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

1st Samrat Sahoo
Georgia Institute of Technology
Financial Services and Innovation Lab
Atlanta, USA
samratsahoo@gatech.edu

2nd Sudheer Chava
Georgia Institute of Technology
Financial Services and Innovation Lab
Atlanta, USA
chava@gatech.edu

***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 that is 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-layered validation mechanism 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.**

***Index Terms*—Blockchains, Distributed Systems, Federated Learning, Smart Contracts, Ethereum, Tokenization**

I. 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 and 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 can reduce some of the inefficiencies associated with centralized federated learning. More specifically, our contributions to alleviate issues with federated learning are as follows:

- We mitigate 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.

II. Related Works

Federated learning is currently an active area of research due to its benefits in privacy, distributed compute and data, and model aggregation. For this work, we look at four primary areas of related work (1) blockchains and smart contracts, (2) malicious node detection, (3) model validation, and (4) incentivization mechanisms.

A. 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 [1] 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 and Vukolic and Yu et al., who also conceptualized alternate decentralized, federated learning systems with a focus of incentivization [2][3].

B. 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 Dong et al. explores the idea of malicious node detection by utilizing a voting mechanism and proof-of-stake mechanisms as insurance against malicious nodes [4]. 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, work by Zhu et al. looks at deterministic malicious node detection by converting a 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) [5].

C. 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 [6]. 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. Work by Heiss et al. and Xing et al. both leverage zero knowledge proofs 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 [7][8]. They note there are some bottlenecks to be considered with the size and computational power required to create zero-knowledge proofs.

D. Incentivization Mechanisms

For decentralized protocols, we require forms of incentivization to ensure that actors are inclined to act benevolently and avoid malicious behaviors.

Research from Tu et al. explores forms of incentivization used in federated learning. One such reward function is self-report based contribution evaluation which the participants proactively report their contributions to the process [9]. This however requires trust and is therefore is found more in centralized federated learning schemes as opposed to decentralized schemes. Work by Zhu et al. looks at the Shapley Values function which fairly evaluates a participant's contribution to a coalition by looking at the impact their contribution has to the goal. However, the author notes there are limitations with utilizing Shapley Values for decentralized federated learning because it requires a trusted entity to properly calculate the value [10].

III. Protocol Design

A. 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.

- **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 in exchange for tokens.
- **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.
- **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.

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

1) Requesters interact with the blockchain by publishing a training and evaluation job for a specific topic (a 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.
4) Once a trainer has completed its training job, it encrypts the model data and publishes the models to the Interplanetary File System (a decentralized storage 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. The validators request a decryption key from the trainers to decrypt the model data before validating.
6) After validations have occurred, challengers can analyze node behavior and transactions 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 III-B)
8) The requester reads model data from the blockchain and aggregates them into one model. The responsibility of model aggregation and the aggregation algorithm of choice (i.e. FedAvg, FedSoftBetter, FedWorse, etc.) is left to the requester. [11].

B. Incentivization and Penalization

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:

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

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.

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. So, 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, $\sqrt{W^t_{st_i}} \cdot {W^t_{sc_i}}^2$, and dividing it by the sum of

all reward scores. The reward score composition ensures there is incentive for trainers to both stake tokens into a specific training job and train a robust 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 and ensures diminishing marginal returns for large stake amounts. We also take an achievement-based approach by squaring the average validation score to more heavily weigh trainer performance [12]. The second part of our reward function calculates the amount of the reward pool that should go to trainers (R_{tr}). We can express this 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. For our reward function, given a pool of validators, $V = \{v_1, v_2 \dots v_k\}$, 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 this 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:

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

This function discourages subsequent malicious behaviors for 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. 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.

TABLE I: 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

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

C. Smart Contract Architecture

Our protocol consists of seven different smart contracts that define the behavior of the system.

- **Training Job Token**: Allows requesters to publish and map ownership of each training job.
- **Evaluation Job Token**: Allows requesters to publish and map ownership of each evaluation job.
- **Model Token**: Allows trainers to publish the encrypted models and map ownership of each model.
- **Reputation Manager**: Manages the reputation of different nodes in the protocol and allows us to determine malevolent and benevolent nodes for each training job.
- **Vote Manager**: Manages votes, primarily for validation for validators and challengers.
- **Scatter Token**: Implements protocol-native tokens which are used for transactions to facilitate rewards and punishments.
- **Scatter Protocol**: A state-management proxy contract that directs requests to different contracts and manages the state of the protocol.

D. 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 using the Fisher-Yates shuffle algorithm. We use block timestamp, difficulty, and a random nonce to introduce stochasticity [13].

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 using 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 the 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 IV-D. Trainers on the other hand do not have this authority because trainers can inspect all training jobs before they commit to them.

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 a node, 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.

E. 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 that allow actions like creating a training job, joining a training job, and starting the training procedure for a specific training job, etc.
- **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.
- **Asynchronous Job Queue:** We anticipate trainer nodes to subscribe to multiple training jobs at once. To handle these jobs, we created an asynchronous job queue that executes training jobs from a job pool, allowing for better scalability.
- **Protocol Event Listeners:** The smart contracts emit various events that participants need to react to 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 save training job or evaluation job information to reference in later parts of the federated learning process.

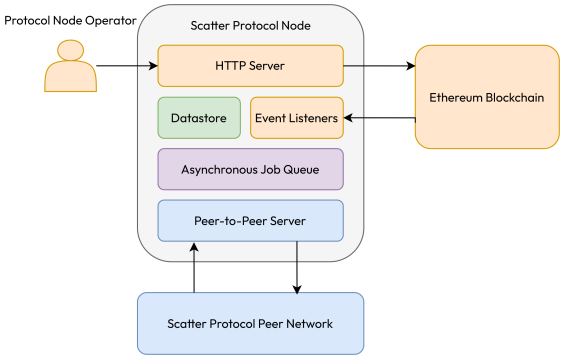


Fig. 1: Node Architecture of Scatter Protocol Node

Figure 1 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.

F. 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 III-B 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 [14].

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 [15].

We also implement encryption standards by encrypting models that are published to the blockchain using Advanced Encryption Standard-256 (AES-256). Due to asynchronous nature of the protocol, 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 could publish the other trainer's model without performing the computational work. To prevent this, we encrypt the model using AES encryption. When the validator is ready to evaluate a model, they request the decryption key before validating it.

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 [16]. This ensures that we can securely share the decryption key with other peers.

IV. Evaluation and Results

To evaluate our system, we focused on simulating different scenarios to determine how the tokenomics and protocol logic behaved. They are representative but not intended to be the exhaustive set of possibilities. To evaluate these, we run simulations on a custom built simulation software which allows us to run and control the actions of nodes. Our simulations used 12 nodes (one requester, six trainers, three validators, two challengers) to complete 10 training jobs. Each node had 100,000 scatter tokens staked. The number of malicious and benevolent nodes varies from simulation to simulation.

A. All Benevolent Nodes

Our first scenario is when all trainers and validators are benevolent. Figure 2 illustrates the token supplies for the trainers, validators, and challengers. We notice 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 III-B which is returned to them at the end of the training job. Because of the consistently benign behavior, trainers have a token supply above their 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 benevolently and hence there is no opportunity to win the lottery.

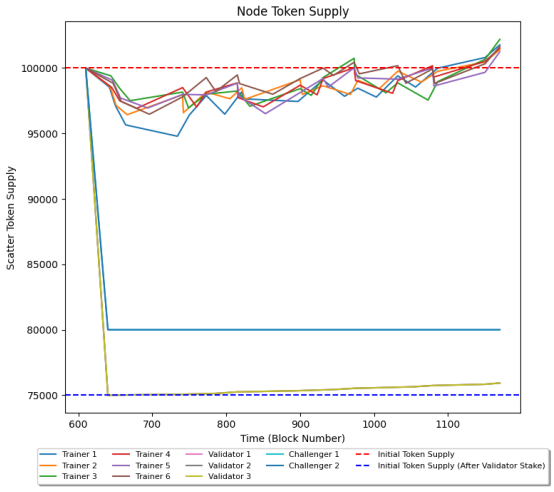


Fig. 2: Token Supplies for Benevolent Trainers and Validators (Section IV-A)

B. 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 3. 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 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).

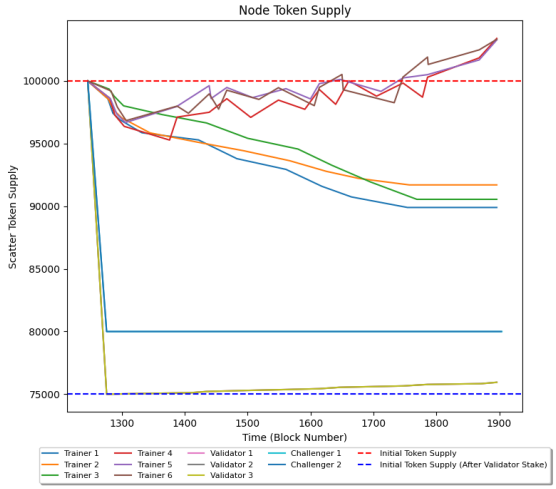
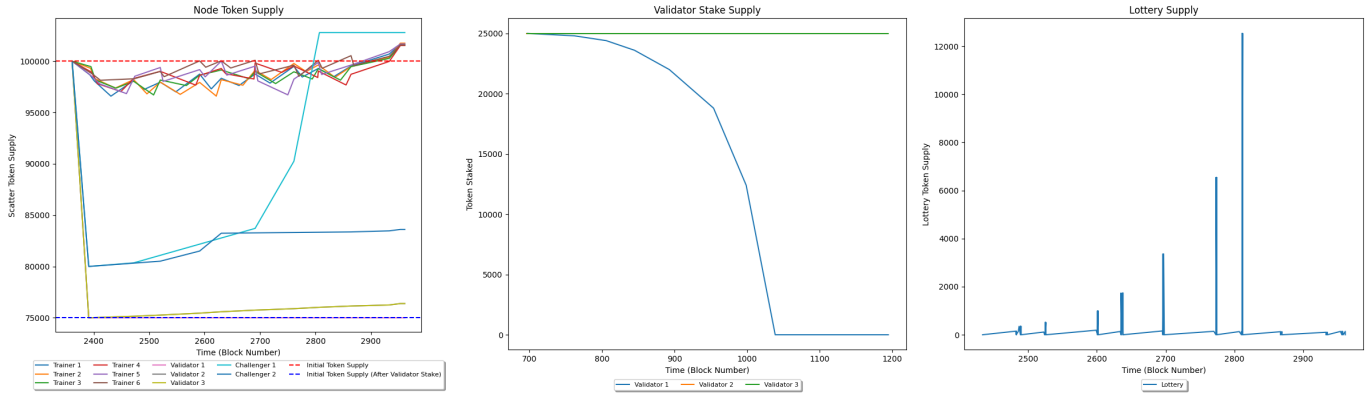


Fig. 3: Token Supplies for Partially Malicious Trainers and Benevolent Validators (Section IV-B)

C. 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 4a. 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 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



(a) Token Supply (b) Stake Amount (c) Lottery Supply

Fig. 4: Token graphs with partially malicious validators and benevolent trainers (Section IV-C).

the winner of the lottery, which is also demonstrated in Figure 4a. We also notice that the trainers continue to make a profit despite 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 the fraudulent validator has a constant token supply. This is because the punishment is taken out from the validator's protocol stake, not the node's token supply. Figure 4b demonstrates how this stake decreases exponentially with every subsequent penalty (the penalty function is defined in section III-B). Once the stake reaches zero, that validator loses privileges to evaluate models.

In Figure 4c, 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, leading to the spikes observed.

D. Malicious Trainers and Validators

The fourth scenario is when all trainers and validators are malicious. In figure 5 We notice that in this scenario that challengers are receiving the maximum profit. The large profits can primarily be 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. If the requester has 100 percent of validators and trainers fail, then it is refunded the reward pool minus the lottery tax.

V. Discussion

From these evaluations, we can see that our protocol holds a lot of promise with our approach to decentralized federated

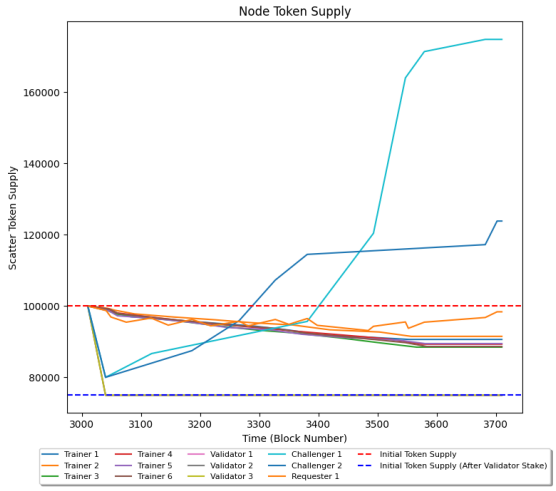


Fig. 5: Token Supply When All Trainer and Validator Nodes are Malicious (Section IV-D)

learning. Specifically, we can see the effectiveness of the following innovations:

- **Dual-Layered Validation:** While a single decentralized validation layer avoids single points of failure, the validation mechanisms still run risks like collusion as seen in section IV-D. 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 compete for a lottery prize along with self-accountability (i.e., challengers can challenge other challengers), we limit intentional malicious behavior that might be present in our validation layers.
- **Multi-Part Token Reward Function:** As mentioned in section III-B, 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 their importance. This is clearly seen in the Figure 2 where there are trainers that end the simulation with higher token counts despite completing the same training jobs due to performance and stakes. Without this reward function, there is a lack of incentive to stake tokens to the protocol or attempt to train a better model.

- **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 [17]. Depending on blockchain, this could cost a few dollars to hundreds of dollars per training job we execute.

VI. Conclusion and Future Work

We propose a protocol for decentralized federated learning that introduces novel approaches to incentivization, penalization, and validation. Through diverse simulations, show that our mechanisms are able to correctly identify malicious and benevolent nodes in a network and punish or reward them accordingly. 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, creating a unified machine learning runtime, or using of zero-knowledge proofs for more deterministic events.

References

[1] M. R. Behera, S. Upadhyay, and S. Shetty, "Federated learning using smart contracts on blockchains, based on reward driven approach," *CoRR*, vol. abs/2107.10243, 2021. [Online]. Available: https://arxiv.org/abs/2107.10243

[2] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," *CoRR*, vol. abs/1707.01873, 2017. [Online]. Available: http://arxiv.org/abs/1707.01873

[3] H. Yu, H.-Y. Chen, S. Lee, S. Vishwanath, X. Zheng, and C. Julien, "idml: Incentivized decentralized machine learning," 2023.

[4] N. Dong, J. Sun, Z. Wang, S. Zhang, and S. Zheng, "Flock: Defending malicious behaviors in federated learning with blockchain," 2022.

[5] W. Zhu, B. Z. H. Zhao, S. Luo, and K. Deng, "MANDERA: malicious node detection in federated learning via ranking," *CoRR*, vol. abs/2110.11736, 2021. [Online]. Available: https://arxiv.org/abs/2110.11736

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

[7] J. Heiss, E. Grünewald, N. Haimerl, S. Schulte, and S. Tai, "Advancing blockchain-based federated learning through verifiable off-chain computations," 2022.

[8] Z. Xing, Z. Zhang, M. Li, J. Liu, L. Zhu, G. Russello, and M. R. Asghar, "Zero-knowledge proof-based practical federated learning on blockchain," 2023.

[9] X. Tu, K. Zhu, N. C. Luong, D. Niyato, Y. Zhang, and J. Li, "Incentive mechanisms for federated learning: From economic and game theoretic perspective," *CoRR*, vol. abs/2111.11850, 2021. [Online]. Available: https://arxiv.org/abs/2111.11850

[10] H. Zhu, Z. Li, D. Zhong, C. Li, and Y. Yuan, "Shapley-value-based contribution evaluation in federated learning: A survey," in *2023 IEEE 3rd International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2023, pp. 1–5.

[11] A. B. Mansour, G. Carenini, A. Duplessis, and D. Naccache, "Federated learning aggregation: New robust algorithms with guarantees," 2022.

[12] K. D. Pandl, F. Leiser, S. Thiebes, and A. Sunyaev, "Reward systems for trustworthy medical federated learning," 2023.

[13] Y. Jo and C. Park, "Blocklot: Blockchain based verifiable lottery," 2019.

[14] S. Vani, M. Doshi, A. Nanavati, and A. Kundu, "Vulnerability analysis of smart contracts," 2022.

[15] Y. Sun, Q. Qu, C. Zhao, A. Krishnamurthy, H. Chang, and Y. Xiong, "Tsor: Tcp socket over rdma container network for cloud native computing," 2023.

[16] B. Dowling, P. Rösler, and J. Schwenk, "Flexible authenticated and confidential channel establishment (facce): Analyzing the noise protocol framework," Cryptology ePrint Archive, Paper 2019/436, 2019, https://eprint.iacr.org/2019/436. [Online]. Available: https://eprint.iacr.org/2019/436

[17] S. Farokhnia, "Lazy contracts: Alleviating high gas costs by secure and trustless off-chain execution of smart contracts," 2023.