Blockchain implementation of Federated Learning

Sreya Francis and Ismael Martinez

April 2019

Abstract— **Although Federated Learning allows for participants to contribute their local data without revealing said data, it faces issues in data security and in accurately paying participants for quality data contributions. In this report, we propose a blockchain design and architecture to establish data security, a novel validation error based metric upon which we qualify gradient uploads for payment, and implement a small example of our blockchain Federated Learning model to analyze its performance.**

Keywords— **blockchain, Federated Learning, distributed machine learning, class sampled validation error**

1 Introduction

In today's data market, users generate data in various forms including social media behaviour, purchasing patterns, and health care records, which is then collected by firms and used either for sale or for in-house data analytics and machine learning . As a result, each of us is essentially giving away a personal resource for no reward. In addition, these organizations have full access to our data, which can be a major invasion of privacy depending on the type of data collected. One proposed method of mitigating this issue of ownership and privacy when the purpose of the data is proprietary machine learning is *Federated Learning*, where an owner sends the training model to users who train on their local data and send back only the updated weights of the model. By doing this, a user never unveils the data to the owner, and keeps ownership of said data. A secondary result of this type of training is that users with sensitive data such as health care data are more likely to partake in the training, meaning the owner also receives more data to use for training.

There still remains the concern of handing out our data, a useful resource to organizational training models, for free. We propose the use of blockchain to facilitate the uploading and tracking of updates from users, as well as rewarding users for the data they used in computation. An additional benefit to using a blockchain is that it renders the updates immutable and thus secure. In all, we achieve privacy and security for users, as well as a potentially larger pool of data for organizations to use.

In § 2, we review the State-of-the-Art in Federated Learning and Distributed Machine Learning with Blockchain integration. In § 3, we propose a possible architecture for achieving Federated Learning with Blockchain. In § 4, we implement a version of our solution with assumptions using both Python and Hyperledger Fabric. In § 5, we look at future work in research and experimentation. Finally, we conclude the report in § 6.

2 Related Work

Federated Learning [1] [2] is an approach to Machine Learning whereby one entity $\mathcal{O}$ owns the training model T but not the data; the full dataset is distributed among many users in U. This model is trained in the following way:

1. $\mathcal{O}$ distributes the same model T to many users $u \in U$.
2. Each user u feeds their local data through the model T, and calculates their gradient update δ [3] to improve the model performance.
3. Each user u returns their update value δ to $\mathcal{O}$.
4. $\mathcal{O}$ receives the collection of update values δ and averages these values to a single value $\bar{\delta}$ [1].
5. $\mathcal{O}$ applies $\bar{\delta}$ to the current model T to obtain the new global model T', and $T \leftarrow T'$.
6. Steps 1-5 are repeated until $\mathcal{O}$ stops the Federated Learning process, at which point training is complete.

Though the original Federated Learning paper was published in 2016 [1], it can be viewed as a special case of *Distributed Machine Learning* (DML) which has been more widely studied [4] [5] [6] [7] [8] [9]. Specifically, it can be viewed as data parallelism across multiple user owned devices with local private data [5] [2]. The original concept of Federated Learning sought to allow users to keep ownership and privacy of their data during the model training process by only returning the δ update based on local data and not the local data itself [1] [2]. This approach can however lead to privacy breaches by analyzing the δ output of users [10] [11] [6]; a proposed solution to these privacy issues is to add noise to the updates at a negligible loss [11] [10] [6]. In the absence of blockchain security, DML techniques have been suggested to use homomorphic encryption to protect training data [12] [13]; however, Federated Learning's approach to only uploading δ updates ensures privacy, and the blockchain ensures security [1] [11].

One implementation of Federated Learning known as *BlockFL* uses blockchain to reward users for their local updates proportional to how many local data points are used [14]. The blockchain is meant to enhance privacy and security of local the update δ for the user, and validity of δ to the model [14]. Referring to Figure 1 [14], the reward system may not produce a lasting solution since many miners would be paying users "out-of-pocket". These rewards are proportional to the number of data points used for the update, and may be inflated by a malicious node; it is proposed for a miner to analyze computation time of users to combat this, though even that could be misrepresented [14]. Though the BlockFL implementation warns that two miners simultaneously creating a block may cause two separate blockchains, using the "Longest Chain" rule will mitigate this [15].

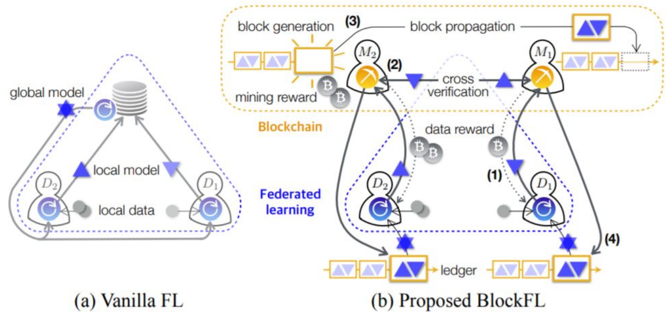

Figure 1: As described in the proposal for BlockFL [14], the architecture of BlockFL compared to "Vanilla" Federated Learning[2].

DeepChain looks into using blockchain for Distributed Deep Learning applications [12] as a means to keep both the data and the model private and secure. DeepChain proposes a incentive-based blockchain mechanism where both *parties* – those who upload data to the model, and *workers* – those who process the data and update the model, get paid an amount π_P and π_W based on their *contribution* ω_P and ω_W for parties and worker respectively. In addition, DeepChain proposes the use of a *penalty mechanism* whereby rewards to dishonest nodes are frozen and re-allocated. Although these two mechanisms works for a DML system where it is public how many data points each party is uploading, this cannot be applied to Federated Learning since only the update values δ are uploaded. Regarding a penalty mechanism, there is no need for one if Smart Contracts [16] are used. With a Smart Contract validating and administering the payment, a faulty transaction would fail and the payment would not be administered. The consensus protocol used by DeepChain is *blockwise-BA*, which elects a randomly chosen worker using cryptographic sortition [17] and a seed that changes with every block. Once a worker is elected, the worker creates a block which is verified by a selected committee before being added to the blockchain. This method relies on choosing an honest committee, and for the random algorithm to be negligibly close to perfectly random, both issues which may not be true in practice.

One blockchain implementation of machine learning sought to reward the user who could produce the best machine learning model for a publicly available dataset and evaluation function published by an organiser [18] in an architecture and process akin to Kaggle Data Science Competitions [19]. In this workflow, users would produce and train a machine learning model with the published dataset that would maximize the score given when the model is applied to the evaluation function; either the first model, the best model, or both would be rewarded by means of Smart Contracts. One issue addressed is that of a biased test/-train split by the organizer that could cause good models to not be rewarded due to poor performance on the chosen split; the proposed solution was to randomize the split in a fashion out of the organizer's control, however there is nothing stopping the organizer from adding new data to the system. One large problem that arises with this system is that all model evaluations are done on the blockchain. Although the issue of too many models being evaluated at once is addressed, the bigger issue of the large gas costs associated with the evaluation is still present; many users must each pay gas for their models to be evaluated, however only a small selected group is paid out. One consideration would be to have this computation and evaluation done off chain, with only the results returned and recorded in the blockchain.

Another study [20] looks into Distributed Deep Learning in order to take advantage of increased processing power and big data; it is proposed to use blockchain to create an incentive-compatible data market. Similarly, [13] attempts to make a DML system with user rewards based on Ethereum Smart Contracts [21]. When we are working with blockchains that are not fully public, it is proposed to be using a form of Access Control [13] [22] [9] [23]. One proposed method is to use *Attribute-Based Access Control* whenever working with blockchains [22][24] [25]. Another proposed form of Access Control in blockchain is through the use of Smart Contracts [22] [9] [23] [16] [26].

Computation of model updates is proposed in Federated Learning [1] [2] to either be distributed over a higher number of active smartphones, or to run complex computations on a smaller number of smartphones in idle states. Both of these methods are based on the assumption that a smartphone has the processing capabilities for these computations.

A number of sites have appeared with services around a decentralized data marketplace. The Ocean Protocol [27] is a blockchain service that describes itself as "a tokenized service layer that exposes data, storage, compute and algorithms for consumption". It is a data marketplace where users can sell or buy data in a secure, safe and transparent fashion, and was created with data sharing for AI in mind. Per their White Paper [28], this service leverages the Ethereum interface for Smart Contracts and token exchanges. Another such marketplace is Wibson [29] where in addition to transparency and anonimity, ensures users maintain control of the use of their data after it is sold. The price of data is dictated by the market, and Wibson utility tokens are rewarded to facilitate the use of the Wibson service on top of the Ether that's rewarded for the data; the system is built on Ethereum and the tokens are stated as ERC20 tokens [30]. The Datum Network [31] uses DAT tokens to be used within the blockchain network with these tokens available for purchase or sale from Ether [32]; there is however the problem of value, since the price at which a user is selling their data is unclear. The Datum Network does propose a functionally rich system where data is able to be queried and searched [32].

One key factor that isn't mentioned in many studies is the format of distributed data [12] [20] [13] [1] [2] [14]; we may not be able to assume homogeneous data with heterogeneous devices. In addition, a standard for the format of the uploaded updates δ is not well defined from the device to the miner [1] [2] [14].

Regarding validating and rewarding devices for their gradient uploads, one method is to compare the value of a given gradient δ against the mean of the whole set $\bar{\delta}$, and rewarding the $n - f$ closest neighbours to $\bar{\delta}$ where n is the total number of uploaded gradients and f is the number of faulty gradients. Known as Multi-Krum [33], this method was specifically designed to counter adversaries in Federated Learning, and has the top f contributions to the model that are furthest from the mean client contribution $\bar{\delta}$ removed from the aggregated gradient. Aside from only being a viable algorithm when $f < 3n$, it also requires all gradients to be collected for comparison with each other prior to determining the validity of a given δ, meaning we cannot immediately and independently decide to reward a user once a gradient δ is uploaded. Furthermore, it assumes knowledge of the number of faulty neighbours f when choosing to reject the f further neighbours of the mean $\bar{\delta}$ from our aggregation.

3 Proposed Design and Architecture

Taking the related work into consideration, we choose to make adjustments to improve privacy, access control and storage; the architecture and workflow of the system are shown in Figure 2 and 3.

The following assumptions are made regarding the devices and data in the system.

- A smartphone device has enough storage to store the kth model.
- A smartphone device does not necessarily have enough extra storage to store the entire blockchain.
- The training data for the model is homogeneous across different devices.

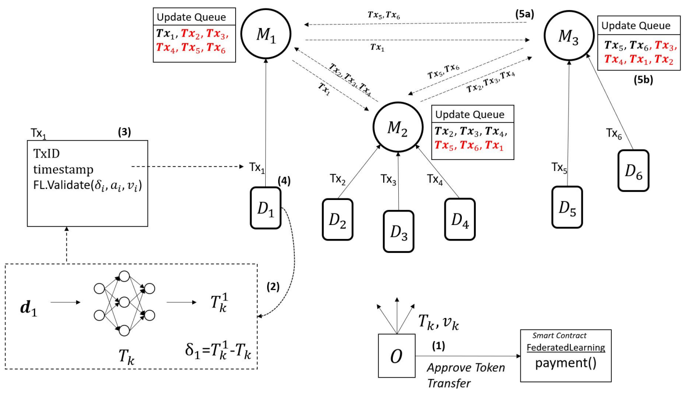

Figure 2: The workflow of round k begins with (1) $\mathcal{O}$ distributing T_k andd the version v_k to all devices, and simultaneously approving token transfer to the Payment() Smart Contract; (2) each device D_i uses a local dataset of size n_i to calculate a gradient δ_i [3] [1]; (3) the values δ_i, n_i – the number of data points, a_i – the address of D_i, and v_i – the version of the model used, are packed into a Transaction, along with a TxID, timestamp and a Smart Contract function call to `UploadGradient()`; (4) each device D_i sends its transaction to its closest miner M_j; (5a) each miner M_j who received Tx_i validates the transaction and adds it to its queue, while simultaneously broadcasting the received transactions to all other miners and (5b) the miners who receive the remaining transactions have their final queue of transactions for training model T_k.

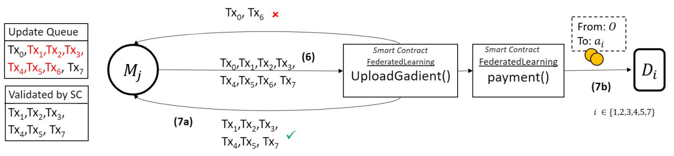

Figure 3: The workflow continues with (6) miner M_j running each transaction in its queue, which involves a call to `UploadGradient()` where the format of δ_i, the correct version number v_i, and the existence of the address a_i are all checked; (7a) the transactions that run without errors are added to the miners queue for the next block and (7b) the devices who sent in the transaction are rewarded an amount of tokens proportional to n_i – the number of datapoints used for training.

3.1 System and Blockchain Architecture

For our system design, we are using Permissioned Blockchain which gives the owner $\mathcal{O}$ more control over who participates in the training process. This also gives the owner $\mathcal{O}$ full liability of payment for the device and miner work, as opposed to devices D needing to pay for their transactions [18], or miners to reward devices out-of-pocket [14]. Although there may be a large number of devices uploading their δ values, the number of miners is restricted to however many the owner $\mathcal{O}$ designates the network needs; in practice the number of miners should be far less than the number of devices since the transactions are infrequent, only arriving once per device per training round at most.

Our base implementation of Federated Learning is built with smartphone devices in mind who act as the users performing the training and sending in δ values. We have a model owner $\mathcal{O}$ who defines the initial model and distributes the reward.

We define our system transaction to carry the information needed for our Federated Learning process. Each transaction contains the parameters shown in Table 1, with $a_i, H(\delta_i), v_i$ and n_i part of a call to the Smart Contract `UploadGradient()`. Each user has a public key and a private key pair which they use to sign transactions.

Parameter	Purpose	Size	Section
TxID	Unique identifier of the transaction	256 bits	
a_i	Address of device D_i	160 bits	
$H(\delta_i)$	SHA256 Hash of binary representation of training model weight updates	~256 bits	§ 3.2
n_i	The data cost – number of data points used to calculate the model update	16 bits	§ 3.2
v_i	The current version k of T_k	16 bits	§ 3.5
timestamp	Timestamp of the transaction	64 bits	
v, r, s	Signature of hash of the transaction	65 bits	

Table 1: Parameters required in a transaction upload from device D_i.

The *version* of the model being used is the integer value k of the current Global Model T_k. If the uploaded version v_i does not match the current version k, either the update value δ_i needs to be adjusted, or the value δ_i should be dropped.

Similar to Bitcoin, the proposed architecture is to use a maximum fixed block size [34]. This is based on the proposed transaction format having little variance in size, meaning each block would have roughly the same number of transactions. For our implementation, we propose a maximum size of 1 MB, including the blockchain header. Regardless of what data we're using, the value of δ will be quite large. For example, the single channel MNIST image data [35] is 1 MB per update δ; as a result, this value could decrease for non-image data, or increase for larger image data. The way we store record of these values within the blockchain is to store signed transactions in a table off-chain, and record only the hash of the value δ on-chain. This process is shown in Figure 4. This means that when the Smart Contract validates the format of δ, it must use an oracle to access the value in the table off-chain. When the owner updates the model, it must grab the gradients from the same place.

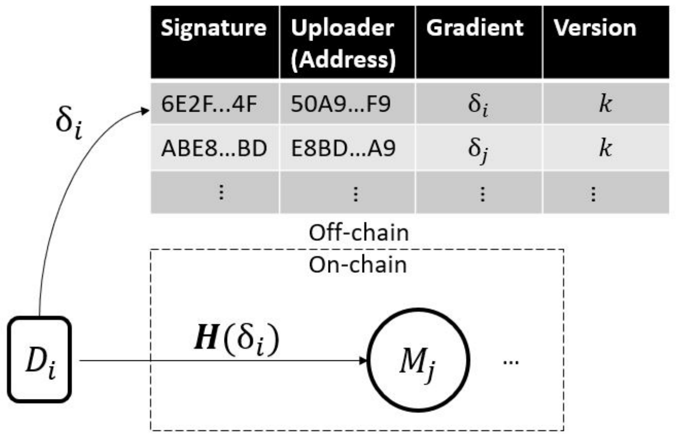

Figure 4: A device uploads the gradient value to an off-chain table where it is later accessed by the Smart Contract for validation, and the owner for gradient aggregation.

The *data cost* $|\boldsymbol{d_i}| = n_i$ of D_i is the number of datapoints used for training the model T_k to obtain δ_i. The amount rewarded to D_i for its gradient δ_i is proportional to the data cost n_i.

3.2 System Design and Workflow

The kth model is calculated from applying all of the model updates currently in the blockchain up to the kth block. We define $\mathcal{D}$ to be the set of all devices and $\mathcal{M}$ to be the set of all miners. As in Figure 2, step (1), each device $D_i \in \mathcal{D}$ has a copy of T_k given to it by O, along with the current version v_k; this is defined as *round k*. We walk-through the process of training the next model, validating the transactions and paying the users, referring to Figure 2 and 3.

For each device $D_i \in \mathcal{D}$ upon receiving the kth model T_k, (2) D_i trains the model T_k off-chain using it's local data $\boldsymbol{d_i}$ of size n_i to get an updated model T_k^i. The gradient is then calculated as $\delta_i = T_k^i - T_k$. (3) The values of δ_i, the size of $\boldsymbol{d_i} = n_i$, the device address a_i and the version v_k are set as parameters to the Smart Contract `UploadGradient()` together with a TxID, a timestamp and a digital signature. (4) The device D_i sends this transaction on-chain to its nearest miner M_j. (5a) Each miner $M_j \in \mathcal{M}$ adds the transactions received by the devices to their transaction queue, and simultaneously broadcasts the transactions to other miners; (5b) each miner receives the remaining transactions from other miners and add them to the final queue of transactions to be verified for Block k. (6) Each miner executes each transaction in its queue, which involves a call to `UploadGradient()` where details about each transaction are validated, such as the correct format for δ_i, the correct version v_i, and that the address a_i exists. (7a) Once validated, this transaction is added to the miner's pending queue for the next block; (7b) the device $D_i \in \mathcal{D}$ who submitted a valid transaction is rewarded an amount proportional to the data cost n_i via the submitted address a_i.

3.3 Consensus

The consensus protocol we elect to use is that of *Byzantine Fault Tolerance* (BFT) [36] [37]. This protocol has a set of *endorsement peers* who each run a transaction, and each output whether a transaction is valid or not. The BFT endorsement protocol requires $3f < n$, where f is the number of faulty nodes and n is the total number of nodes taking part in BFT [37]. In other words, we need more than 2/3 of the endorsement peers to agree on the validity of the transaction in order for consensus to be reached. Once consensus is determined for each transaction, the next Block k is created.

3.4 Initial Model

The *initial model* T_0 has all weights initialized to normally distributed values with mean 0 and variance 1; therefore, we define each weight $w \sim \mathcal{N}(0,1)$. This initial model, defined as model-0, is seen as a black box to all users except O; we do not allow for any device or miner to view the model. In order to both keep the model from unaltered visibility from devices while allowing devices to train on the model, we employ homomorphic encryption to the model via the Paillier Cryptosystem which is commonly used in distributed deep learning [12]. This symmetric cryptographic scheme has the property such that $E(x+y) = E(x) \cdot E(y)$, where $E(x)$ is the encryption of x. Since for a given weight in our training model we ultimately want to set $w \leftarrow w + \eta \cdot \bar{\delta}$ [5], we can achieve this by setting $E(w) \leftarrow E(w) \cdot E(\eta \cdot \bar{\delta})$.

3.5 Global Model

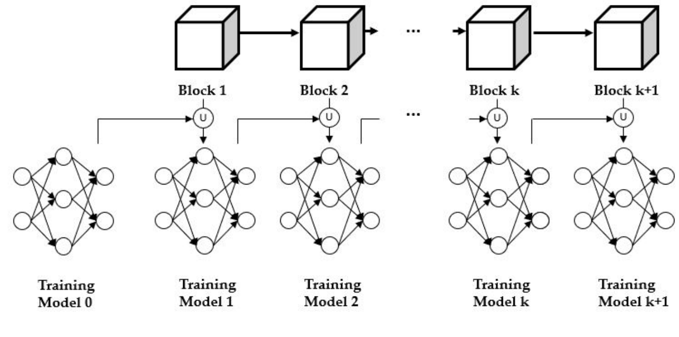

Figure 5: Model T_{k+1} is calculated from applying the gradient values δ_i in Block $k+1$ to previous model T_k.

To calculate model-1, a miner applies the aggregate of all δ_i updates from the 1st block; we denote the number of updates in block 1 by b_1. We define w_l as being the lth weight in T_0, and $\delta_{i,l}$ as the lth weight of the ith gradient value; the training model will apply the update

$$w_l \leftarrow w_l + \eta \cdot \bar{\delta}_l = w_l + \eta \cdot \frac{1}{b_1} \sum_{i=1,\ldots,b_1} \delta_{i,l}$$

to each $w_l \in T_0$, where η is the *learning rate* [5].

Similarly, to calculate model-$(k+1)$, the owner $\mathcal{O}$ applies the update

$$w_l \leftarrow w_l + \eta \cdot \bar{\delta}_l = w_l + \eta \cdot \frac{1}{b_{k+1}} \sum_{i=1,\ldots,b_{k+1}} \delta_{i,l}$$

to each $w_l \in T_k$. If there are currently K blocks in the blockchain, then model-K is the *Global Model*; this process is shown in in Figure 5.

3.6 Data Validity and Quality

The validation check we have defined earlier requires miners to trust that the data cost value a device claims to have used is correct. As noted in § 2, Multi-Krum won't work for our purpose, so we propose a new concept of tailoring a validation set to a device's data breakdown which we will call a *Class-Sampled Validation Error Scheme*.

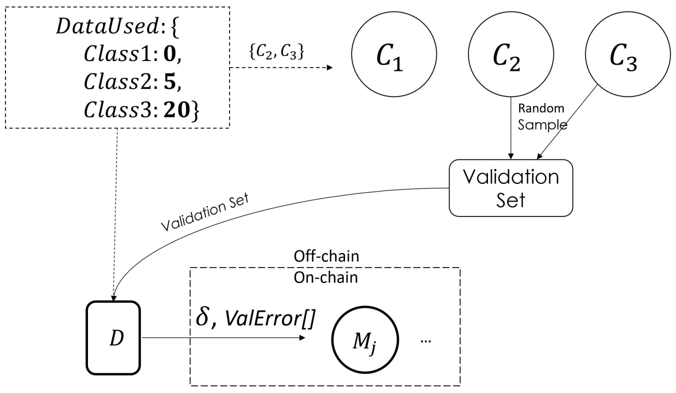

Figure 6: Prior to training, a device $D \in \mathcal{D}$ sends a list of the classes for which it has data, and receives a validation set containing data from only those classes. Once the validation set is received, training proceeds and the set of validation errors throughout training is sent along with the gradient δ.

This proposed method, shown in Figure 6, begins prior to training. We define the set of all classes available as $C = \{C_1, C_2, ..., C_p\}$ for p classes. For a device $D \in \mathcal{D}$, we define the set C_D as the set of all classes for which D has data. Then, $C_D \subseteq C$ because there may exist a class $C_i \in C$ such that for all local datapoints $d \in \boldsymbol{d}$, $d \notin C_i$. D sends the set

C_D to $\mathcal{O}$ and receives a validation set with datapoints chosen only from the classes in C_D. Once received, D begins to train model T_k with the local training dataset $\boldsymbol{d}$ and the received validation set. During training, the model will intermittently apply the validation set to the training model and record the validation error; if the model is improving, we expect the validation error to decrease. At the end of training, D will send the validation errors alongside other parameters for the function call `UploadGradient()`. We modify this function to look at the general trend of the validation errors over training time; if the validation errors are decreasing, we say δ is a valuable and valid gradient update, and we reward D based on its data cost n.

This algorithm does not require any trust of the devices who can inflate their data cost n for a higher reward; however, it is possible for either a faulty gradient δ to appear valuable and thus be rewarded, or for a valid gradient δ to have an increasing validation error if the datapoints used by D don't reflect the data in the received validation set and thus not be rewarded.

The most obvious issue with this Class-Sampled Validation Error Scheme is that it is only defined here for classification problems; it would be useful to find other similar schemes for tailoring validation sets based on non-classification data. This scheme has also not been implemented and tested, so it is still to be seen whether it is possible to filter out "garbage" data that does not pertain to the model at all, or conversely how accurate this scheme is at identifying valuable gradients.

3.7 Smart Contracts

We propose the creation and usage of two Smart Contracts in our system.

`UploadGradient` – This Smart Contract verifies that every parameter as described in Table 1 is present in the transaction. It compares the hash of the transaction update value δ_i with the same value on the off-chain record as per Figure 4, ensures that the hashes are equal, that the true value of δ_i follows the required size and format requirements for T_k as set by $\mathcal{O}$, and verifies the submitted version v_i is the same value as the last version value v_k sent out by $\mathcal{O}$ (§ 3.1). Once a transaction is validated as being a proper update transaction, the Smart Contract returns `true`.

`Payment` – Once a transaction from device $D_i \in \mathcal{D}$ is validated, this Smart Contract is triggered to send payment proportional to the data cost n_i to address a_i. Since we are using Hyperledger, this payment would take the form of a token of which $\mathcal{O}$ has given prior approval to the Smart Contract to transfer to devices.

3.8 Computation and Storage

A device $D_i \in \mathcal{D}$ must calculate it's update value δ_i locally since we assume we cannot use any miner $M_j \in \mathcal{M}$ to aid in computation since sending local data would break the data privacy benefits of Federated Learning. In order to avoid the need for Mobile Edge Computing [38], it is assumed that a device $D_i \in \mathcal{D}$ has enough processing power and memory to calculate an updated model T_k^i from model T_k; this may not be realistic if the size of T_k becomes much larger than what we had considered in § 3.1. It is not assumed that D_i can store the entire blockchain in-memory – only the block headers are useful to D_i along with the Merkle Path to the reward transaction of D_i.

3.9 Permissioned Blockchain – Restrictions

This blockchain system is partially private, since all block data needs to be visible to all those that partake in the blockchain, but to nobody outside the system. We define a *member* x of the blockchain as either a device $x \in \mathcal{D}$, a miner $x \in \mathcal{M}$ or the model owner $x = \mathcal{O}$. The only restriction we put is that $\mathcal{O} \cap \mathcal{D} = \varnothing$, meaning $\mathcal{O}$ doesn't take part in the Federated Learning training process and none of the devices take part in the model aggregation process.

To achieve this, we are using a *permissioned blockchain* [26] for this task, and specifically we'll be working with Hyperledger Fabric [26] which has *permissions* – a ruleset which defines which members have access to which actions. In this ruleset, we define the owner $\mathcal{O}$ as not being able to send any transactions that call the `UploadGradient()` Smart Contract, thus impeding it from contributing its own data to the process. Furthermore, we define a rule to limit a device's only action as sending in a transaction with a call to `UploadGradient()`. At any time, the owner $\mathcal{O}$ can amend this ruleset in order to complete the training process by stopping all devices from uploading gradients.

We also want to restrict each device from uploading more than one gradient to the model for each version v_k. We can do this by adding a nonce to each device transaction in order to both keep track of who has uploaded and how many times they have uploaded; we can then define the ruleset to only allow this nonce value to increase at most once per round k. Another method which takes advantage of Hyperledger Fabric *assets* (§ 4.1) is to define the unique ID of created assets to be a function of the id of D_i and the uploaded version number v_i; a Smart Contract attempting to create an asset with the same ID would then fail. In practice, we want a device to upload the gradient obtained from their best dataset; due to our restriction, it is suggested that devices are confident with their off-chain dataset and training process before uploading their one and only transaction for round k.

In addition to having a ruleset to restrict/allow actions from members of the network, a permissioned blockchain gives the owner $\mathcal{O}$ full control over who joins the network in the first place, and can revoke access to joined devices at any time.

4 Proof of Concept

We made a small implementation of our design in order to test out the general workflow in practice. For this implementation, we used 15 training rounds of 10 local `Device` participants who performed the model training off-chain in Python, and sent transactions to the Hyperledger Fabric blockchain.

The Proof of Concept seeks to answer whether a Hyperledger Fabric blockchain could work with Federated Learning implementation in Python; since we're using a REST API to interact with Hyperledger Fabric, then any programming language that supports API calls can interact with our blockchain system.

```
namespace org.martinezfrancis.federatedlearning

participant Device identified by deviceId{
 o String deviceId
}

asset Gradient identified by deviceVersionId{
  --> Device device
  o String deviceVersionId
  o String hash
  o Integer version
  o Integer dataCost
}

asset TrainingModel identified by modelId{
  o String modelId
  o Integer version
}

asset Token identified by deviceVersionId{
  o String deviceVersionId
  o Integer value
  o Integer version
  --> Device device
}

transaction UploadGradient{
  --> Device device
  --> TrainingModel trainingmodel
  o String hash
  o Integer dataCost
  o Integer version
}
```

Listing 1: `model.cto` – Participant, Asset and Transaction definition.

4.1 Hyperledger Fabric - REST API

For this implementation on Hyperledger Fabric we made use of Hyperledger Fabric Composer where we defined the files `model.cto` – where we define the *participants*, *assets* and *transactions*, `logic.js` – where we define the Smart Contract chain code, and `permissions.acl` – where we define the ruleset restricting/allowing the actions of the participants.

The `model.cto` script definition is shown in Listing 1. We can have multiple `Device` participants, which make up the users in our Federated Learning process. Although training of the model occurs off-chain, we have a single instance of a `TrainingModel` asset which we use to keep track for which *version* we are currently uploading gradient values. A `Gradient` asset is linked to a `Device` participant, and has the *hash* of the gradient as in Figure 4, the current training *version*, and the *dataCost* claimed by the *Device* participant. The `Token` asset is also linked to a `Device` participant, has the current training *version*, and a *value* which is equal to the claimed *dataCost*. The `UploadGradient()` transaction is the parameter description of our Smart Contract, requires a `Device` participant instance and the `TrainingModel` asset instance, and has the *hash*, *dataCost*, and *version* parameters which we've seen in the other assets.

The `logic.js` file is the *chaincode* of our application which is HyperLedger Fabric's version of a Smart Contract written in pure JavaScript. In this function, we have decided to combine the `Payment()` functionality within the same `UploadGradient()` function – the script validates the submitted parameters, creates the necessary assets, and rewards the user for their upload. The script follows these steps:

`UploadGradient`(**`tx`**.{`Device`, `TrainingModel`, *version*, *hash*, *dataCost*})

1. *Validation*
   (a) Verify that the uploaded **`tx`**.*version* is equal to the current **`tx`**.`TrainingModel`.*version* attribute.
2. *Create new Gradient*
   (a) Create a blank asset of type `Gradient` with unique id <**`tx`**.`Device`.*deviceId*_**`tx`**.*version*>.
   (b) `Gradient`.*device* ← **`tx`**.`Device`
   (c) `Gradient`.*version* ← **`tx`**.*version*
   (d) `Gradient`.*hash* ← **`tx`**.*hash*
3. *Pay user via a Token*
   (a) Create a blank asset of type `Token` with unique id **`tx`**.`Device`.*`deviceId`*_**`tx`**.*`version`*.
   (b) `Token`.*value* ← **`tx`**.*dataCost*
   (c) `Token`.`Device` ← **`tx`**.`Device`

Within Hyperledger Fabric, the *deviceId* of a `Device` is used in the same way as the *address* a_i from our design. We leave it for future work to both implement and test the Class-Sample Validation Error Scheme and to use an Oracle to check the off-chain format of the gradient δ, both as part of the *Validation* phase.

Our `permission.acl` file defines the *allowance* of participants of which actions they can perform. We set the following rules:

- Devices have no access to create or modify `Gradient` assets unless submitting a transaction to `UploadGradient()`
- Devices have no access to create or modify `Token` assets unless submitting a transaction to `UploadGradient()`
- All participants have READ access to the all `Gradient` assets.

This ruleset ensures that participants only see the information they need to see, which are at most all the gradients being uploaded; they do not need to see the rewarded tokens.

To run and interact with the blockchain locally, we deployed the blockchain with a local REST API. Then, we could perform off-chain training and data storage with Python while sending calls to the API to read the blockchain data, getting the list of active participants for whom we need to train, and posting transactions with calls to `UploadGradient()`.

4.2 Python Implementation of Federated Learning

For the implementation of Federated Learning, we made use of Python and Tensorflow framework through the following scripts.

`federated-data-extractor.py` – Take the full dataset and split it among the n client devices.

`federatedlearning.py` – Build, train, and evaluate a training model.

`clients.py` – For a client device, perform training on the local dataset and save the gradients obtained. Upon calculating the gradient, a call is made to Hyperledger Fabric where the gradient is recorded and the client is rewarded proportional to the data cost. This script is executed for many local device instances to simulate a federation of individual devices.

`offchain-database.py` – For each of the saved client gradients, create an off-chain database available to the Owner to facilitate in model aggregation. The database has the column labels `DeviceId`, `DataCost`, `DeviceGradientPath`, and `ModelVersion`.

`globalmodel.py` – All gradients whose paths are in the off-chain database and pertaining to a specific model version are aggregated and added to the current model to make the next global model using the federated averaging algorithm. This new model is recorded in Hyperledger Fabric.

`BlockchainFederatedLearning.py` – This script ties everything together; for a given number of clients and versions. It calls `federated-data-extractor.py` to split the dataset, loops through calls to `clients.py` and `globalmodel.py`, creates an off-chain database of gradient paths with `offchain-database.py`, before printing the accuracy of the global models in order along with a graph of their accuracy values.

4.3 Implementation Worflow

We have Python and Hyperledger Fabric working together to achieve Federated Learning using the following workflow.

1. (a) Create the Initial Model in Python as per § 3.4.
   (b) Create the `TrainingModel` asset in Hyperledger Fabric with *version*: 0.
2. (a) Create D local devices in Python, each with a unique id.
   (b) Create D `Device` participants with the same id values as in Python.
3. For version $v = k, k \in \{0, ..., V\}$, perform the Federated Learning Process:

```
for $D \in \mathcal{D}$ do
    $T'_k \leftarrow \texttt{train}(T_k)$
    $\delta \leftarrow T'_k - T_k$
    · Write $\delta$ off-chain with version and dataCost
    · Upload $\delta$, version and dataCost to Hyperledger Fabric, creating a Gradient asset and a Token asset
```

4. Average all gradients with *version* $v = k$ to obtain $\bar{\delta}_k$.
5. Apply the federated aggregation on T_k to obtain

$$T_{k+1} \leftarrow T_k + \eta \cdot \bar{\delta}_k$$

   where $\eta \in (0, 1]$ is the *aggregation factor.*

6. Increment `TrainingModel`.*version* $v \leftarrow k + 1$ on Hyperledger Fabric.

4.4 Results

For $n = 10$ clients each with an equal split of the MNIST dataset, and running 15 rounds, we obtain the following accuracy metrics of the Global Model. .

Global Model Version	Accuracy
v.0	0.10949999839067459
v.1	0.3617999851703644
v.2	0.620199978351593
v.3	0.6942999958992004
v.4	0.7509999871253967
v.5	0.767300009727478
v.6	0.7742999792098999
v.7	0.7774999737739563
v.8	0.7760999798774719
v.9	0.7821999788284302
v.10	0.7785999774932861
v.11	0.7736999988555908
v.12	0.777400016784668
v.13	0.7838000059127808
v.14	0.7935000061988831
v.15	0.7986000180244446

Table 2: Results of training 10 devices for 15 rounds.

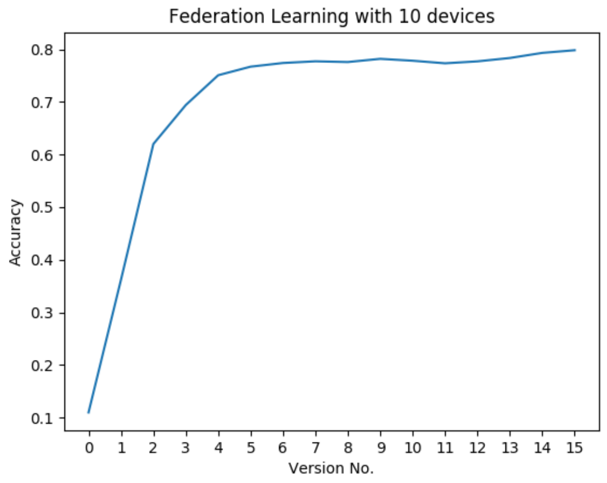

Figure 7: Graphical results of training accuracy of 10 devices over 15 rounds.

We can see from Figure 7, Figure 8 and the evaluation values in Table 2 that the model improves but quickly plateau with small dips in accuracy without deviating significantly from a centralized approach. This confirms that the blockchain does not interfere with the Federated Learning process, while recording and rewarding devices for their data in Hyperledger Fabric which we can easily view from the browser's REST API dashboard. This plateau could be due to the lack of diversity of the devices per round, or the lack of diversity in the datasets of each device per round. Ideally, the Federated Learning training model would see new data every round; this type of experimentation is left for future work. Another reason may be that the number of steps taken by the training model is not optimal; we attempted to make every device train the same way, fixing the number of training iterations to do so. The small dips in accuracy could be due to over-fitting on the part of each device as the models become more accurate. For these two reasons, more optimal methods of enforcing a standard training process to avoid over-fitting and to reach training optimization are left for future work [39].

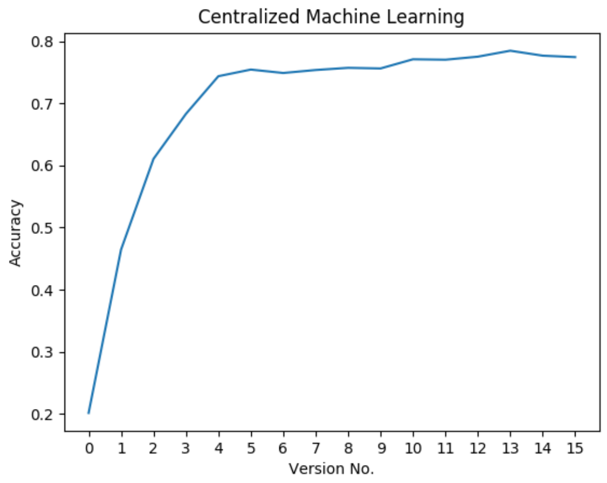

Figure 8: Graphical results of training accuracy of centralized dataset over 15 rounds.

5 Future Work

Many different adaptations, tests, and experiments have been left for the future due to lack of time. Future work concerns deeper analysis of particular mechanisms, new methods to try or propose, or simple curiosity.

The following is a list of topics we'd like to explore further within our current framework:

- Diversification of data per round to increase training accuracy.
- Experimentation with standardized training across all devices.
- Implementation and testing of the Class-Sampled Validation Error Scheme.
- Further expansion of current implementation, including checking off-chain gradient format using an Oracle during the Validation phase.
- Device rewards based on Model Performance.
- Automated termination of training before it over-fits the data. This also extends to having a training standard or automated training process so all devices train the model in the same manner.
- Encryption of training model shared with users using Pailler Cryptosystem [12].

- Modify the system to train and aggregate with non-iid data, i.e. imbalanced classes. One approach is to globally share a small subset of data from each class to every device to improve training accuracy [40].
- Enhancement of adversarial robustness to data poisoning attacks on Federated Learning [41].
- Experimentation with neural network compression schemes in conjunction with Federated Learning to reduce the communication overload. [42].
- Implementation of this system within spaces of Health Care where the data gains are limited by data privacy and security.

6 Conclusion

In this proposal, we addressed the problems of data privacy, security, and fair reward in distributed machine learning using blockchain and Federated Learning. An in-depth workflow was presented for scalable recording and rewarding of gradients using a combination of blockchain and off-chain databases of records. We have also proposed a Class-Sampled Validation-Error Scheme to validate and verify gradients to determine a reasonable device reward. We implemented a Proof of Concept with a small set of clients and rounds to demonstrate that the blockchain does not interfere with the federated learning aggregation, while limiting the number of uploads and validating the claimed data cost per device. Finally, we composed a list of aspects of Federated Learning and Blockchain that require more in-depth study for implementation as part of future work. With the proposed system, individuals benefit by retaining ownership and receiving incentives for their data, and model owners benefit from access to a larger and more diverse set of client data, leading to more robust and higher performing Machine Learning models.

References

[1] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[2] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. *Google Research Blog*, 2017.

[3] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[4] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11, 2013.

[5] TORSTEN HOEFLER TAL BEN-NUN. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. 2018.

[6] Jha S. Ristenpart T Fredrikson, M. Model inversion attacks that exploit confidence information and basic countermeasures. pages pp. 1322–1333, 2015.

[7] Peter Richtarik Barnabas Poczos Alex Smola Sashank J Reddi, Jakub Konecny. Aide: Fast and commu- nication efficient distributed optimization. 2016.

[8] Emiliano De Cristofaro Vitaly Shmatikov Luca Melis, Congzheng Song. Exploiting unintended feature leakage in collaborative learning. 2018.

[9] Damiano Di Francesco Maesa, Laura Ricci, and Paolo Mori. Distributed access control through blockchain technology. *Blockchain Engineering*, page 31, 2017.

[10] Moin Nabi Robin C. Geyer, Tassilo Klein. Differentially private federated learning: A client level perspective. 2017.

[11] Ramage D. Talwar K. Zhang H. Brendan McMahan, H.B. Learning differentially private language models without losing accuracy. 2017.

[12] J Weng, Jian Weng, J Zhang, M Li, Y Zhang, and W Luo. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *Cryptology ePrint Archive, Report 2018/679*, 2018.

[13] Decentralized Machine Learning. Decentralized machine learning white paper, cited March 2019. URL `https://decentralizedml.com`.

[14] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. On-device federated learning via blockchain and its latency analysis. *arXiv preprint arXiv:1808.03949*, 2018.

[15] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[16] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

[17] Silvio Micali. Algorand: the efficient and democratic ledger. *arXiv preprint arXiv:1607.01341*, 2016.

[18] A Besir Kurtulmus and Kenny Daniel. Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain. *arXiv preprint arXiv:1802.10185*, 2018.

[19] Kaggle: Your home for data science – competitions. `https://www.kaggle.com/competitions`. Accessed: 2019-03-25.

[20] Gihan J Mendis, Moein Sabounchi, Jin Wei, and Rigoberto Roche. Blockchain as a service: An autonomous, privacy preserving, decentralized architecture for deep learning. *arXiv preprint arXiv:1807.02515*, 2018.

[21] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[22] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. Blockchain based access control. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 206–220. Springer, 2017.

[23] Uchi Ugobame Uchibeke, Sara Hosseinzadeh Kassani, Kevin A Schneider, and Ralph Deters. Blockchain access control ecosystem for big data security. *arXiv preprint arXiv:1810.04607*, 2018.

[24] Hamza Es-Samaali, Aissam Outchakoucht, and Jean Philippe Leroy. A blockchain-based access control for big data. *International Journal of Computer Networks and Communications Security*, 5(7):137, 2017.

[25] Junbeom Hur and Dong Kun Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2011.

[26] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, 2016.

[27] The ocean protocol. `https://oceanprotocol.com/protocol/`. Accessed: 2019-03-25.

[28] Ocean Protocol Foundation with BigchainDB GmbH and Newton Circus (DEX Pte. Ltd.). Ocean protocol: A decentralized substrate for ai data services technical whitepaper, cited March 2019. URL `https://oceanprotocol.com/tech-whitepaper.pdf`.

[29] Wibson: Don't give away your data for free. make a profit. `https://wibson.org/`. Accessed: 2019-03-25.

[30] Matias Travizano, Martin Minnoni, Gustavo Ajzenman, Carlos Sarraute, Nicolas Della Penna . Wibson: A decentralized marketplace empowering individuals to safely monetize their personal data, cited March 2019. URL `https://wibson.org/wp-content/uploads/2018/10/Wibson-Technical-Paper-v1.1.pdf`.

[31] Wibson: Don't give away your data for free. make a profit. `https://wibson.org/`. Accessed: 2019-03-25.

[32] Roger Haenni. Datum network: The decentralized data marketplace, cited March 2019. URL `https://datum.org/assets/Datum-WhitePaper.pdf`.

[33] R. Guerraoui P. Blanchard, E. M. El Mhamdi and J. Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. 2017.

[34] BLOCKCHAIN LUXEMBOURG S.A. Average bitcoin block size, line chart, 2019. URL `https://www.blockchain.com/en/charts/avg-block-size`.

[35] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[36] Joao Sousa, Alysson Bessani, and Marko Vukolic. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 51–58. IEEE, 2018.

[37] Lakshmi Siva Sankar, M Sindhu, and M Sethumadhavan. Survey of consensus protocols on blockchain applications. In *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1–5. IEEE, 2017.

[38] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[39] Yiqing Hua Deborah Estrin Vitaly Shmatikov Eugene Bagdasaryan, Andreas Veit. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459v2*, 2018.

[40] Liangzhen Lai Naveen Suda Damon Civin Vikas Chandra Yue Zhao, Meng Li. Federated learning with non-iid data. 2018.

[41] Prateek Mittal Seraphin Calo Arjun Nitin Bhagoji, Supriyo Chakraborty. Analyzing federated learning through an adversarial lens. 2018.

[42] Ramesh Raskar Otkrist Gupta Abhimanyu Dubey Praneeth Vepakomma, Tristan Swedish. No peek: A survey of private distributed deep learning. 2018.