ACCESS.2020.2998012.

[61] K. Chen, Performance evaluation by simulation and analysis with applications to computer networks, John Wiley & Sons, 2015.

[62] Y. Sun, Z. Li, Y. Li, B. Ding, Improving LoRA in privacy-preserving federated learning, in: The Twelfth International Conference on Learning Representations, 2024, URL https://openreview.net/forum?id=NLPzL6HWNl.

[63] Y. Matsubara, M. Levorato, F. Restuccia, Split computing and early exiting for deep learning applications: Survey and research challenges, ACM Comput. Surv. 55 (5) (2022) http://dx.doi.org/10.1145/3527155.

[64] M.H.u. Rehman, A.M. Dirir, K. Salah, E. Damiani, D. Svetinovic, TrustFed: A framework for fair and trustworthy cross-device federated learning in IIoT, IEEE Trans. Ind. Inform. 17 (12) (2021) 8485–8494, http://dx.doi.org/10.1109/TII.2021.3075706.

[65] F. Barbàra, M. Zichichi, S. Ferretti, C. Schifanella, DLT-based personal data access control with key-redistribution, in: 2023 5th International Conference on Blockchain Computing and Applications, BCCA 2023, 2023, pp. 166–173, http://dx.doi.org/10.1109/BCCA58897.2023.10338895.

[66] H. Jia, M. Yaghini, C.A. Choquette-Choo, N. Dullerud, A. Thudi, V. Chandrasekaran, N. Papernot, Proof-of-learning: Definitions and practice, in: Proceedings of the 42nd IEEE Symposium on Security and Privacy, 2021.

[67] P. Valiant, Incrementally verifiable computation or proofs of knowledge imply time/space efficiency, in: R. Canetti (Ed.), Theory of Cryptography, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 1–18.