Scatter Protocol: An Incentivized and Trustless Protocol for Decentralized Federated Learning

ABSTRACT

Federated Learning is a form of privacy-preserving machine learning where multiple entities train local models which are then aggregated into a global model. Current forms of federated learning rely on a centralized server to orchestrate the process, leading to issues such as requiring trust in the orchestrator, the necessity of a middleman, and a single point of failure. Blockchains provide a way to record information on a transparent, distributed ledger accessible and verifiable by any entity. We leverage these properties of blockchains to produce a decentralized, federated learning marketplace-style protocol for training models collaboratively. Our core contributions are as follows: first, we introduce novel staking, incentivization, and penalization mechanisms to deter malicious nodes and encourage benign behavior. Second, we introduce a dual-faceted lottery-based validation layer to ensure the authenticity of the models trained. Third, we test different components of our system to verify sufficient incentivization, penalization, and resistance to malicious attacks.

CCS CONCEPTS

• Computing methodologies → Distributed algorithms; **Machine learning**; **• Security and privacy** → Distributed systems security; **Trust frameworks**.

KEYWORDS

Blockchains, Distributed Systems, Federated Learning, Smart Contracts, Ethereum, Tokenization

ACM Reference Format:

. 2023. Scatter Protocol: An Incentivized and Trustless Protocol for Decentralized Federated Learning. In . ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

With the rise of artificial intelligence - particularly through foundation models - researchers, companies, and other entities alike have become increasingly reliant on data and compute power. One way to extend the limits of the amount of data and compute that can be used for training models is through federated learning. Federated learning is a form of collaborative machine learning where multiple entities aggregate locally trained weights into one global model. With federated learning, models are trained with data privacy in mind - this means that each entity has a local dataset that is not exposed to the rest of the participants, which is used to train the model. Current approaches with federated learning require a centralized authority to orchestrate the process and validate the model. Centralized federated learning results in local entities needing to be able to trust a single authority to ensure the validity of the process (i.e. models are trained properly, models are aggregated properly, etc.), introducing inefficiencies. This causes three issues:

- A necessity of a root of trust, which, in turn, inhibits federated learning from being used across entities who do not trust each other
- A single point of decision-making that is prone to failure or malicious acting
- A lack of incentives to participate in federated learning due to the aforementioned shortcomings.

A common alternative approach to centralized federated learning is decentralized federated learning. Decentralized federated learning is when a system of nodes are able to coordinate themselves to obtain a global model without the help of a middleman or orchestrator. The process of decentralized federated learning can be done in many ways.

We propose a blockchain-based protocol for decentralized federated learning that integrates smart contracts into the federated learning process to create a trustless environment while incorporating decentralized validation and incentivization layers. Our system has four primary roles: requesters, trainers, validators, and challengers. The high-level overview of our system works as follows: first, requesters will request a specific model to be trained by publishing a training job. Next trainers agree to train a model based on the data they have. After, all the trainers have trained a model or the training job terminates, validators will validate the model via a consensus using a validation job published by requesters. Finally, challengers can challenge nodes that may conduct fraudulent actions during the federated learning process. Each role has associated tokenomics - a general term referring to economic systems revolving around a token - via staking, taxes, and lottery mechanisms, which incentivizes all network participants to act benevolently. We publish data to the blockchain via smart contracts to act as the unified source of truth for all the transactions that occur on our network.

This work serves as a cornerstone for the future of federated learning because it eliminates many of the inefficiencies associated with centralized federated learning. More specifically, our contributions to eliminating issues with federated learning are as follows:

- We eliminate the necessity of trust, middlemen, and a single point of decision making to instead rely on a distributed quorum based voting mechanism to facilitate the process of federated learning
- We introduce multiple layers of decentralized validation - first through validators, and second through challengers - to determine whether transactions and nodes on our protocol

are valid, reinforcing the protocol's ability enable authentic federated learning

- We implement a tokenomics system that takes advantage of staking, tax, and lottery mechanisms to reward participating nodes in the network based on performance, minimizing malicious behaviors in federated learning

2 RELATED WORKS

Federated learning is currently an active area of research due to its benefits in multiple realms including privacy, decentralized compute and data, and model aggregation. For this work, we look at five primary areas of related work (1) blockchains and smart contracts, (2) voting and decision making mechanisms, (3) malicious node detection, (4) model validation, and (5) federated learning evaluation.

2.1 Blockchains and Smart Contracts

Smart contracts are pieces of code that can be executed on a blockchain. Both smart contracts and blockchains are completely transparent, permissionless, and verifiable by any entity. We leverage these properties to implement our incentivization, validation, and voting mechanisms.

Prior works explore the idea of applying blockchains and smart contracts for decentralized federated learning infrastructure. Work by Behera et al. explored how we could use smart contracts to address challenges such as model validation, transaction transparency, and model leaks. To address incentivization, they introduced the idea of using blockchain-native tokens to reward nodes based on performance [1]. Blockchain-native tokens governed by smart contracts enable easily implementing incentivization mechanisms while presenting a deterministic and permissionless method to distribute tokens (as defined by the smart contract). Their work with incentivization schemes is further substantiated by Cachin et al. and Haoxiang et al., who also conceptualized alternate decentralized, federated learning systems with a focus of incentivization [3][21].

Complete transparency in a blockchain based system also means a lack of privacy with transaction data (i.e. datasets, models, etc.). Behera et al. utilized common cryptography algorithms like RSA to mask private data in public transactions on the ledger[1]. This enabled them to communicate model weights in an encrypted manner while keeping the transaction itself public. This ensured that peer nodes could not view the model it was only available to the aggregation server.

2.2 Voting and Decision Making Mechanisms

In collaborative protocols, voting and decision making mechanisms play a large role in decentralized governance. We leverage decentralized governance to make decisions about the authenticity of specific transactions within our protocol (i.e., whether a model was trained properly, validated properly, etc.).

Works by Fritsch et al. and Li et al. explore liquid democracy - a form of decentralized governance - which is seen through Delegated-Proof-of-Stake based blockchains and decentralized autonomous organizations. In a liquid democracy, users can delegate their voting power to other users or vote directly on specific decisions, resulting in higher voter engagement [9][15]. However, liquid democracy sacrifices decentralization and security. For instance, the most rational action for entities with small amounts of voting power would be to delegate it all to one entity, leading to pools of concentrated voting power. Furthermore, rich entities can create pseudo-identities by running multiple nodes with high stakes - leading to weak identity management [13]. Lalley et al. explores quadratic price voting - an alternative to liquid democracies - where instead of entities delegating votes, votes can be bought. However, the price of votes increases quadratically with each additional vote. Quadratic voting allows participants to compare the marginal cost of an additional vote against the perceived chance it will be able to swing a decision [14]. This alleviates some of the concerns associated with decentralization because participants are not incentivized to buy an infinite amount of votes.

Work by Joshi has studied Proof of Authority for decision making. Proof of Authority a permissioned consensus mechanism that attempts to make decisions based on a set of authorized validators. It relies on reputation mechanisms as a form of incentivization for nodes to make the best decisions for the protocol. However, it also requires some loss of decentralization because of its permissioned nature and security due to the identity of validators being public, potentially causing third party manipulation [12].

2.3 Malicious Node Detection

In decentralized systems, malicious actors are inevitable. Being able to identify and punish malicious actors in the context of federated learning is central to the success of the protocol. Having malicious node detection mechanisms in place will disincentivize malicious actions.

Prior work by Nanqing et al. explores the idea of malicious node detection by utilizing a voting mechanism and proof-of-stake mechanisms as insurance against malicious nodes [5]. Their works leverage the ideas from aforementioned voting mechanisms to ensure malicious nodes are voted out and the network stays as malicious-free as possible.

On the other hand, Zhu et al. work on malicious node detection deterministically by converting the gradient space into a ranking matrix. From this, they calculate the mean and standard deviations of the ranking matrix and apply a K-means clustering algorithm to predict where the benign and malicious cluster nodes are (with the assumption that the largest cluster consists of benign nodes) [22].

2.4 Model Validation

With decentralized learning, being able to ensure that the models that were trained are valid is necessary for adoption. To do this, we can utilize model validation mechanisms to determine a model's legitimacy and, as a result, punish or incentivize a specific node.

Research by Chen et al. introduces a framework for decentralized model validation via voting mechanisms [4]. This framework works by having validators perform one epoch of local training on the global model. The test dataset is run through these models and the accuracies of the global and local models are compared to see if the difference in accuracies is below a specific threshold. A vote is then cast by the validators on whether a model has been distorted.

By leveraging consensus, decentralized systems can maintain a trustless state while adding a robust validation layer.

Another approach to model validation was taken with zero-knowledge proofs - a mechanism that enables an entity to prove a statement without revealing any additional information information, an approach commonly used with privacy-centered systems. Work by Heiss et al. and Xing et al. both leverage zero knowledge proofs to which entailed being able to create arithmetic circuits (a set of constraints that prove a computation was carried out correctly) to prove that a model that was trained in a valid way [10][20]. They note there are some bottlenecks to be considered with the size and computational power required to create zero-knowledge proofs. However, zero-knowledge proofs provide a deterministic and easily verifiable computation to prove a model's validity.

2.5 Federated Learning Evaluation

Evaluating decentralized systems is an important active area of research to us assess whether components of a system perform as intended while still performing their end goal.

Research from Beltrán et al. and Dong et al. touch on these metrics that evaluate various components of their decentralized system. They assess the networking topology by measuring network capacity (i.e., latency, throughput, up-time, bandwidth, jitter, etc.). Machine learning model performance is measured via common machine learning model indicators (i.e., loss, accuracy, convergence time, sensitivity, etc.). Incentivization and penalization schemes were measured by tracking token supply over time for malicious and benevolent nodes. [2][6].

3 PROTOCOL DESIGN

3.1 Protocol Overview

Our work provides an end-to-end protocol and set of rules for incentivization, penalization, validation, and consensus for decentralized federated learning. Our protocol takes the format of a marketplace with four different roles:

- **Requester:** Requesters are the "buyers" of the marketplace. They request specific training jobs to be completed and also provide reward with the training job. The reward has no lower or upper bound because participation is voluntary amongst nodes. Instead, we let the market determine the rewards requesters set - pricing too low will result in a lack of participation and too high will result in a net negative for the requester. The training jobs are uploaded to the Interplanetary File System - a decentralized storage solution - and the content ID hashes are published on the blockchain for nodes to inspect. Requesters also publish an evaluation job in a similar fashion for validators to run later in the federated learning process.
- **Trainer:** Trainers are the core worker nodes and primary "sellers" of the marketplace. They provide local datasets and compute power to train local models for the requesters. Due to the transparent nature of the blockchain, they are able to inspect any training jobs to determine dataset type, malicious code, etc. to determine whether they want to join a training job. Trainers can also inspect the evaluation jobs to ensure that the job does not maliciously mark a model they have trained as invalid.
- **Validator:** Validators are the first layer of validation in the protocol. Their role is to take models trained by trainers and determine the validity of the models. They do this by running an evaluation job (also provider by requesters) on an evaluation dataset and publishing the results of the evaluation job in the form of a score.
- **Challenger:** Challengers are the final layer of validation before a training job is complete. Challengers can challenge any part of the process which can be checking whether a model is valid, whether a validation is valid, and even can challenge other challenger nodes who may publish incorrect challenges. Essentially, their role is to generally check whether any node participating in the protocol is malicious. This is generally done through simulating different transactions or operations but the implementation and algorithms are open-ended and specific implementation details are left to the node client. In order for a challenge to be successful, the challenger must gain consensus with other challenger nodes on whether the challenged node is malicious.

In essence, our protocol's general workflow can be simplified down into eight steps:

(1) Requesters interact with the blockchain by publishing a training job and evaluation job for a specific topic (topic representing what they want the model to do - i.e., a classification model for numbers).
(2) Trainer jobs read the topics on the blockchain and inspect the jobs. If they see a topic they can complete, they can choose to subscribe to it.
(3) A requester can choose to kick off the training job at anytime before a time threshold (a fixed predetermined time set by our protocol). After the time threshold has passed, any node in the network can kick off the training job. This emits an event indicating trainers to start training.
(4) Once a trainer has completed its training job, it encrypts the model data and publishes the models to the Interplanetary File System for validators to inspect
(5) Once all trainers have published a model, the validators will start validating the models and publish scores to the blockchain. To decrypt the models, the validators request a decryption key from the trainers.
(6) After validations have occurred, challengers can analyze node behavior data to determine whether there were any malicious nodes and publish these results.
(7) The protocol will then distribute rewards based on the performance and behaviors of nodes (as outlined in 3.2)
(8) The requester reads model data from the blockchain and aggregates them into one model. The responsibility of model aggregation is left to the requester because it was the initial purchaser of the locally trained models. Also, because of this, the aggregation algorithm of choice (i.e. FedAvg, FedSoftBetter, FedWorse, etc.) is left to the requester [16].

3.2 Incentivization and Penalization

The protocol defines different incentivization and penalization rules for each role as well as protocol-wide rules to maximize security and benevolent behavior while minimizing malevolent behavior.

To implement incentivization and penalization mechanisms, we use Ethereum Request for Comment-20 (ERC-20) tokens - a standard in Ethereum that enables us to make our own token systems - to implement a protocol currency: scatter tokens. The total supply of our scatter tokens is set to N. Each training job transaction on the network has a tax rate set at D_{tx}. The transactions build a token pool to be used in other areas of the protocol. We also have distribution rates for both trainers, D_{tr} and validators, D_v that specify the proportion of a training job's reward pool that goes to trainers and validators. We can summarize the rates as follows:

$$D_{tx} + D_{tr} + D_v = 1 \tag{1}$$

Requesters on the network are subject to two rules. First requesters must provide some reward for each training job they create. If we denote the reward pool to be R_p, the amount of reward taken as tax to be R_{tx}, the rewards given to trainers as R_{tr} and rewards given to validators as R_v, the total reward pool can be represented as:

$$R_p = R_{tx} + R_{tr} + R_v \tag{2}$$

We have also established that the protocol takes on a D_{tx} tax rate and we can represent the tax amount as follows:

$$R_{tx} = R_p \cdot D_{tx} \tag{3}$$

Secondly, if a requester is found to be malicious, the reward that they have allocated is equally distributed amongst training job participants with no returns. The protocol is designed so that rewards that have been offered to participants cannot be withdrawn which ensures that requesters do not withdraw offered rewards in the case that they are found malicious.

Finally requesters are given some security assurance in the scenario that trainers and/or validators do not fulfill their roles. In the scenario that 100 percent of trainers and 100 percent of validators are found to be malicious, then requesters get a complete refund on their training job transaction (with the exception of the protocol-wide tax).

The tokenomics of trainers introduces a system that encourages proper training of models. To participate in a training job, trainers are required to stake - a mechanism commonly used in blockchains where tokens are locked into the protocol as collateral for additional privileges - any amount of tokens. The unbonding period - the period of time an entity must wait before being able to unstake tokens from the protocol - of the tokens is the lifetime of the training job (as long as the training job is active the tokens cannot be unstaked). In the scenario where a trainer is found to be malicious, they lose all tokens that they have staked. By introducing a short term staking mechanism for trainers, we achieve two things: 1) trainers are now risking tokens and therefore are more incentivized to train better models and 2) we can reward trainers accordingly based on how much stake they have decided to dedicate to a specific training job.

For our reward function, given a pool of trainers $T = \{t_1, t_2, \ldots, t_k\}$ we can define $W^t_{st_j}$ as the stake amount for the j^{th} trainer node and $W^t_{sc_j}$ for the average validation score of the j^{th} trainer node (as determined from the validators). We can define the reward function for a trainer i, R_{t_i}, as follows:

$$R_{t_i} = \frac{\sqrt{W^t_{st_i}} \cdot {W^t_{sc_i}}^2}{\sum_{j=1}^{k} \sqrt{W^t_{st_j}} \cdot {W^t_{sc_j}}^2} \cdot R_{tr} \tag{4}$$

We can decompose this reward function into two parts. The first part calculates the proportion of the reward pool allocated to trainers that should go to trainer i. We do this by calculating a reward score which is the product of the square root of the stake amount and the square of the average validation score ($\sqrt{W^t_{st_i}} \cdot {W^t_{sc_i}}^2$) and dividing it by the sum of all reward scores. The two factors that are taken into the reward score ensure there is incentive for trainers to both stake tokens into a specific training job as well as train a valid model. We square root the stake to avoid a scenario where trainers with large amounts of stake do not completely dominate the distribution of the rewards, leaving a disproportionately small amount of rewards for the remainder of the trainers. This ensures diminishing marginal returns, especially as stake amounts become increasingly larger. We also take a partially achievement-based approach to our reward function as well by squaring the average validation score to more heavily weigh trainer performance [17]. By squaring the validation score, nodes receive increasing marginal returns from focusing on better performance. The second part of our reward function calculates the amount of the reward pool that should go to trainers (R_{tr}). We can express as a product of the reward pool and the trainer distribution rate:

$$R_{tr} = R_p \cdot D_{tr} \tag{5}$$

Validators are subject to a long-term staking requirement of S_v tokens. These tokens have an unbonding period for a protocol-specified time period of τ_v. Validators can either validate models correctly or incorrectly. For our reward function, given a pool of validators, $V = \{v_1, v_2 \ldots v_k\}$, we can define the stakes for an arbitrary validator, i, to be $W^v_{st_i}$. We define our reward function for validator i, R_{v_i}, as follows:

$$R_{v_i} = \frac{W^v_{st_i}}{\sum_{j=1}^{k} W^v_{st_j}} \cdot R_v \tag{6}$$

This reward function is entirely based on stakes. Because each validator performs the same amount of work, there is no performance factor like there was in the trainer reward function. By keeping the reward function entirely based on stakes, it encourages validators to stake more tokens to support the protocol. Similar to the trainers, this reward function calculates a proportion of the total reward pool for validators (R_v) to give out. We can express as follows:

$$R_v = D_v \cdot R_p \tag{7}$$

Because validators can also be malicious, we design a penalization function that penalizes validators based on number of previous penalties. Given a validator, i, with a stake amount of S, we define a base penalty, P_b, a penalty multiplier P_m, and the penalty count of this specific validator as P_c. We calculate the penalty for validator i, P_{v_i}, as follows:

$$P_{v_i} = \left\{ \begin{array}{ll} P_b \cdot {P_m}^{P_c}, & \text{if } P_b \cdot {P_m}^{P_c} \leq S \\ S, & \text{if } P_b \cdot {P_m}^{P_c} > S \end{array} \right\} \tag{8}$$

Table 1: Protocol Tokenomics Parameters

Paramter Name	Paramter Symbol	Parameter Value
Token Supply	N	100 Billion Tokens
Tax Rate	D_{tx}	0.10 (10%)
Trainer Reward Rate	D_{tr}	0.70 (70%)
Validator Reward Rate	D_v	0.20 (20%)
Validator Staking Requirement	S_v	25000 Tokens
Validator Unbonding Period	τ_v	30 Days
Challenger Staking Requirement	S_c	20000 Tokens
Challenger Unbonding Period	τ_c	30 Days
Base Penalty	P_b	200 Tokens
Penalty Multiplier	P_m	2

This function discourages subsequent malicious behaviors by validators by introducing increasing marginal costs for each additional malicious behavior committed. When a validator's stake reaches zero, it automatically loses validator privileges.

Challengers, like validators, are also subject to a long-term staking requirement of S_c. These tokens have an unbonding period for a protocol-wide specified time of τ_c. Challengers do not receive rewards for each transaction they challenge. Instead, all challengers are racing to compete for a lottery (which is accumulated from the taxes collected). On a successful challenge, the challenger will receive the tokens from the lottery. Challengers can also be challenged themselves - in this scenario, challengers follow the same penalty function imposed on validators in equation 8.

We define the parameters for our protocol's incentivization scheme. Our implementation of the protocol sets the parameter values as defined in Table 1.

3.3 Smart Contract Architecture

Our protocol consists of seven different smart contracts that define the behavior of the system. The first smart contract we created was a training job token contract. The training job token contract is an ERC-721 contract standard (a contract standard for non-fungible tokens) that defines the logic around handling different training jobs. This contract creates an immutable token representing the training job. The token contains a reference to the training job details (the content ID of the training job) stored in the Interplanetary File System. This allows the protocol to declare that a specific node has ownership over a training job, allowing enabling better observability of the different training jobs available.

The next contract we have is a model token contract. This contract is another ERC-721 based contract that defines the logic for models and their ownership. Similar to the training job token contract, this contract creates a token representing each submitted model and attributes it to the respective trainer for better observability over the different models trained. Additionally, this contract contains logging capabilities that allows other parts of the protocol to see which trainers have and have not submitted models for a specific training job.

We implement an evaluation job contract to manage an evaluation job. Like the model token and training job token contracts, this is an ERC-721 contract that defines ownership of the different evaluation jobs. Additionally, evaluations jobs have two different parts to them: the job portion and the associated dataset. When initially publishing a training job and its respective evaluation job, requesters only publish the job portion so that trainers cannot exploit the evaluation dataset and train the model on the dataset itself. Because of this, this smart contract provides functionality to publish an evaluation dataset later down the road and associate it with the previously published evaluation job token.

The protocol contains a reputation manager contract which is a smart contract used to determine benevolent and malicious nodes in the network for a specific training job (i.e., this reputation is not globally persistent across all training jobs). The reputation manager is used in other areas in the protocol to determine whether a specific node should be rewarded or penalized based on the schemes presented in 3.2.

We introduce a governance mechanism through the vote manager smart contract. This contract facilitates the validation process for models submitted by trainers. Validators utilize this contract to participate in the voting process by submitting scores after running the validation job on each model. The contract contains logic to make a decision on whether a model is valid after votes are submitted. more details on the specifics of the voting and decision making mechanism can be found in 3.4 This contract also enables validators to flag a specific evaluation job as malicious and refrain from the voting process.

We implement our incentivization layer using an ERC-20 contract standard (a contract standard for fungible tokens which enables protocol-specific currencies) with our scatter token contract. The scatter token contract provides functionality for nodes to stake tokens, both short-term for trainers and long-term for validators and challengers. It also implements the reward and penalization logic for all nodes as specified in 3.2. Finally, it contains the logic that manages protocol-wide requirements like the tokenomics parameters specified in Table 1.

The final contract we implement is the scatter protocol contract which is the main entry-point and interface for nodes to interact with the protocol. It contains actions like creating training jobs for requesters, joining training jobs for trainers, challenging transactions, etc. It integrates with all the other protocol contracts into a unified transaction layer - in essence, all transactions to other contracts are proxied through the scatter protocol contract.

Figure 1 illustrates the different smart contract and their interactions with one another.

3.4 Model Validation and Consensus

Our protocol implements a validation layer through validators. Validators rely on governance (via voting) to determine whether specific models are valid or invalid. How this validation layer works is that each job will have a proportion of the validators, V_{prop} assigned to each training job. These validators are randomly chosen through the smart contract which leverages the Fisher-Yates shuffle algorithm along with block timestamp, difficulty, and a random nonce for a degree of stochasticity, preventing entities from consistently validating their own jobs by running multiple nodes [11].

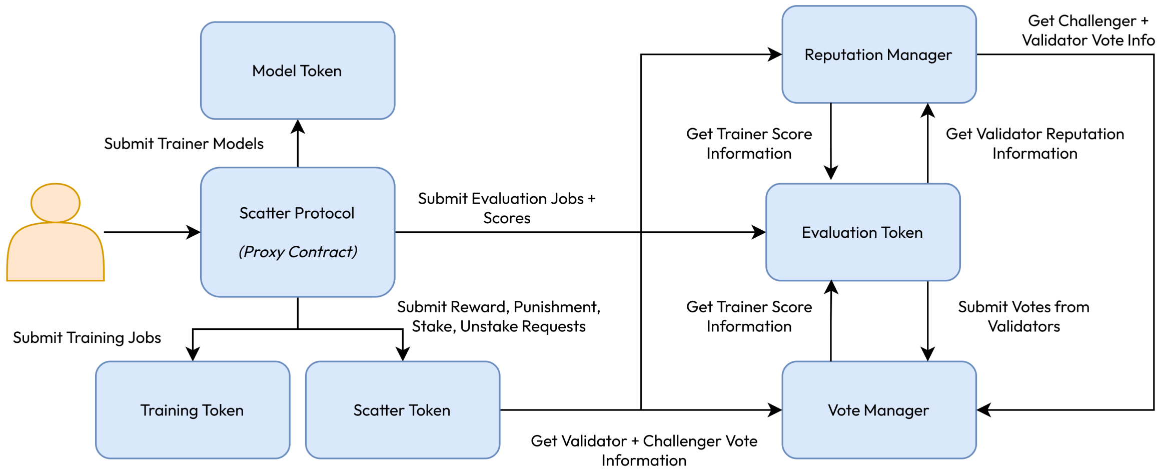
Model Token
Submit Trainer Models
Scatter Protocol
(Proxy Contract)
Submit Evaluation Jobs + Scores
Submit Training Jobs
Submit Reward, Punishment, Stake, Unstake Requests
Training Token
Scatter Token
Get Validator + Challenger Vote Information
Reputation Manager
Get Challenger + Validator Vote Info
Get Trainer Score Information
Get Validator Reputation Information
Evaluation Token
Get Trainer Score Information
Submit Votes from Validators
Vote Manager

Figure 1: This figure illustrates the smart contract architecture and how each smart contract interacts with one another.

Once the models are ready for validation for the validators, the smart contract emits an event indicating that all models have been uploaded and the validators can start validating models with the given evaluation job and submitting evaluation scores. The evaluation job comes with an associated evaluation dataset (also provided by the requester). This dataset is published after the trainers have submitted their models to prevent trainers from exploiting the evaluation job by training their models directly on the evaluation dataset.

We allow validators to continue to submit scores until a minimum number of validators have voted, Q_c (this is the quorum count). Once the quorum count has been reached, the average score for each model is calculated and if it is above the validation threshold (set by the requester when submitting a job), then it is accepted, else rejected.

Another area where validators rely on consensus is when a requester submits a malicious evaluation job (i.e., the evaluation job contains code that is deemed malicious). In this scenario, the validator can refrain from conducting an evaluation. If a super-majority of validators (two-thirds) refrain from validating, then all validators are compensated. Otherwise validators that refrain without a super-majority, are neither rewarded nor punished. Validators are given ultimate authority to determine whether a requester is malicious because validators are chosen at random for each training job. Therefore validators could be a target for malicious training jobs from requesters to avoid having to pay a portion of the rewards out as seen in the scenario in section 4.4. Trainers on the other hand do not have this authority because trainers can see all training jobs before they commit to them. Therefore, before a trainer commits to a training job, they can evaluate for themselves whether it is malicious or not and act in their best interests.

Our secondary validation layer with challengers leverages a different governance mechanism to achieve consensus - when a challenger decides it would like to challenge the results of anode, all challengers are signaled to vote on whether a node is fraudulent (a binary yes or no decision). In order for a challenge to to be accepted, at least two-thirds of challengers must reach a consensus on the decision. Because challengers are the last line of defense before a training job reaches finality and challengers have the highest earning potential if a challenge is successful, we require a supermajority.

3.5 Client Node Architecture

We provide a client node that participants can run to interact with the protocol and run training jobs from the protocol. Our client node has the following components:

- **HTTP Server:** The HTTP server is the main entry-point to interact with the node. There are various endpoints that enable network participants take actions like creating a training job, viewing all the available training jobs, getting scatter token balance, and starting the training procedure for a specific training job. The HTTP server has handlers which interact with the blockchain when a transaction needs to occur.
- **Peer-to-Peer Server:** The peer to peer server is used for inter-peer communication. Because the blockchain is a medium of completely transparent and public exchange, it is difficult to share private information directly. We use a peer-to-peer server to facilitate communication for private information. This primarily takes the form of communicating AES keys to nodes directly to decrypt data on-chain - we expand on how we use our peer-to-peer server in 3.6.
- **Asynchronous Job Queue:** We anticipate trainer nodes to subscribe to multiple training jobs at once. To ensure there is not an excessive number of training jobs occurring at once which may overwhelm the system the node is running on, we created an asynchronous job queue that executes training jobs from a job pool, allowing for better scalability. Currently, the workers pull from the pool in a first-in, first-out manner; however, the order in which they pull from the pool and execute the jobs can be customized and is up to the node operator.

- **Protocol Event Listeners:** The smart contracts emit various events such as models are ready to validate, training has been initialized, etc. For these events, nodes need to react in a specific ways so we provide an event listener module that listens to events emitted from the blockchain and conducts an action accordingly.
- **Datastore:** Our node relies on a datastore to handle meta-data information during different transactions. This includes things like saving training job or evaluation job information to reference in later parts of the federated learning process. We use PostgreSQL for our datastore.

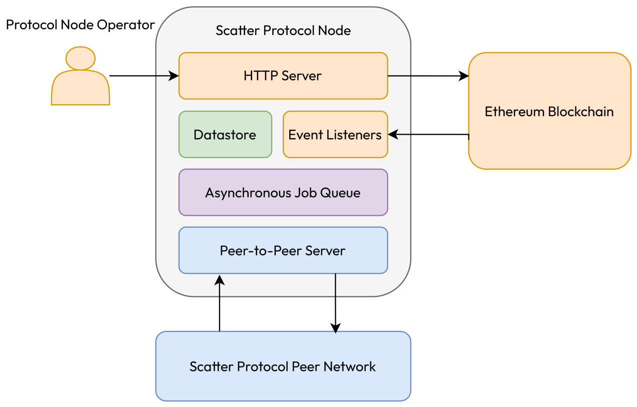

Figure 2: Node Architecture of Scatter Protocol Node

Figure 2 is a high level overview of the client node architecture. This is just one implementation of a potential client node - the permissionless nature of the protocol enables anyone to create their own client node implementation and interact with the protocol directly.

3.6 Security

Security in our system is introduced both at the protocol level as well as the node level.

At the protocol level, we leverage our incentivization and penalization mechanisms as outlined in 3.2 to discourage malevolent behavior like publishing malicious training jobs. Our smart contract also accounts for common smart contract attacks like reentrancy attacks and overflow/underflow attacks [19].

At the client node level, we enforce security through three channels: training job runtime, data, and peer communication. For our training job runtime environment, we rely on Docker containers which provides a level of isolation with each container having its own file-system, process space, and network interface. We also leverage the Open Container Initiative (OCI) runtime through gVisor which provides a seperate operating system kernel and virtual machine monitor. It intercepts system calls, enabling greater isolation between the application and the host operating system [18].

We also implement encryption standards that are utilized both at the protocol and client-node level. Specifically, we encrypt models that are published to the blockchain using Advanced Encryption Standard-256 (AES-256). We do this because this protocol is asynchronous in nature which means that trainers can publish their models at different times. In the case that one trainer publishes their model before another, the one that has not published the model would be able to publish the other trainer's model without doing the computational work to train a new model. To prevent this, we encrypt the model using AES encryption. When the validator is ready to evaluate a model, they request the decryption key via the peer-to-peer server and decrypt the encrypted model before validating it. If the trainer does not provide a valid decryption key, the validator can mark the trainer as malicious. In doing this, we are both able to ensure that the trainer is training their own models while still being able to publish a model publicly (albeit it being encrypted).

Another security concern arises with communicating the decryption key over the peer-to-peer server. We leverage the Noise Protocol framework via the LibP2P library which enables us to create secure and authenticated channels of communication between two peers [7]. This ensures that we can securely share the decryption key with other peers.

Table 2: Simulation Parameters

Paramter Name	Parameter Value
Requester Node Count	1 Node
Trainer Node Count	6 Nodes
Validator Node Count	3 Nodes
Challenger Node Count	2 Nodes
Number of Training Jobs Completed	10 Jobs
Initial Token Supply (Per Node)	100000 Tokens

4 EVALUATION AND RESULTS

To evaluate our system, we focused on simulating different scenarios to determine how the tokenomics and protocol logic behaved. To evaluate these, we run simulations on a custom built simulation software which allows us to run and control the actions of nodes. We leveraged Hardhat network, a local blockchain development tool to run on-chain transactions. For our simulations, we consider five different scenarios. For each scenario, our simulation has the parameters specified in Table 2. The number of malicious and benevolent nodes varies from simulation to simulation.

4.1 All Benevolent Nodes

Our first scenario is when all trainers and validators are benevolent. Figure 3 illustrates the token supplies for the trainers, validators, and challengers. For trainers, we notice throughout the life-cycle of the simulation, trainers periodically lose some amount of token supply. This is due to the short-term stake on a training job basis that was described in section 3.2 which is returned to them at the end of the training job. We notice that ultimately because of the consistently benign behavior, trainers do make a profit as at the end of the simulation, their token supply is above the initial token supply. With the validators, we initially see a large drop off due to the required stake to become a validator but after that, the validators slowly accumulate tokens, also demonstrating profitability from the validator standpoint. Finally, with challengers, we see that there is an initial drop off due to the stake (similar to validators) but there is no increase in token supply because all nodes are acting correctly and hence there is no opportunity to win the lottery.

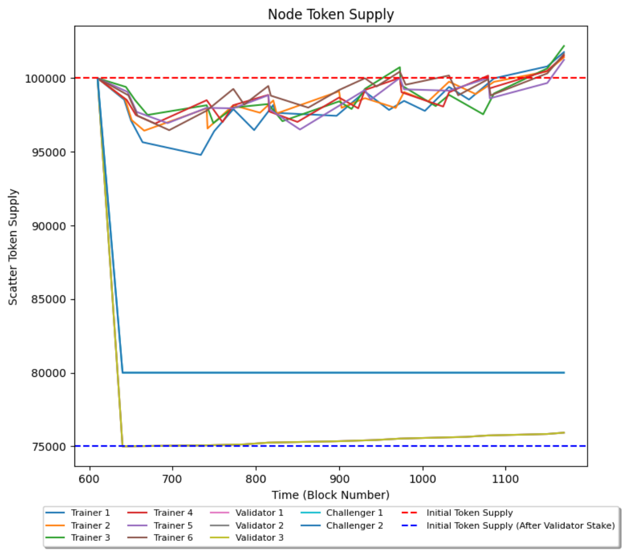

Figure 3: Token Supplies for Benevolent Trainers and Validators (Section 4.1)

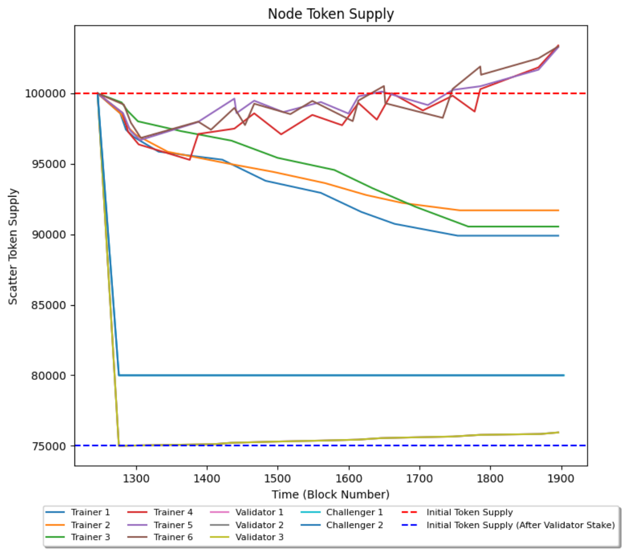

Figure 4: Token Supplies for Partially Malicious Trainers and Benevolent Validators (Section 4.2)

4.2 Partially Malicious Trainers

Our second scenario occurs when we have some malicious trainers, some benevolent trainers, and all benevolent validators. We designate three of our trainers as malicious and three as benevolent. We can see the token supplies over time in Figure 4. For this scenario, we notice that the trainers who are not malicious have their token supply increasing from the rewards of the training job. On the other hand those who are malicious have their token supply decreasing because they lose the stake they've invested into the protocol for being malicious. Validators continue to slowly earn rewards as well as they are still acting as intended. Challengers do not earn rewards here, even if they help with identifying a node as malicious because validators are given priority for malicious node identification and hence generally identify the malicious trainers first. This ensures challengers do not reap rewards for marking a trainer as malicious just because a validator does (i.e., a challenger is required to actually put in computational work to determine the validity of a trainer).

4.3 Partially Malicious Validators

Our third scenario occurs when we have all benevolent trainers and some malicious validators, and some benevolent validators. We designate one of our validators as malicious and two validators as benevolent. We can see the token supplies over time in Figure 5. If we look at the trainers, because they are all benevolent, their token supplies allow a profit over time. With the validators, we notice that one validator has a stagnant token supply. That is the malicious validator and has been identified as malicious by the challengers and therefore does not earn a reward. On the other hand, the two benevolent validators do continue earning rewards. The challengers also see an increase in token supply but at a much steeper rate than any other nodes. This is because challengers are winning tokens from the lottery mechanism. The challenger that starts the challenge for a specific node first and is successful is the winner of the lottery which is also demonstrated in Figure 5. We also notice that the trainers continue to make a profit despite the malicious validators. This is attributed to the fact that when a validator is marked as malicious, their evaluations are omitted when determining which trainers have acted maliciously.

We also notice that while the fraudulent validator here is not earning any rewards, it is also not being punished for it. This is because the punishment is taken out from the validator's protocol stake, not the node's token supply. Figure 6 demonstrates how this stake decreases exponentially with every subsequent penalty (the penalty function is defined in section 3.2). Once the stake reaches zero, that validator loses privileges to evaluate models.

Finally, we can also observe the lottery supply in Figure 7. We notice that the lottery supply has periodic spikes and the magnitude of these spikes increase. The lottery is determined by a variety of factors like protocol taxes, trainers stakes, and validator stakes. Because the validator stake penalty increases exponentially, the amount that is transferred to the lottery with each penalty increases at that same rate. This lottery is then transferred to the challenger who discovered the malicious validator. This flow from validator to lottery to challenger is the reason behind the spikes observed.

4.4 Malicious Validators and Trainers

The fourth scenario is when all validators and trainers are malicious. In figure 8 We notice that in this scenario that challengers are receiving the maximum profit. The large profits can primarily be

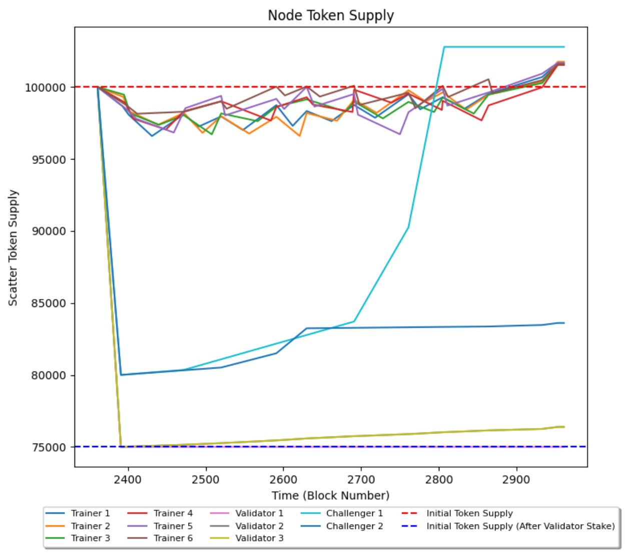

Figure 5: Token Supplies for Partially Malicious Validators and Benevolent Trainers (Section 4.3)

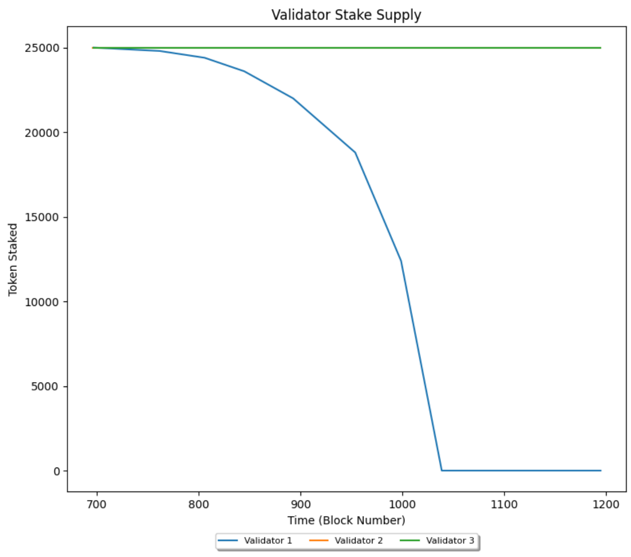

Figure 6: Stake Amounts for Validators in a Malicious Validator Scenario (Section 4.3)

associated from two factors 1) lost validator stakes and 2) lost trainer stake. Because both trainers and validators are losing their stakes to the lottery for being malicious, this is when challengers are able to earn the maximum profits while maintaining in the integrity of the network. We also notice that the requester initially loses a portion of its token supply that was meant to act as the reward pool but then recovers most of it. This is because requesters are returned tokens if 100 percent of validators, 100 percent of trainers fail, or both. It does not completely recover all tokens due do the

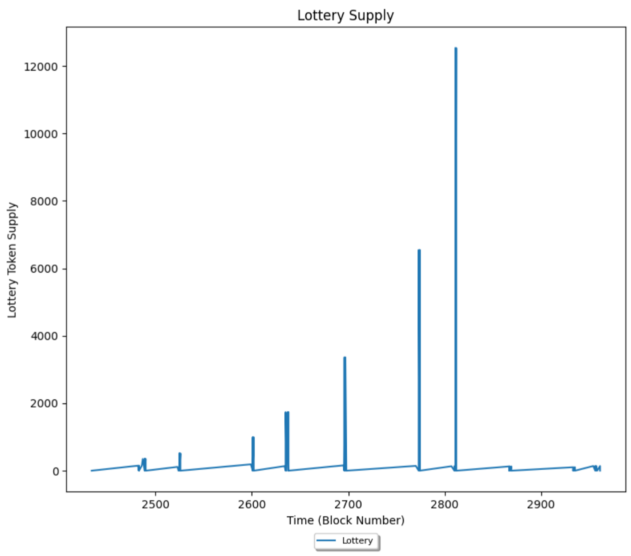

Figure 7: Lottery Supply in a Malicious Validator Scenario (Section 4.3)

lottery tax; however, it does recover the tokens that were originally allocated for trainer and validator rewards.

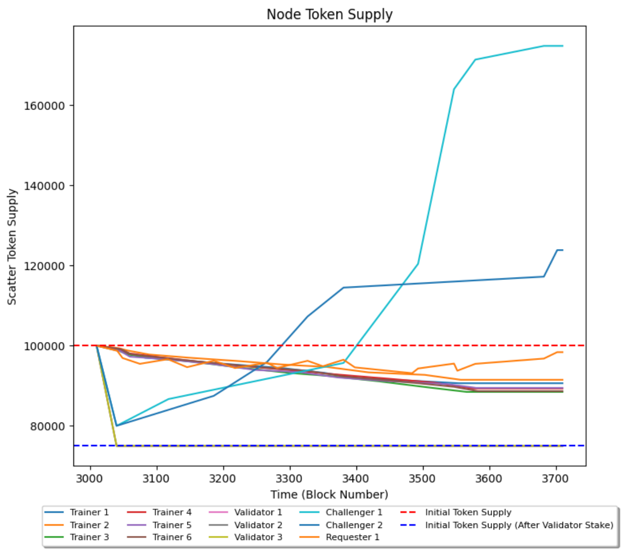

Figure 8: Token Supply When All Trainer and Validator Nodes are Malicious (Section 4.4)

4.5 Malicious Requester

The final scenario we consider is when the requester node is malicious and all other nodes are benevolent. In this scenario, validators can flag the requester as malicious. This causes two things to happen: 1) the validators gives every trainer an equal validation score and 2) a vote to see if there is consensus on whether that requester is malicious. If a consensus is gained as outlined by section 3.4, then both the trainers and validators are rewarded irrespective of the results of the training jobs. This is why in Figure 9, we see both trainers and validators creating a profit. The cost that the requester takes on are from the rewards it stakes into the protocol and no guarantee that the models trained are valid.

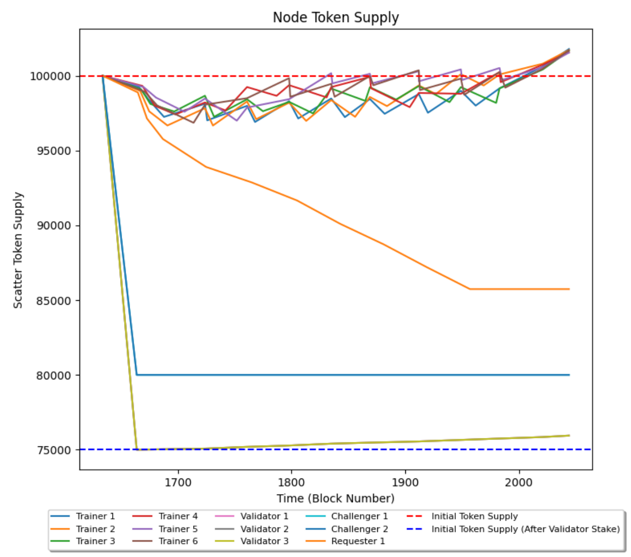

Figure 9: Node Token Supply With a Malicious Requester (Section 4.5)

5 DISCUSSION

From these evaluations, we can see that Scatter Protocol holds a lot of promise to approach decentralized federated learning. Specifically, we can see the effectiveness of the following innovations:

- **Dual-Layered Validation:** Dual layered validation systems are invaluable to the success of this protocol. With previous blockchain-based federated learning systems, validation stopped with a validator network [21]. While this is an improvement from single point of failure systems with centralized federated learning systems, the validation mechanisms still run risks like collusion as seen in section 4.4. Adding additional redundancy to the validation mechanisms via a second validation layer enables a layer of security that was previously unachievable while further decentralizing the validation process. By designing this second validation layer to where they are all competing for a lottery prize and with accountability functionality in this layer (i.e., challengers can challenge other challengers), we limit collusion or intentional malicious behavior that might be present in our validation layers.
- **Multi-Part Token Reward Function:** As mentioned in section 3.2, the trainer reward function is partially based on their stake in the training job and their performance. This reward function is important to the success of the protocol because it enables the protocol to give rewards based on a multitude of factors and scales these factors based on the importance. This is clearly seen in the Figure 3 where there are trainers that end the simulation with higher token counts despite completing the same training jobs. This is a direct result of performing better and/or having higher stakes in the corresponding training job. Without this reward function, there is a lack of incentive to stake tokens to the protocol or attempt to train a robust model (i.e., a trainer could just stake one token or perform poorly just to get the reward if the reward did not correspond directly to these factors)
- **Mixed Governance Mechanisms:** Throughout our validation mechanisms, we used a mixture of quorum voting (for validators) and super-majorities (through challengers). This provides us flexibility on assigning importance to specific events. For example, challengers require a super-majority because they are the last line of defense while validators require a quorum vote from a subset of validators because validating a single model does not require the entire computational effort of all validators on the network.

With this approach to federated learning, we also recognize some limitations:

- **Deterministic Events:** We currently rely on generalized evaluation and training jobs that may produce two different results when run in the same way. This may cause variations in the training job or evaluation job results.
- **Cost:** Currently our protocol makes hundreds of transactions across all nodes to the blockchain. When a transaction occurs, a transaction fee known as gas also occurs [8]. Depending on blockchain, this could cost a few dollars to hundreds of dollars per training job we execute.

6 CONCLUSION AND FUTURE WORK

We propose a protocol for decentralized federated learning that introduces novel approaches to incentivization, penalization, and validation. The results of our simulations show that our mechanisms are able to correctly identify malicious and benevolent nodes in a network and punish or reward them accordingly. Additionally, we demonstrate how each mechanism interacts with a diverse set of scenarios by evaluating this protocol in scenarios involving malicious requesters, trainers, and validators.

Future work on this protocol may have the goal of increasing the robustness of this system. This might be through exploring proof of authority and reputation mechanisms to choose validators, moving additional protocol logic outside of the blockchain to the client nodes to reduce transaction costs, or creating a unified machine learning runtime or usage of zero-knowledge proofs for more deterministic events.

REFERENCES

[1] Monik Raj Behera, Sudhir Upadhyay, and Suresh Shetty. 2021. Federated Learning using Smart Contracts on Blockchains, based on Reward Driven Approach. *CoRR* abs/2107.10243 (2021). arXiv:2107.10243 https://arxiv.org/abs/2107.10243

[2] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. 2023. Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges. *IEEE Communications Surveys Tutorials* (2023), 1–1. https://doi.org/10.1109/COMST.2023.3315746

[3] Christian Cachin and Marko Vukolic. 2017. Blockchain Consensus Protocols in the Wild. *CoRR* abs/1707.01873 (2017). arXiv:1707.01873 http://arxiv.org/abs/1707.01873

[4] Hang Chen, Syed Ali Asif, Jihong Park, Chien-Chung Shen, and Mehdi Bennis. 2021. Robust Blockchained Federated Learning with Model Validation and Proof-of-Stake Inspired Consensus. *CoRR* abs/2101.03300 (2021). arXiv:2101.03300 https://arxiv.org/abs/2101.03300

[5] Nanqing Dong, Jiahao Sun, Zhipeng Wang, Shuoying Zhang, and Shuhao Zheng. 2022. FLock: Defending Malicious Behaviors in Federated Learning with Blockchain. arXiv:2211.04344 [cs.CR]

[6] Nanqing Dong, Zhipeng Wang, Jiahao Sun, Michael Kampffmeyer, Yizhe Wen, Shuoying Zhang, William Knottenbelt, and Eric Xing. 2023. Defending Against Malicious Behaviors in Federated Learning with Blockchain. arXiv:2307.00543 [cs.LG]

[7] Benjamin Dowling, Paul Rösler, and Jörg Schwenk. 2019. Flexible Authenticated and Confidential Channel Establishment (fACCE): Analyzing the Noise Protocol Framework. Cryptology ePrint Archive, Paper 2019/436. https://eprint.iacr.org/2019/436 https://eprint.iacr.org/2019/436.

[8] Soroush Farokhnia. 2023. Lazy Contracts: Alleviating High Gas Costs by Secure and Trustless Off-chain Execution of Smart Contracts. arXiv:2309.11317 [cs.CR]

[9] Robin Fritsch, Marino Müller, and Roger Wattenhofer. 2022. Analyzing Voting Power in Decentralized Governance: Who controls DAOs? arXiv:2204.01176 [cs.CY]

[10] Jonathan Heiss, Elias Grünewald, Nikolas Haimerl, Stefan Schulte, and Stefan Tai. 2022. Advancing Blockchain-based Federated Learning through Verifiable Off-chain Computations. arXiv:2206.11641 [cs.CR]

[11] Yongrae Jo and Chanik Park. 2019. BlockLot: Blockchain based Verifiable Lottery. arXiv:1912.00642 [cs.DC]

[12] Shashank Joshi. 2021. Feasibility of Proof of Authority as a Consensus Protocol Model. arXiv:2109.02480 [cs.DC]

[13] Yujin Kwon, Jian Liu, Minjeong Kim, Dawn Song, and Yongdae Kim. 2019. Impossibility of Full Decentralization in Permissionless Blockchains. arXiv:1905.05158 [cs.CR]

[14] Steven P. Lalley and E. Glen Weyl. 2018. Quadratic Voting: How Mechanism Design Can Radicalize Democracy. *AEA Papers and Proceedings* 108 (May 2018), 33–37. https://doi.org/10.1257/pandp.20181002

[15] Chao Li, Runhua Xu, and Li Duan. 2023. Liquid Democracy in DPoS Blockchains. arXiv:2309.01090 [cs.CR]

[16] Adnan Ben Mansour, Gaia Carenini, Alexandre Duplessis, and David Naccache. 2022. Federated Learning Aggregation: New Robust Algorithms with Guarantees. arXiv:2205.10864 [stat.ML]

[17] Konstantin D. Pandl, Florian Leiser, Scott Thiebes, and Ali Sunyaev. 2023. Reward Systems for Trustworthy Medical Federated Learning. arXiv:2205.00470 [cs.LG]

[18] Yulin Sun, Qingming Qu, Chenxingyu Zhao, Arvind Krishnamurthy, Hong Chang, and Ying Xiong. 2023. TSoR: TCP Socket over RDMA Container Network for Cloud Native Computing. arXiv:2305.10621 [cs.NI]

[19] S. Vani, M. Doshi, A. Nanavati, and A. Kundu. 2022. Vulnerability Analysis of Smart Contracts. arXiv:2212.07387 [cs.CR]

[20] Zhibo Xing, Zijian Zhang, Meng Li, Jiamou Liu, Liehuang Zhu, Giovanni Russello, and Muhammad Rizwan Asghar. 2023. Zero-Knowledge Proof-based Practical Federated Learning on Blockchain. arXiv:2304.05590 [cs.CR]

[21] Haoxiang Yu, Hsiao-Yuan Chen, Sangsu Lee, Sriram Vishwanath, Xi Zheng, and Christine Julien. 2023. iDML: Incentivized Decentralized Machine Learning. arXiv:2304.05354 [cs.LG]

[22] Wanchuang Zhu, Benjamin Zi Hao Zhao, Simon Luo, and Ke Deng. 2021. MANDERA: Malicious Node Detection in Federated Learning via Ranking. *CoRR* abs/2110.11736 (2021). arXiv:2110.11736 https://arxiv.org/abs/2110.11736